Deep Learning Tutorial

The National Conference on Communications (NCC), 2017

March 2, 2017

Mitesh M. Khapra

Teaching Assistants

Preksha

Poonam

Ditty

Rupam

Dilip

Beethika

Shreyas

Ayesha

Siddharth Arora

Sidharth Bafna

Ashish

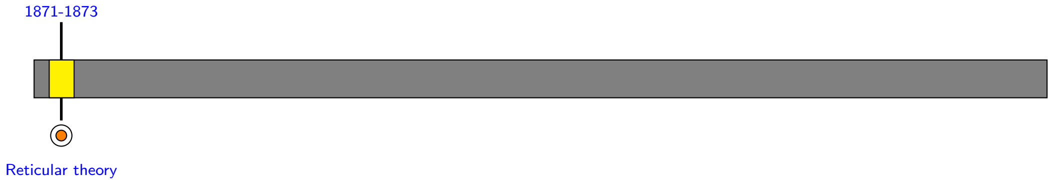
Hardik

A brief history of Deep Learning

Chapter 1: Biological Neurons

Reticular Theory

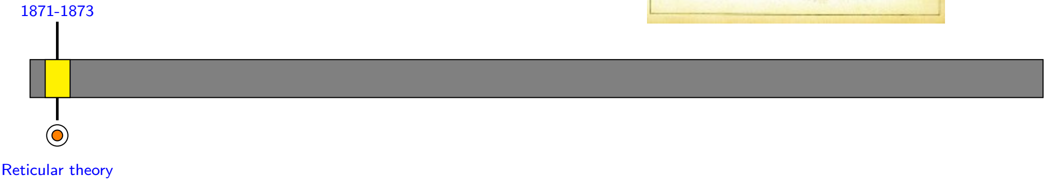
Joseph von Gerlach proposed that the nervous system is a single continuous network as opposed to a network of many discrete cells!



Staining Technique

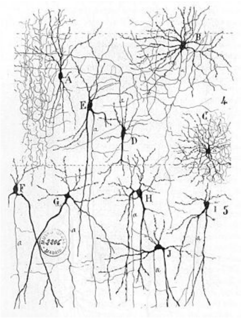
Camillo Golgi discovered a chemical reaction that allowed him to examine nervous tissue in much greater detail than ever before

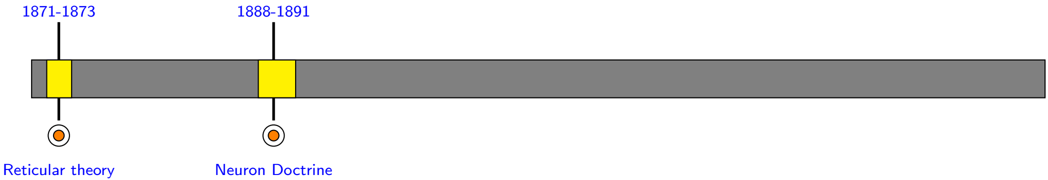
He was a proponent of Reticular theory.



Neuron Doctrine

Santiago Ramón y Cajal used Golgi's technique to study the nervous system and proposed that it is actually made up of discrete individual cells formimg a network (as opposed to a single continuous network)

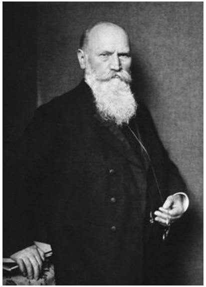


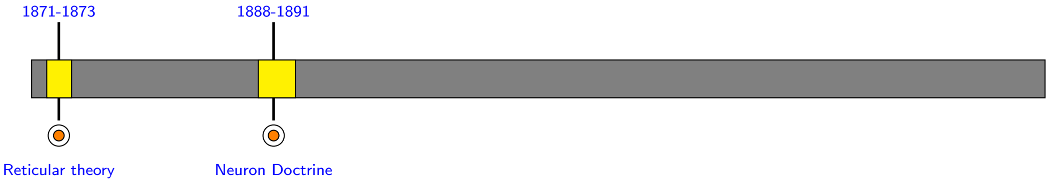


The Term Neuron

The term neuron was coined by Heinrich Wilhelm Gottfried von Waldeyer-Hartz around 1891.

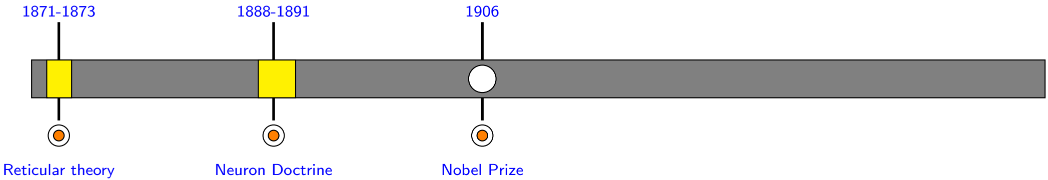
He further consolidated the Neuron Doctrine.





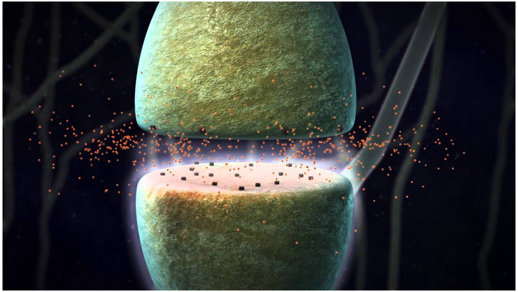
Nobel Prize

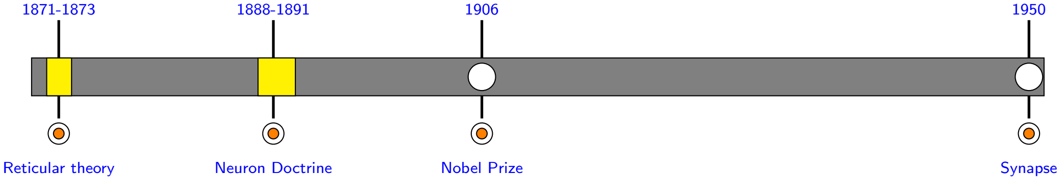
Both Golgi (reticular theory) and Cajal (neuron doctrine) were jointly awarded the 1906 Nobel Prize for Physiology or Medicine, that resulted in lasting conflicting ideas and controversies between the two scientists.



The Final Word

In 1950s electron microscopy finally confirmed the neuron doctrine by unambiguously demonstrated that nerve cells were individual cells interconnected through synapses (a network of many individual neurons).

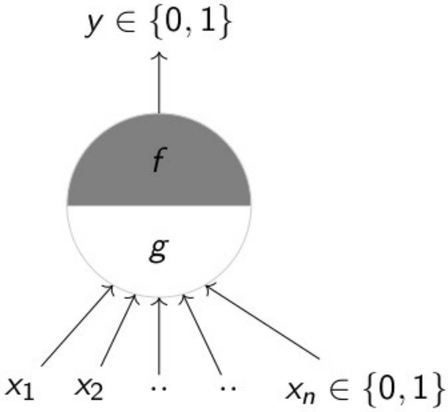


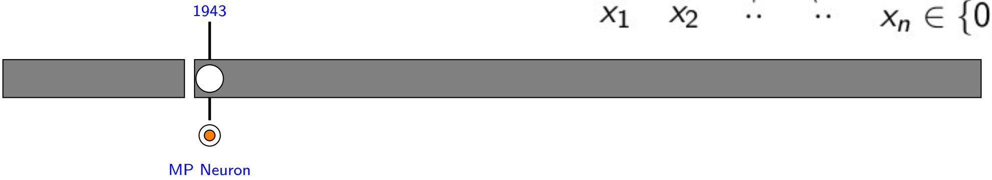


Chapter 2: From Spring to Winter

McCulloch Pitts Neuron

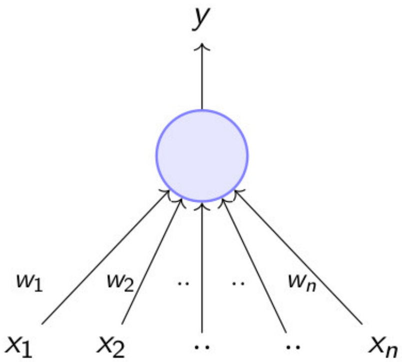
McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943)

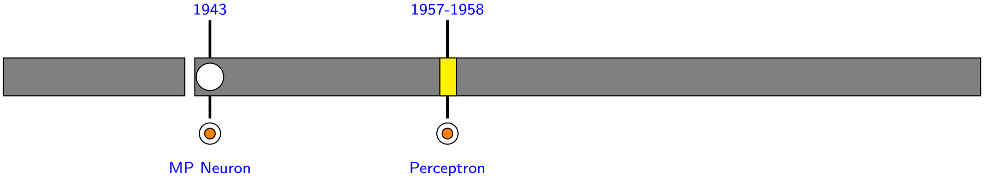




Perceptron

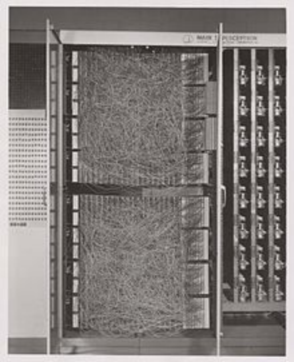
"the perceptron may eventually be able to learn, make decisions, and translate languages" -Frank Rosenblatt

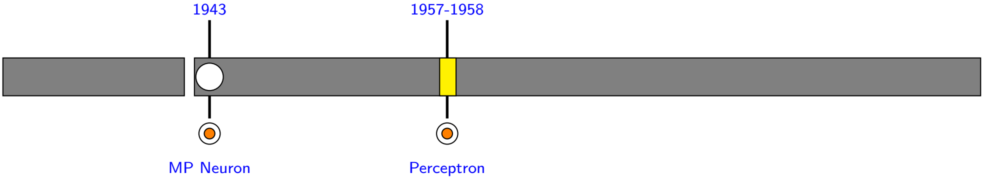




Perceptron

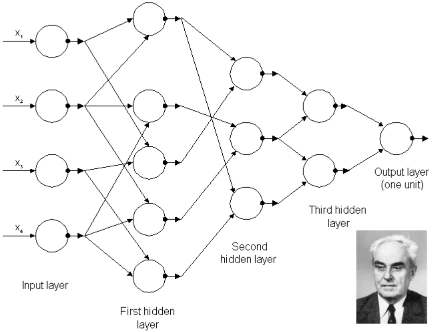
"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." -New York Times

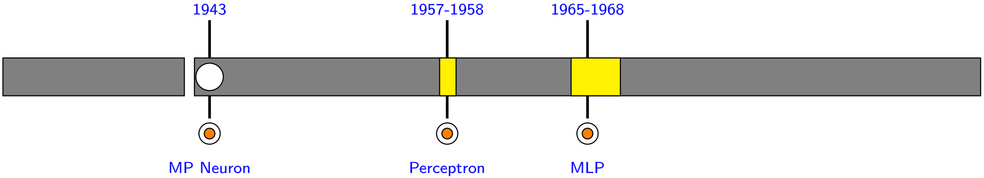




First generation Multilayer Perceptrons

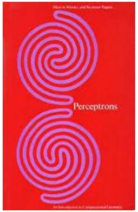
Ivakhnenko et. al.

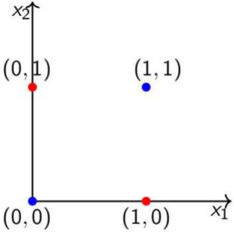


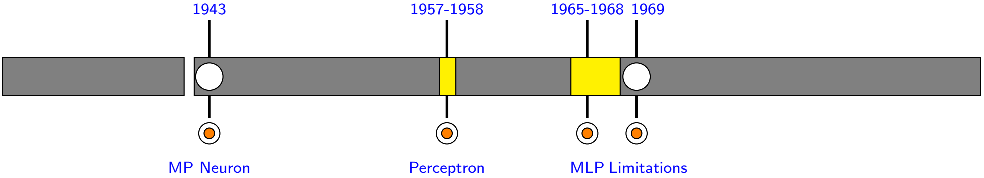


Perceptron Limitations

In their now famous book "Perceptrons", Minsky and Papert outlined the limits of what perceptrons could do

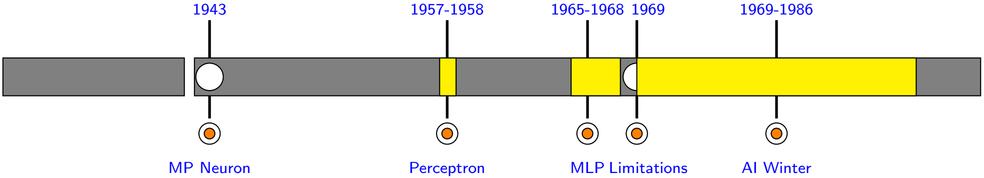






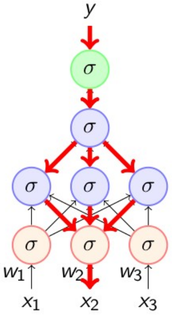
AI Winter of connectionism

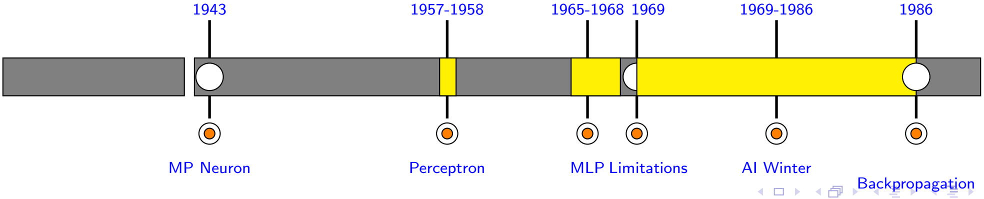
Almost lead to the abandonment of connectionist AI



Backpropagation

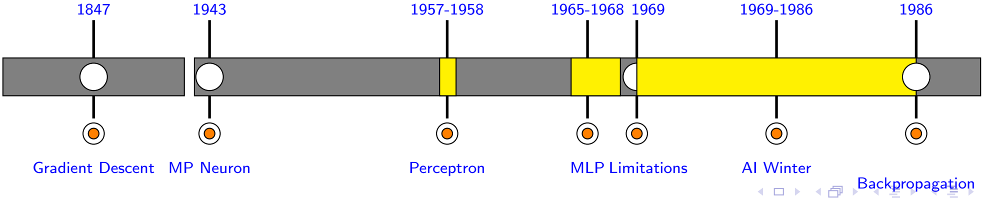
- Discovered and rediscovered several times throughout 1960's and 1970's
- Werbos [1982] first used it in the context of artificial neural networks
- Eventually popularized by the work of Rumelhart et. al. in 1986





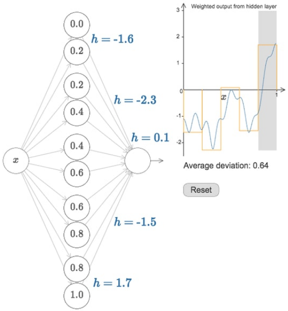
Gradient Descent

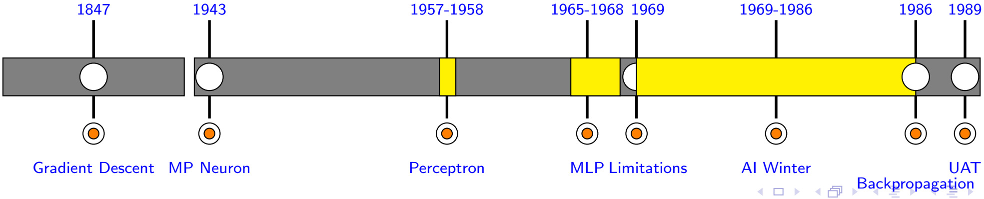
Cauchy discovered Gradient Descent motivated by the need to compute the orbit of heavenly bodies



Universal Approximation Theorem

A multilayered network of neurons with a single hidden layer can be used to approximate any continuous function to any desired precision

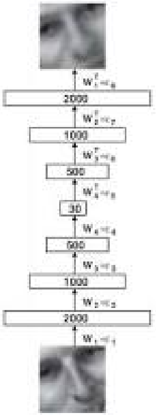




Chapter 3: The Deep Revival

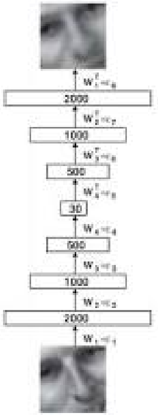
Unsupervised Pre-Training

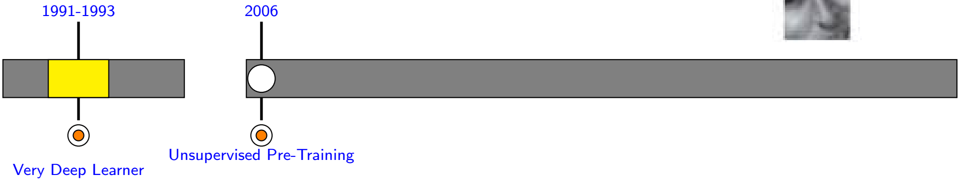
Hinton and Salakhutdinov described an effective way of initializing the weights that allows deep autoencoder networks to learn a low-dimensional representation of data.



Unsupervised Pre-Training

The idea of unsupervised pre-training actually dates back to 1991-1993 (J. Schmidhuber) when it was used to train a "Very Deep Learner"





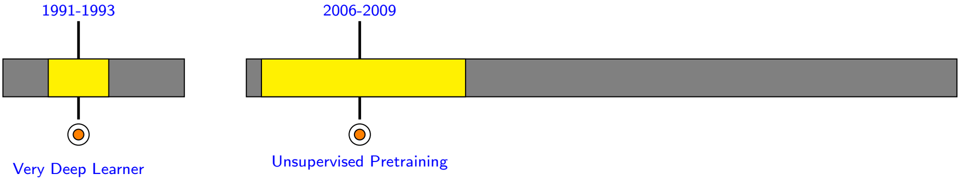
More insights (2007-2009)

Further Investigations into the effectiveness of Unsupervised Pre-training

Greedy Layer-Wise Training of Deep Networks

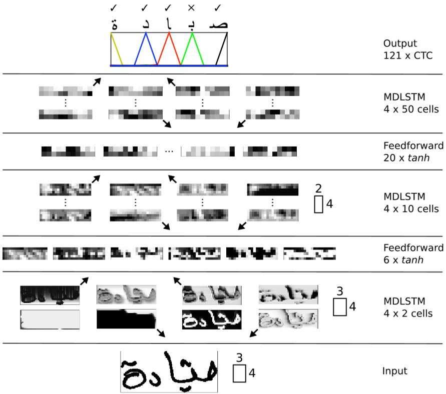
Why Does Unsupervised Pre-training Help Deep Learning?

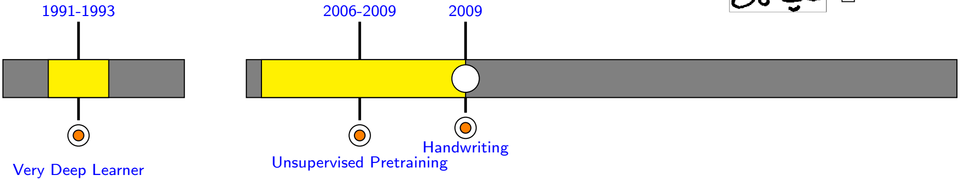
Exploring Strategies for Training Deep Neural Networks



Success in Handwriting Recognition

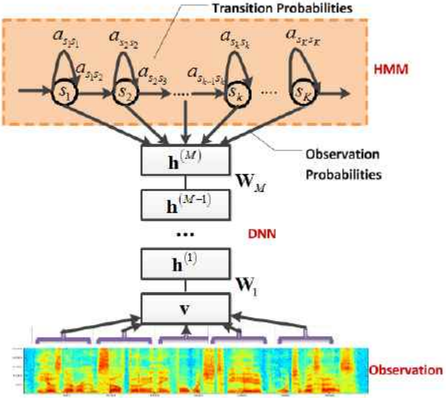
Graves et. al. outperformed all entries in an international Arabic recognition competition

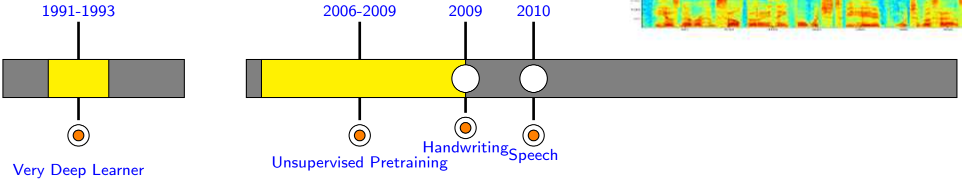




Success in Speech Recognition

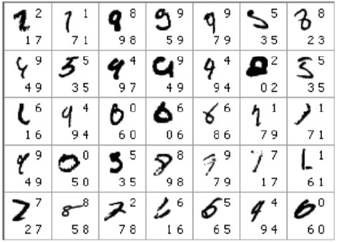
Dahl et. al. showed relative error reduction of 16.0% and 23.2% over a state of the art system

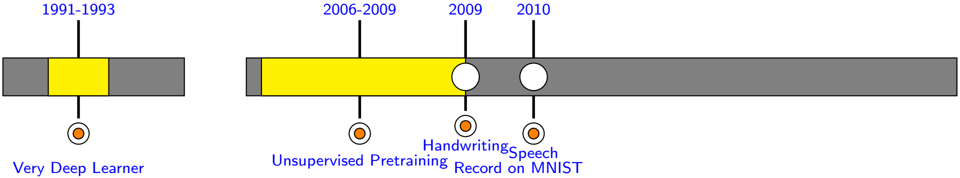




New record on MNIST

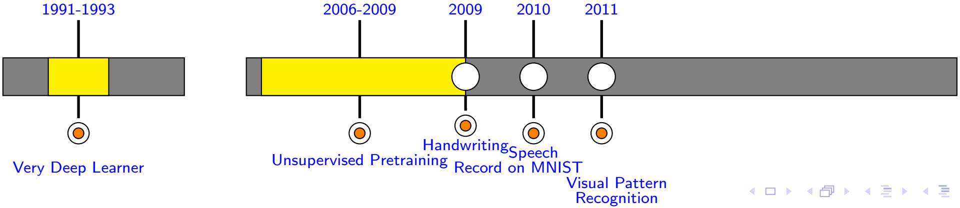
Ciresan et. al. set a new record on the MNIST dataset using good old backpropagation on GPUs (GPUs enter the scene)





First Superhuman Visual Pattern Recognition

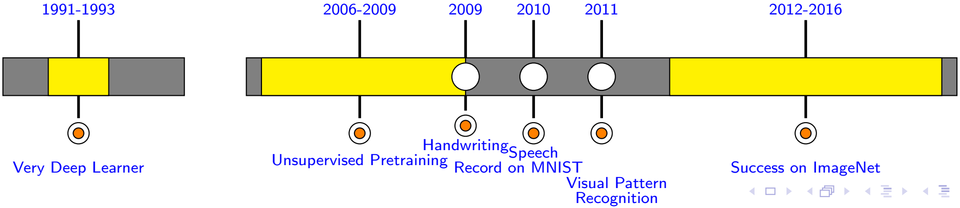
D. C. Ciresan et. al. achieved 0.56% error rate in the IJCNN Traffic Sign Recognition Competition



Winning more visual recognition challenges

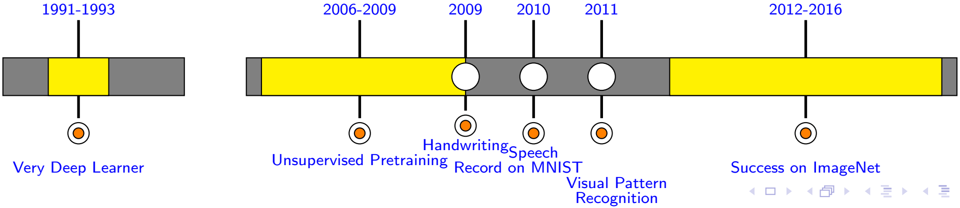
Network	Error	Layers
AlexNet	16.0%	8

1



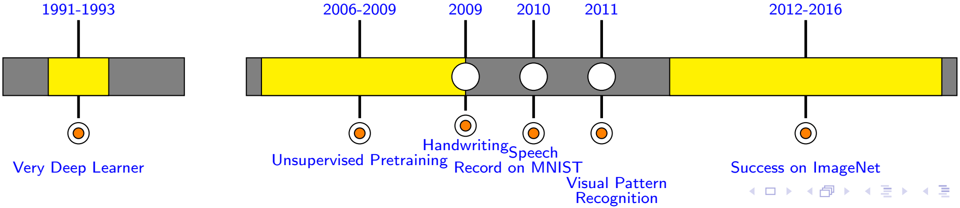
Winning more visual recognition challenges

Network	Error	Layers
AlexNet	16.0%	8
ZFNet	11.2%	8
		1



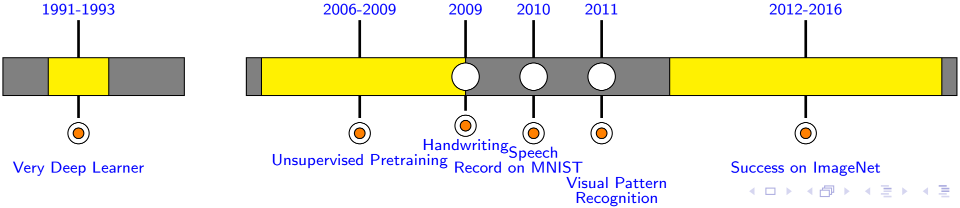
Winning more visual recognition challenges

Network	Error	Layers
AlexNet	16.0%	8
ZFNet	11.2%	8
VGGNet	7.3%	19



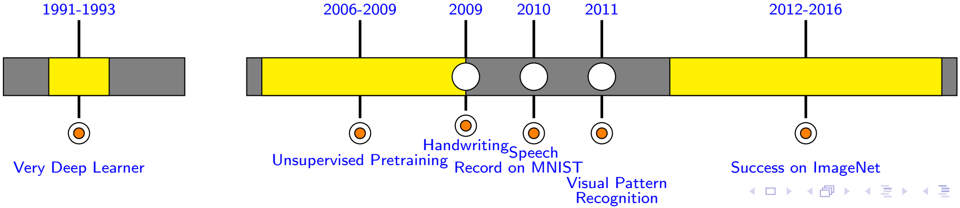
Winning more visual recognition challenges

Network	Error	Layers
AlexNet	16.0%	8
ZFNet	11.2%	8
VGGNet	7.3%	19
GoogLeNet	6.7%	22



Winning more visual recognition challenges

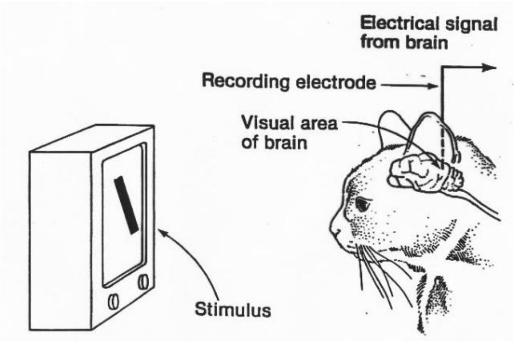
Network	Error	Layers
AlexNet	16.0%	8
ZFNet	11.2%	8
VGGNet	7.3%	19
GoogLeNet	6.7%	22
MS ResNet	3.6%	152!!



Chapter 4: Cats

Hubel and Wiesel Experiment

Experimentally showed that each neuron has a fixed receptive field - i.e. a neuron will fire only in response to a visual stimuli in a specific region in the visual space

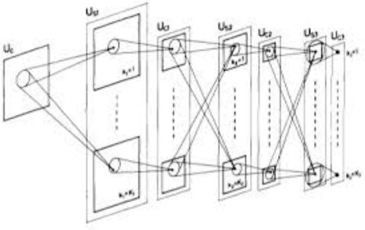


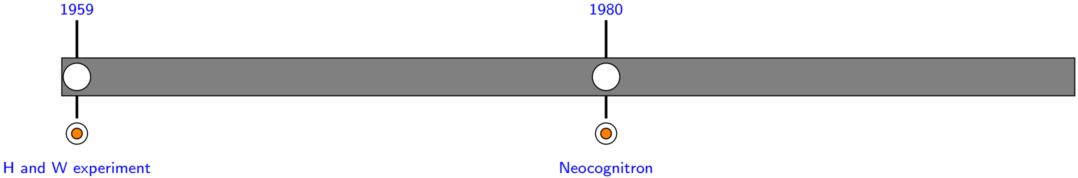
1959

H and W experiment

Neocognitron

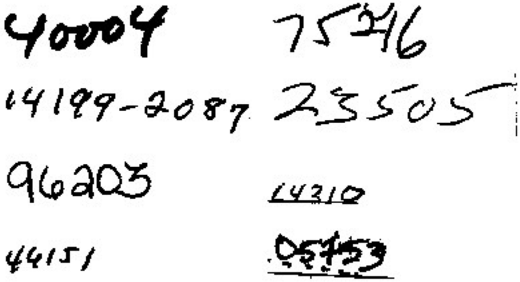
Used for Handwritten character recognition and pattern recognition (Fukushima et. al.)

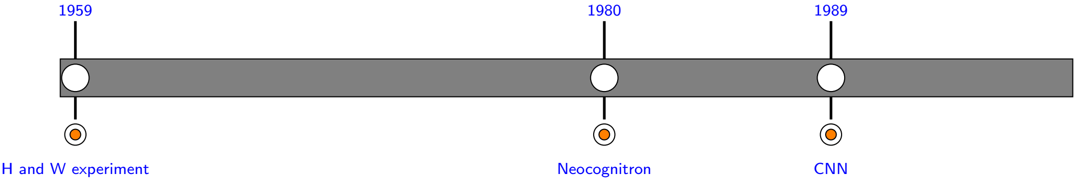




Convolutional Neural Network

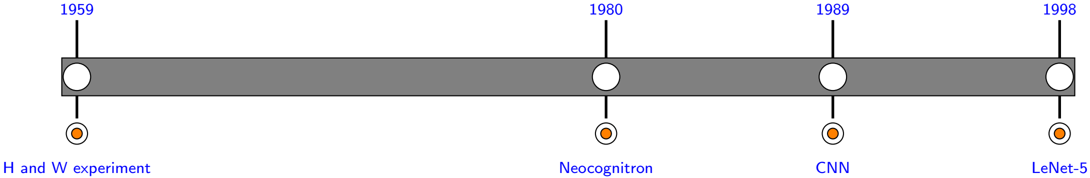
Handwriting digit recognition using back-propagation over a Convolutional Neural Network (LeCun et. al.)





LeNet-5

Introduced the (now famous) MNIST dataset (LeCun et. al.)

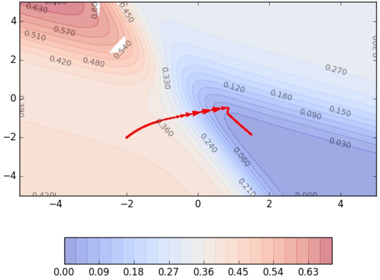


An algorithm inspired by an experiment on cats is today used to detect cats in videos :-)

Chapter 5: Faster, higher, stronger

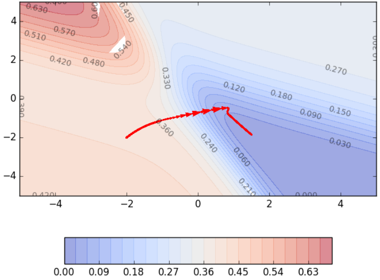
Better Optimization Methods

Faster convergence, better accuracies



Better Optimization Methods

Faster convergence, better accuracies



1983

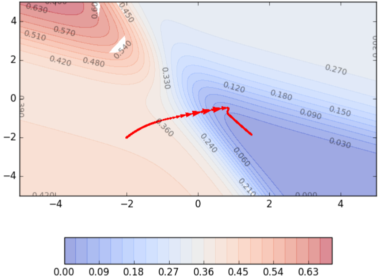
Nesterov

2011

Adagrad

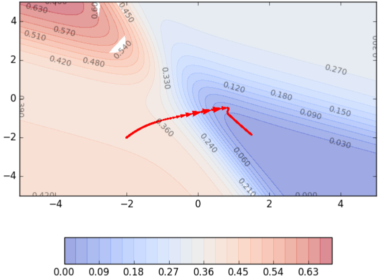
Better Optimization Methods

Faster convergence, better accuracies



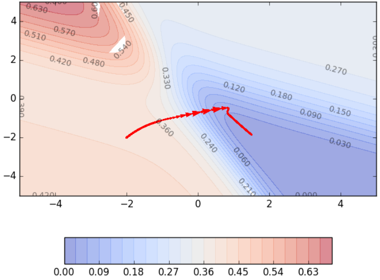
Better Optimization Methods

Faster convergence, better accuracies



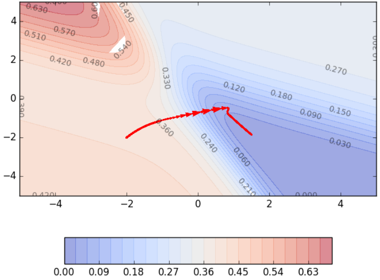
Better Optimization Methods

Faster convergence, better accuracies



Better Optimization Methods

Faster convergence, better accuracies



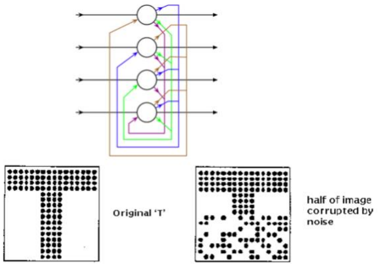
Chapter 6: The Curious Case of Sequences

Sequences

- They are everywhere
- Time series, speech, music, text, video
- Each unit in the sequence interacts with other units
- Need models to capture this interaction

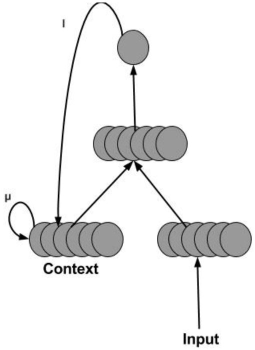
Hopfield Network

Content-addressable memory systems for storing and retrieving patterns



Jordan Network

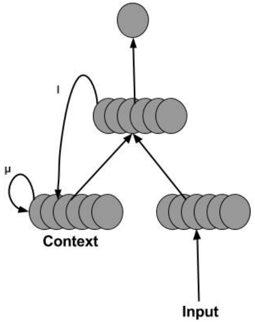
The output state of each time step is fed to the next time step thereby allowing interactions between time steps in the sequence



Jordan

Elman Network

The hidden state of each time step is fed to the next time step thereby allowing interactions between time steps in the sequence



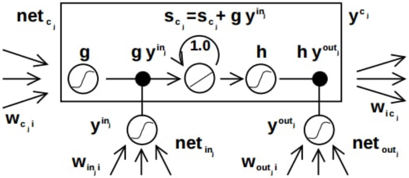
Elman

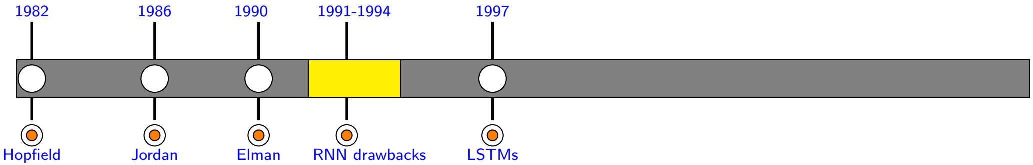
Drawbacks of RNNs

Hochreiter et. al. and Bengio et. al. showed the difficulty in training RNNs (the problem of exploding and vanishing gradients)

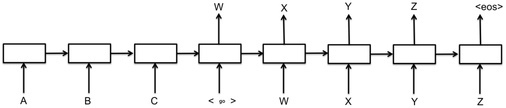
Long Short Term Memory

Showed that LSTMs can solve complex long time lag tasks that could never be solved before

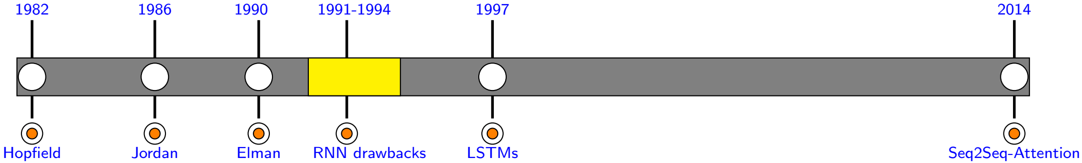




Sequence To Sequence Learning

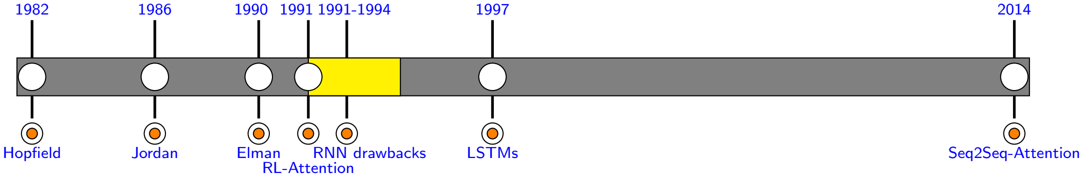


- Initial success in using RNNs/LSTMs for large scale Sequence To Sequence Learning Problems
- Introduction of Attention which inspired a lot of research over the next two years



RL for Attention

Schmidhuber & Huber proposed RNNs that use reinforcement learning to decide where to look



Chapter 7: The Madness (2013-2016)

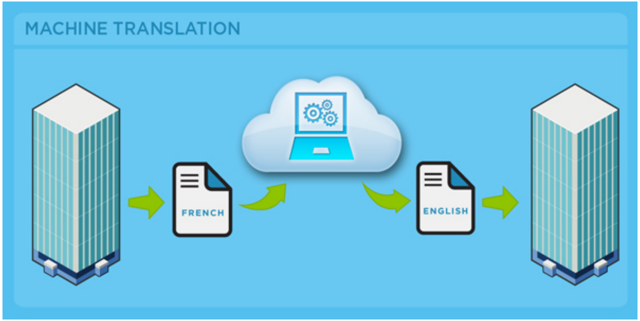
He sat on a <u>chair</u>.

Language Modeling

- Mikolov et al. (2010)
- Li et al. (2015)
- Kiros et al. (2015)
- Kim et al. (2015)

Speech Recognition

- Hinton et al. (2012)
- Graves et al. (2013)
- Chorowski et al. (2015)
- Sak et al. (2015)



Machine Translation

- Kalchbrenner et al. (2013)
- Cho et al. (2014)
- Bahdanau et al. (2015)
- Jean et al. (2015)
- Gulcehre et al. (2015)
- Sutskever et al. (2014)
- Luong et al. (2015)

Time	User	Utterance
03:44	Old	I dont run graphical ubuntu, I run ubuntu server.
03:45	kuja	Taru: Haha sucker.
03:45	Taru	Kuja: ?
03:45	bur[n]er	Old: you can use "ps ax" and "kill (PID#)"
03:45	kuja	Taru: Anyways, you made the changes right?
03:45	Taru	Kuja: Yes.
03:45	LiveCD	or killall speedlink
03:45	kuja	Taru: Then from the terminal type: sudo apt-get update
03:46	_pm	if i install the beta version, how can i update it when the final version comes out?
03:46	Taru	Kuja: I did.

Sender	Recipient	Utterance
Old		I dont run graphical ubuntu, I run ubuntu server.
bur[n]er	Old	you can use "ps ax" and "kill (PID#)"
kuja	Taru	Haha sucker.
Taru	Kuja	?
kuja	Taru	Anyways, you made the changes right?
Taru	Kuja	Yes.
kuja	Taru	Then from the terminal type: sudo apt-get update
Taru	Kuja	I did.

Conversation Modeling

- Shang et al. (2015)
- Vinyals et al. (2015)
- Lowe et al. (2015)
- Dodge et al. (2015)
- Weston et al. (2016)

Task 1: Single Supporting Fact
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

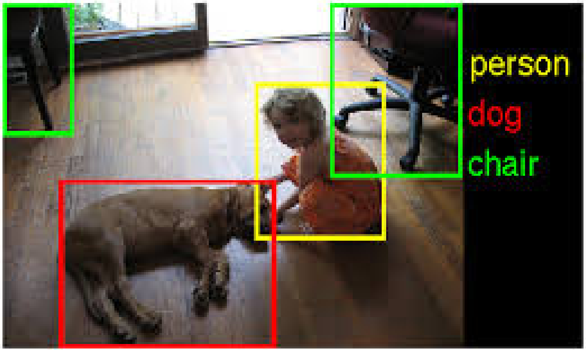
Task 2: Two Supporting Facts
John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

Task 3: Three Supporting Facts
John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Where was the apple before the kitchen? A:office

Task 4: Two Argument Relations
The office is north of the bedroom.
The bedroom is north of the bathroom.
The kitchen is west of the garden.
What is north of the bedroom? A: office
What is the bedroom north of? A: bathroom

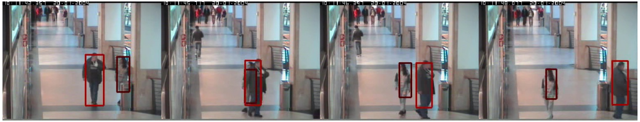
Question Answering

- Weston et al. (2015)
- Bordes et al. (2015)
- Hill et al. (2016)
- Hermann et al. (2015)
- Kumar et al. (2016)



Object Recognition

- Pinheiro et al. (2015)
- Liang et al. (2015)
- Byeon et al. (2015)
- Serban et al. (2015)
- Zheng et al. (2015)
- Liang et al. (2015)
- Bell et al. (2015)



Visual Tracking

- Gan et al. (2015)
- Gregor et al. (2015)
- Lazaridou et al. (2015)
- Theis et al. (2015)
- Van et al. (2016)

Retr.	1. Top view of the lights of a city at night, with a well-illuminated square in front of a church in the foreground; 2. People on the stairs in front of an illuminated cathedral with two towers at night;	1. Tourists are sitting at a long table with beer bottles on it in a rather dark restaurant and are raising their bierglaeser; 2. Tourists are sitting at a long table with a white table-cloth in a somewhat dark restaurant;
Gen.	A square with burning street lamps and a street in the foreground;	Tourists are sitting at a long table with a white table cloth and are eating;

Image Captioning

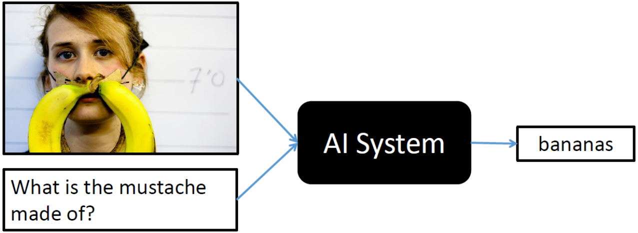
- Mao et al. (2014)
- Mao at al. (2015)
- Kiros et al. (2015)
- Donahue et al. (2015)
- Vinyals et al. (2015)
- Karpathy et al. (2015)
- Fang et al. (2015)
- Chen et al. (2015)

A group of young men playing a game of soccer

A man riding a wave on top of a surfboard.

Video Captioning

- Donahue et al. (2014)
- Venugopalan at al. (2014)
- Pan et al. (2015)
- Yao et al. (2015)
- Rohrbach et al. (2015)
- Zhu et al. (2015)
- Cho et al. (2015)



Visual Question Answering

- Antol et al. (2014)
- Malinowski at al. (2015)
- Ren et al. (2015)
- Gao et al. (2015)
- Kim et al. (2016)
- Fukui et al. (2016)
- Noh et al. (2016)
- Tapaswi et al. (2015)

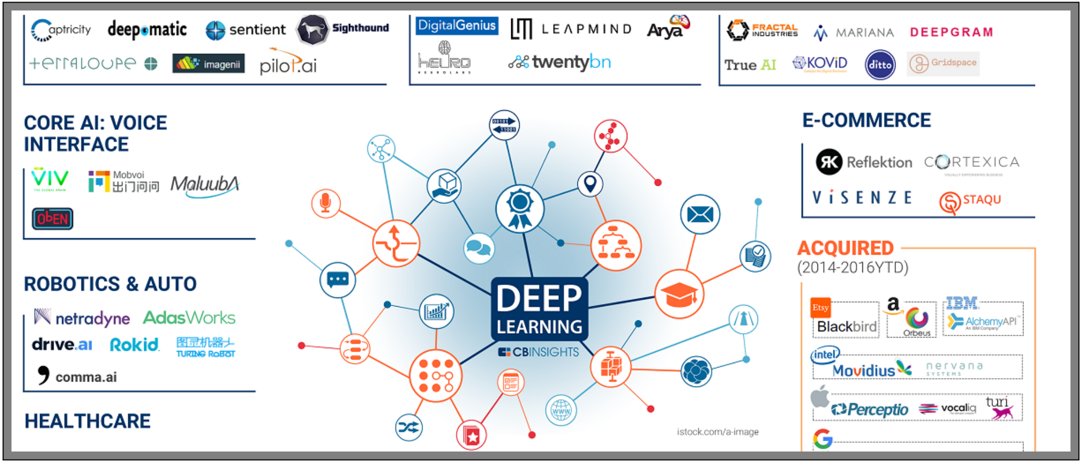
Generating Authentic Photos

- Generative Adversarial Networks (Goodfellow et. al., 2014)
- Variational Autoencoders (Kingma et. al., 2013)

Generating Raw Audio

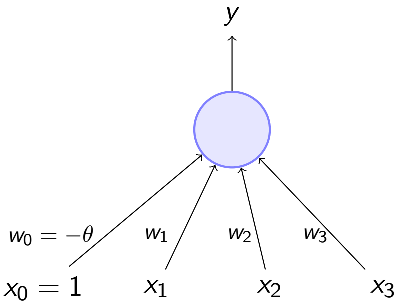
- Wavenets (Oord et. al., 2016)

http://blog.xukui.cn/awesome-recurrent-neural-networks/

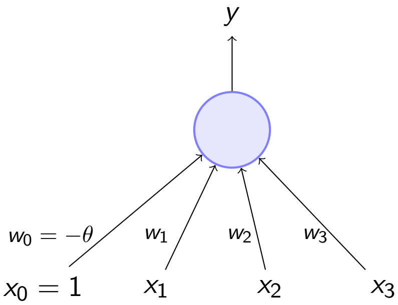


[1] Source: https://www.cbinsights.com/blog/deep-learning-ai-startups-market-map-company-list/

Training Deep Neural Networks

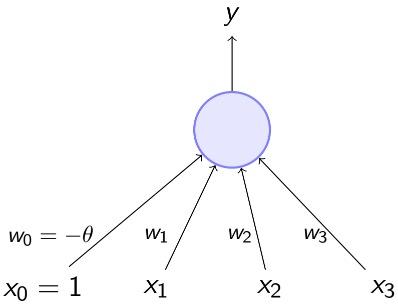


- Consider the task of predicting whether we would like a movie or not



$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

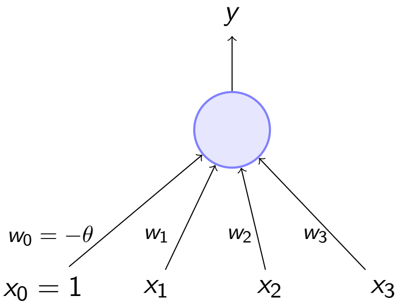
- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)



- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

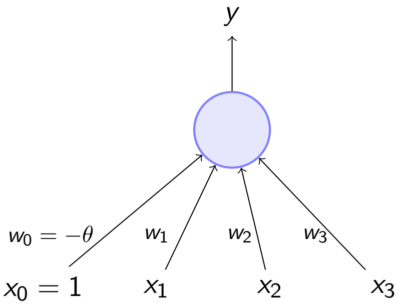
$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$



$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta \\
&= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta
\end{aligned}
$$

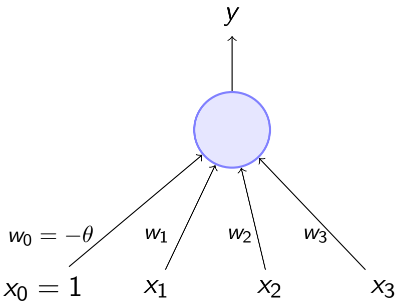


$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta \\
&= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta
\end{aligned}
$$

- Based on our past viewing experience (**data**), we may give a high weight to *isDirectorNolan* as compared to the other inputs

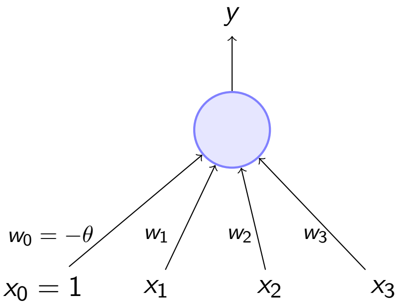


$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

- Based on our past viewing experience (**data**), we may give a high weight to *isDirectorNolan* as compared to the other inputs
- Specifically, even if the actor is not *Matt Damon* and the genre is not *thriller* we would still want to cross the threshold θ by assigning a high weight to *isDirectorNolan*

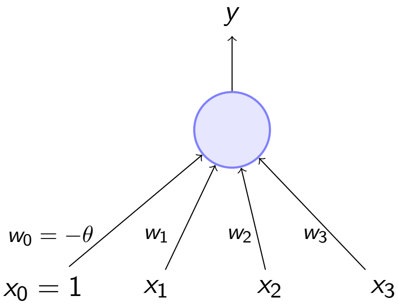


$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)

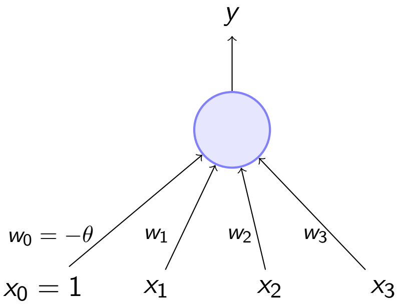
$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

- Based on our past viewing experience (**data**), we may give a high weight to *isDirectorNolan* as compared to the other inputs
- Specifically, even if the actor is not *Matt Damon* and the genre is not *thriller* we would still want to cross the threshold θ by assigning a high weight to *isDirectorNolan*



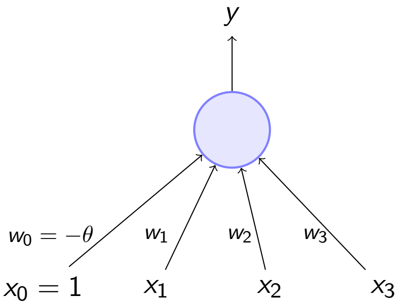
$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

- w_0 is called the bias as it represents the prior (prejudice)



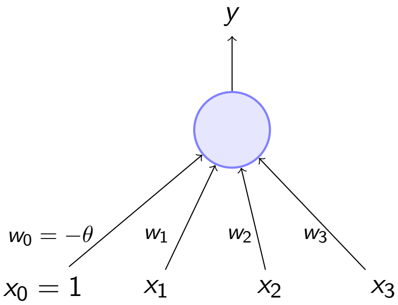
$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

- w_0 is called the bias as it represents the prior (prejudice)
- A movie buff may have a very low threshold and may watch any movie irrespective of the genre, actor, director $[\theta = 0]$



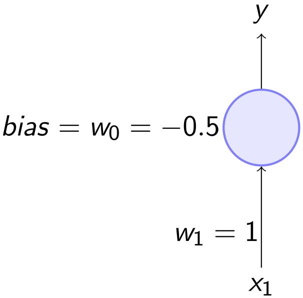
$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

- w_0 is called the bias as it represents the prior (prejudice)
- A movie buff may have a very low threshold and may watch any movie irrespective of the genre, actor, director [$\theta = 0$]
- On the other hand a selective viewer, may only watch thrillers starring Matt Damon and directed by Nolan [$\theta = 3$]

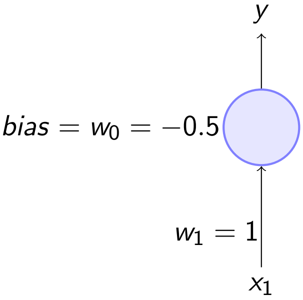


$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

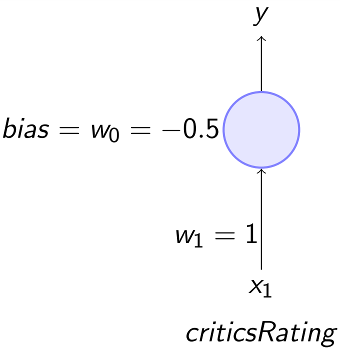
- w_0 is called the bias as it represents the prior (prejudice)
- A movie buff may have a very low threshold and may watch any movie irrespective of the genre, actor, director [$\theta = 0$]
- On the other hand a selective viewer, may only watch thrillers starring Matt Damon and directed by Nolan [$\theta = 3$]
- The weights ($w_1, w_2, ..., w_n$) and the bias (w_0) will depend on the data (viewer history in this case)



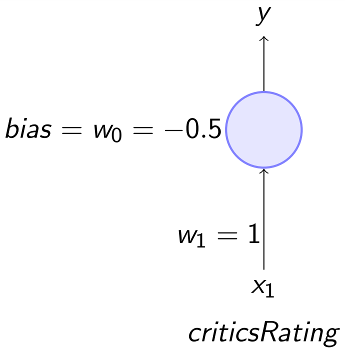
- Notice that the thresholding logic used by a perceptron is very harsh !



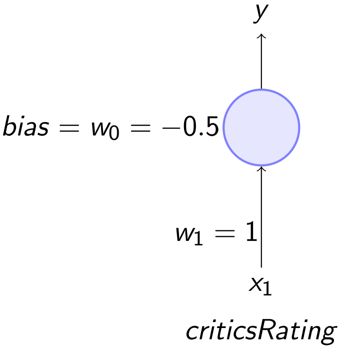
- Notice that the thresholding logic used by a perceptron is very harsh !
- For example, consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)



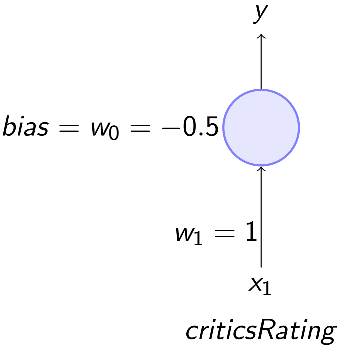
- Notice that the thresholding logic used by a perceptron is very harsh !
- For example, consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)
- If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$?



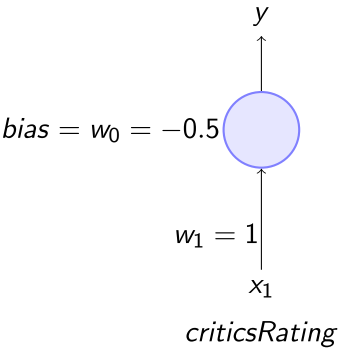
- Notice that the thresholding logic used by a perceptron is very harsh !
- For example, consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)
- If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$? (like)



- Notice that the thresholding logic used by a perceptron is very harsh !
- For example, consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)
- If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$? (like)
- What about a movie with $criticsRating = 0.49$?

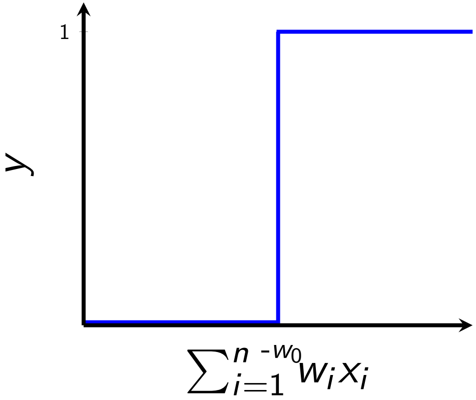


- Notice that the thresholding logic used by a perceptron is very harsh !
- For example, consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)
- If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$? (like)
- What about a movie with $criticsRating = 0.49$? (dislike)

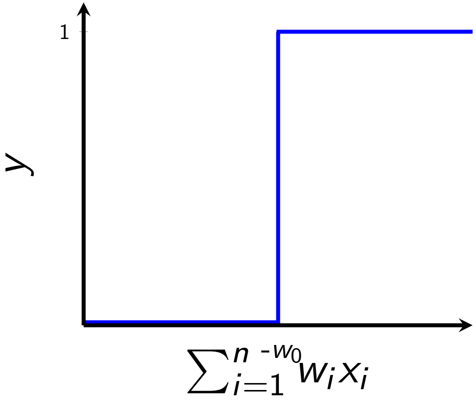


- Notice that the thresholding logic used by a perceptron is very harsh !
- For example, consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)
- If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$? (like)
- What about a movie with $criticsRating = 0.49$? (dislike)
- It seems harsh that we would like a movie with rating 0.51 but not one with a rating of 0.49

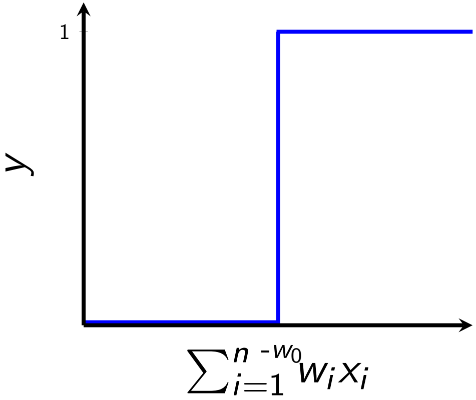
- This behavior is not a characteristic of the specific problem we chose or the specific weight and threshold that we chose



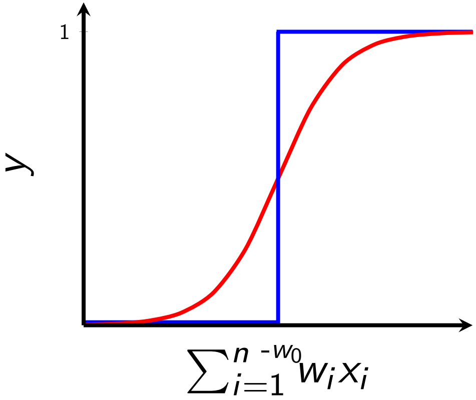
- This behavior is not a characteristic of the specific problem we chose or the specific weight and threshold that we chose
- It is a characteristic of the perceptron function itself which behaves like a step function



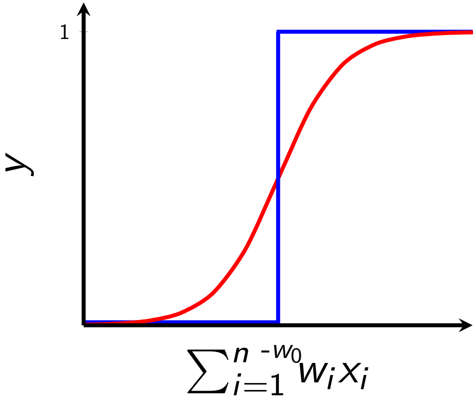
- This behavior is not a characteristic of the specific problem we chose or the specific weight and threshold that we chose
- It is a characteristic of the perceptron function itself which behaves like a step function
- There will always be this sudden change in the decision (from 0 to 1) when $\sum_{i=1}^{n} w_i x_i$ crosses the threshold (-w_0)



- This behavior is not a characteristic of the specific problem we chose or the specific weight and threshold that we chose
- It is a characteristic of the perceptron function itself which behaves like a step function
- There will always be this sudden change in the decision (from 0 to 1) when $\sum_{i=1}^{n} w_i x_i$ crosses the threshold (-w_0)
- For most real world applications we would expect a smoother decision function which gradually changes from 0 to 1

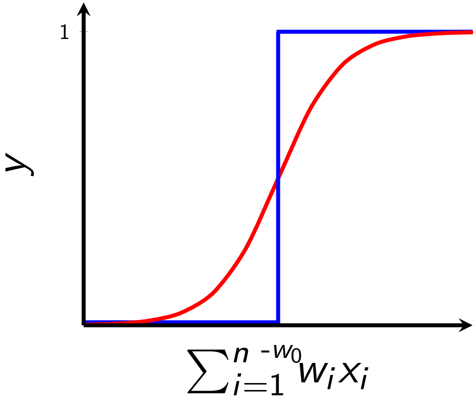


- Introducing sigmoid neurons where the output function is much smoother than the step function



- Introducing sigmoid neurons where the output function is much smoother than the step function
- Here is one form of the sigmoid function called the logistic function

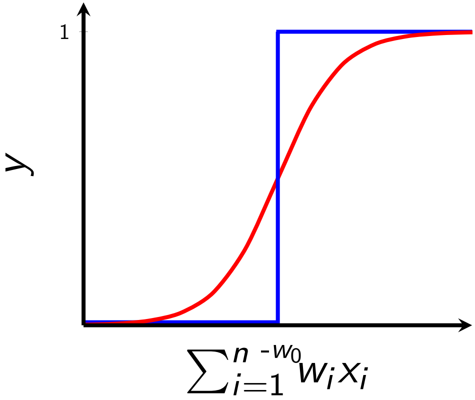
$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$



- Introducing sigmoid neurons where the output function is much smoother than the step function
- Here is one form of the sigmoid function called the logistic function

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$

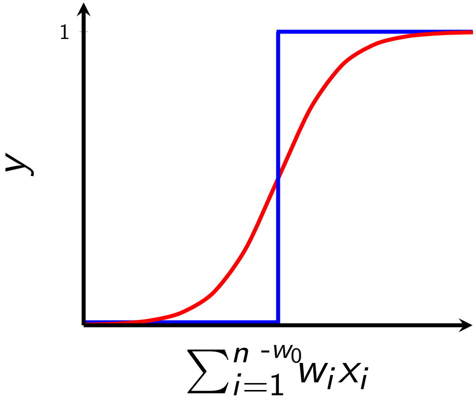
- We no longer see a sharp transition around the threshold $-w_0$



- Introducing sigmoid neurons where the output function is much smoother than the step function
- Here is one form of the sigmoid function called the logistic function

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$

- We no longer see a sharp transition around the threshold -w_0
- Also the output y is no longer binary but a real value between 0 and 1 which can be interpreted as a probability

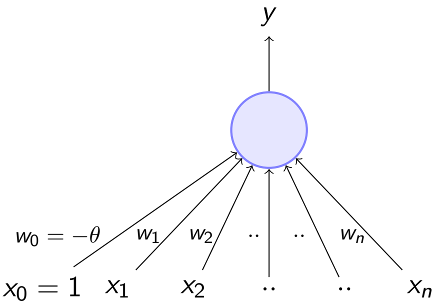


- Introducing sigmoid neurons where the output function is much smoother than the step function
- Here is one form of the sigmoid function called the logistic function

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$

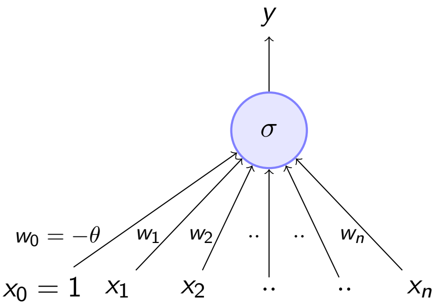
- We no longer see a sharp transition around the threshold $-w_0$
- Also the output y is no longer binary but a real value between 0 and 1 which can be interpreted as a probability
- Instead of a like/dislike decision we get the probability of liking the movie

Perceptron



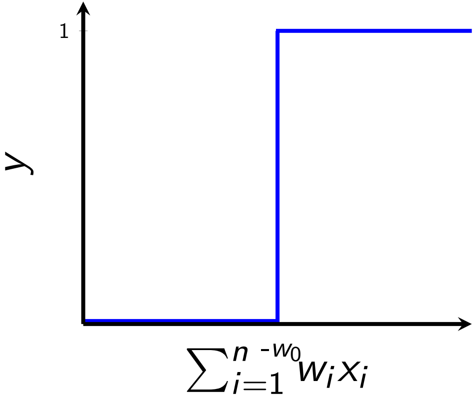
$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

Sigmoid (logistic) Neuron



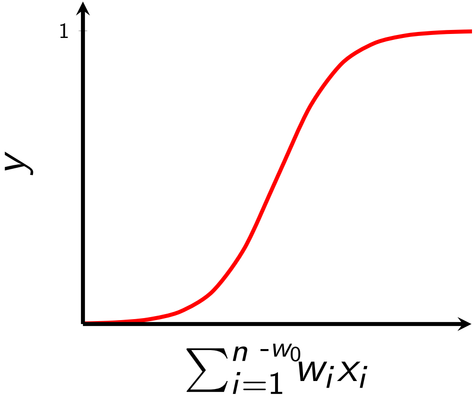
$$y = \frac{1}{1 + e^{-(\sum_{i=0}^{n} w_i x_i)}}$$

Perceptron



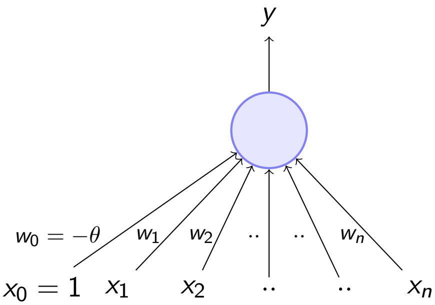
Not smooth, not continuous (at $w0$), **not differentiable**

Sigmoid Neuron



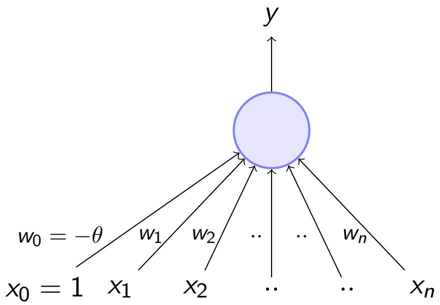
Smooth, continuous, **differentiable**

Sigmoid (logistic) Neuron



- What next ?

Sigmoid (logistic) Neuron



- What next ?
- Well, we also need a way of learning the weights of a sigmoid neuron

This brings us to a typical machine learning setup which has the following components...

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

$$or \quad \hat{y} = (w^T x)^2$$

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^n$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

$$or \quad \hat{y} = (w^T x)^2$$

or just about any function

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$
$$or \quad \hat{y} = w^T x$$
$$or \quad \hat{y} = (w^T x)^2$$

  or just about any function
- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^n$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$
$$or \quad \hat{y} = w^T x$$
$$or \quad \hat{y} = (w^T x)^2$$

  or just about any function

- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data
- **Learning algorithm:** An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$
$$or \quad \hat{y} = w^T x$$
$$or \quad \hat{y} = (w^T x)^2$$

  or just about any function
- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data
- **Learning algorithm:** An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)
- **Objective/Loss/Error function:** To guide the learning algorithm

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

$$or \quad \hat{y} = (w^T x)^2$$

  or just about any function
- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data
- **Learning algorithm:** An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)
- **Objective/Loss/Error function:** To guide the learning algorithm - the learning algorithm should aim to minimize the loss function

As an illustration, consider our movie example

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]
- **Objective/Loss/Error function:**

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]
- **Objective/Loss/Error function:** One possibility is

$$\mathscr{L}(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

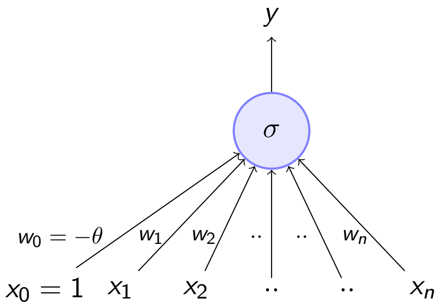
$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]
- **Objective/Loss/Error function:** One possibility is

$$\mathscr{L}(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

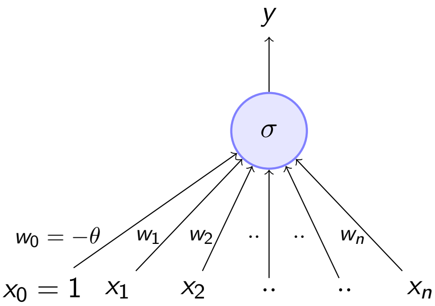
The learning algorithm should aim to find a w which minimizes the above function (squared error between y and $\hat{y}$)

Sigmoid (logistic) Neuron

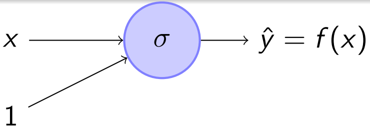


- With this setup in mind, we will now focus on this **model** and discuss an **algorithm** for learning the **parameters** of this model from some given **data**

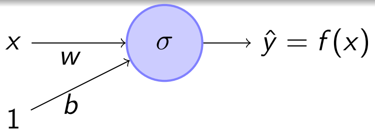
Sigmoid (logistic) Neuron



- With this setup in mind, we will now focus on this **model** and discuss an **algorithm** for learning the **parameters** of this model from some given **data**
- σ stands for the sigmoid function (logistic function in this case)

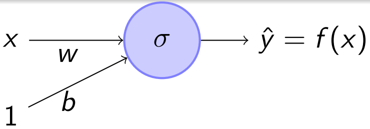


- With this setup in mind, we will now focus on this **model** and discuss an **algorithm** for learning the **parameters** of this model from some given **data**
- σ stands for the sigmoid function (logistic function in this case)
- For ease of explanation, we will consider a very simplified version of the model having just 1 input



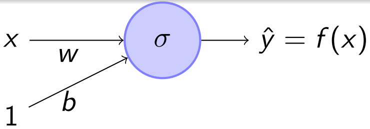
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

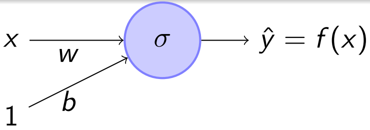
- With this setup in mind, we will now focus on this **model** and discuss an **algorithm** for learning the **parameters** of this model from some given **data**
- σ stands for the sigmoid function (logistic function in this case)
- For ease of explanation, we will consider a very simplified version of the model having just 1 input
- Further to be consistent with the literature, from now on we will refer to w_0 as b (bias)



$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

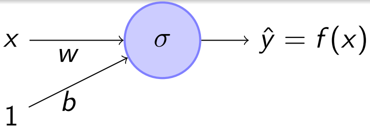
- With this setup in mind, we will now focus on this **model** and discuss an **algorithm** for learning the **parameters** of this model from some given **data**
- σ stands for the sigmoid function (logistic function in this case)
- For ease of explanation, we will consider a very simplified version of the model having just 1 input
- Further to be consistent with the literature, from now on we will refer to w_0 as b (bias)
- Lastly, instead of considering the problem of predicting like/dislike we will assume that we want to predict $criticsRating(y)$ given $imdbRating(x)$ (for no particular reason)


x
w
σ
ŷ = f(x)
1
b



Input for training

$\{x_i, y_i\}_{i=1}^{N} \rightarrow N$ pairs of (x, y)



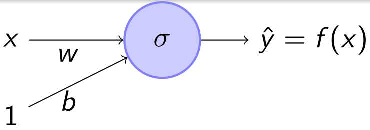
Input for training

$\{x_i, y_i\}_{i=1}^{N} \rightarrow N$ pairs of (x, y)

Training objective

Find w and b such that:

$$minimize_{w,b}\mathscr{L}(w, b) = \sum_{i=1}^{N}(y_i - f(x_i))^2$$



$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

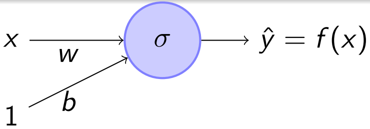
Input for training

$\{x_i, y_i\}_{i=1}^{N} \rightarrow N$ pairs of (x, y)

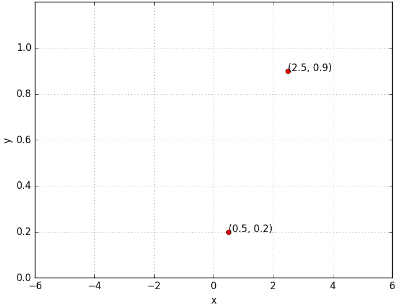
Training objective

Find w and b such that:

$$minimize_{w,b} \mathscr{L}(w, b) = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

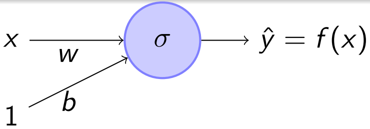


$$f(x) = \frac{1}{1+e^{-(w \cdot x+b)}}$$

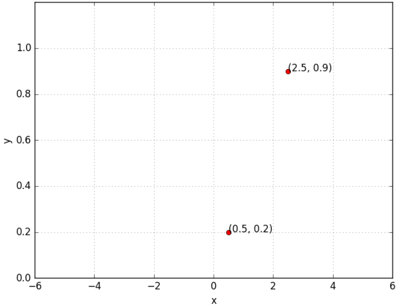


What does it mean to train the network?

- Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$

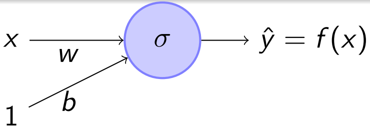


$$f(x) = \frac{1}{1+e^{-(w \cdot x+b)}}$$

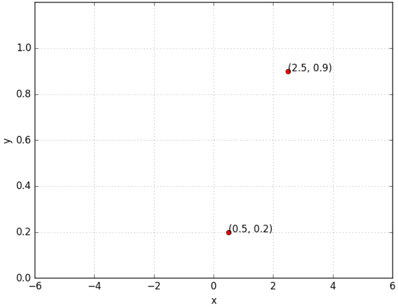


What does it mean to train the network?

- Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$
- At the end of training we expect to find w*, b* such that:

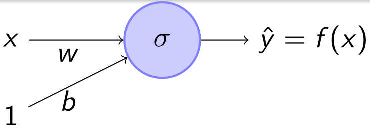


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

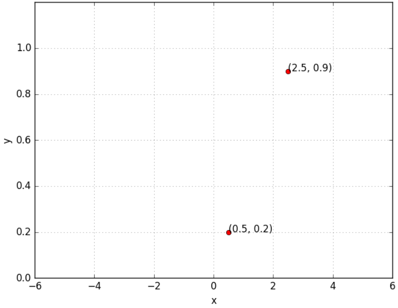


What does it mean to train the network?

- Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$
- At the end of training we expect to find w*, b* such that:
- $f(0.5) \rightarrow 0.2$ and $f(2.5) \rightarrow 0.9$



$$f(x) = \frac{1}{1+e^{-(w \cdot x+b)}}$$

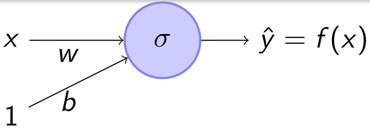


What does it mean to train the network?

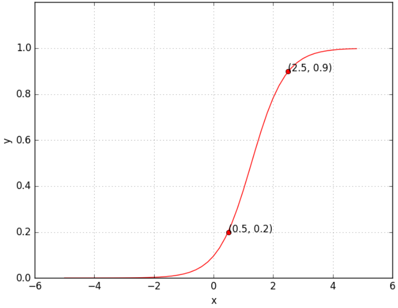
- Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$
- At the end of training we expect to find w*, b* such that:
- $f(0.5) \rightarrow 0.2$ and $f(2.5) \rightarrow 0.9$

In other words...

- We hope to find a sigmoid function such that $(0.5, 0.2)$ and $(2.5, 0.9)$ lie on this sigmoid



$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



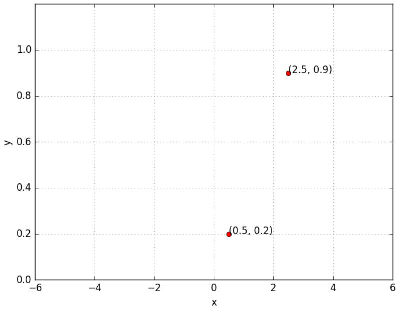
What does it mean to train the network?

- Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$
- At the end of training we expect to find w*, b* such that:
- $f(0.5) \rightarrow 0.2$ and $f(2.5) \rightarrow 0.9$

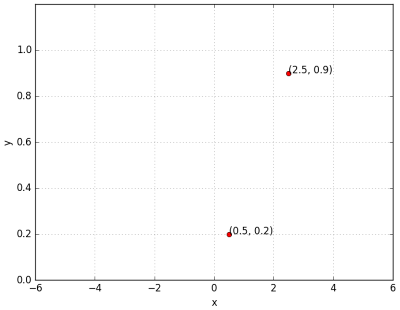
In other words...

- We hope to find a sigmoid function such that $(0.5, 0.2)$ and $(2.5, 0.9)$ lie on this sigmoid

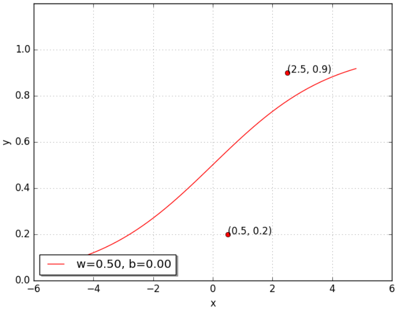
Let's see this in more detail....



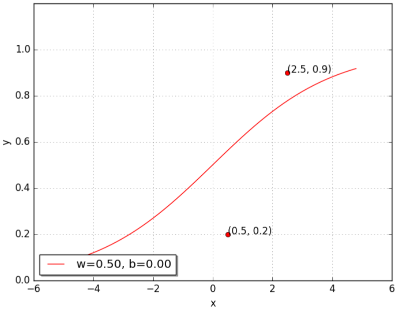
(2.5, 0.9)
(0.5, 0.2)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6



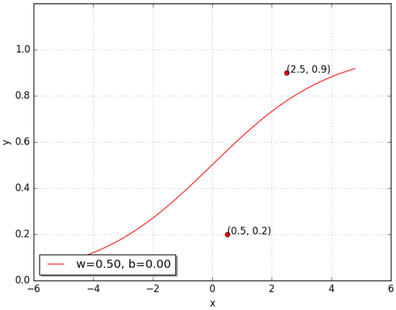
- Can we try to find such a $w*, b*$ manually



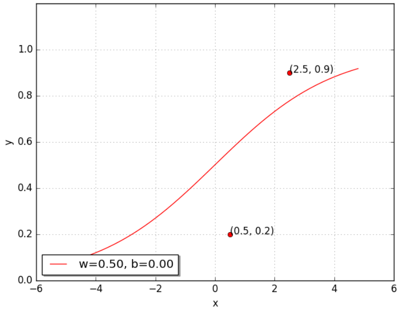
- Can we try to find such a $w*, b*$ manually
- Lets try a random guess.. (say, $w = 0.5, b = 0$)



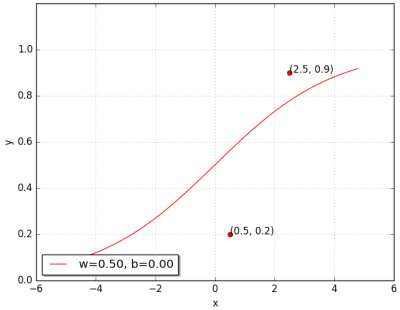
- Can we try to find such a $w*, b*$ manually
- Lets try a random guess.. (say, $w = 0.5, b = 0$)
- Clearly not good, but how bad is it ?



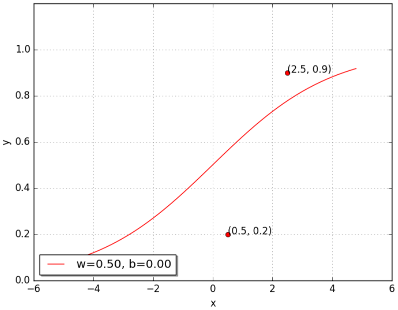
- Can we try to find such a $w*, b*$ manually
- Lets try a random guess.. (say, $w = 0.5, b = 0$)
- Clearly not good, but how bad is it ?
- Lets revisit $\mathscr{L}(w, b)$ to see how bad it is ...



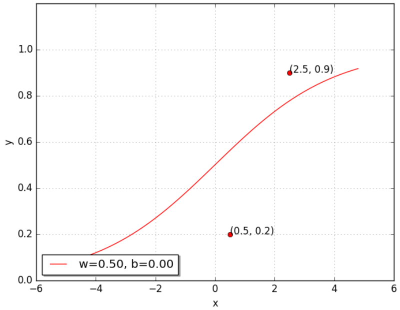
$$\mathscr{L}(w, b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$



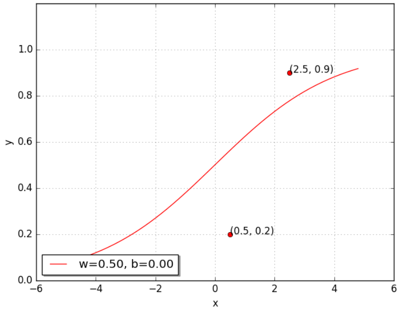
$$\mathscr{L}(w,b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$
$$= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2$$



$$\mathscr{L}(w,b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$
$$= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2$$
$$= \frac{1}{2} * (0.9 - f(2.5))^2 + (0.2 - f(0.5))^2$$



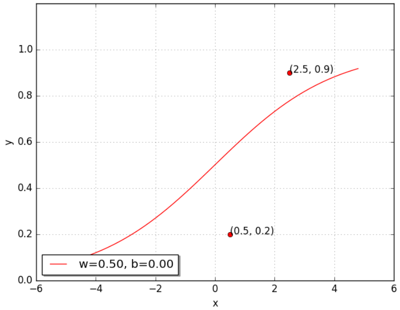
$$
\begin{aligned}
\mathscr{L}(w, b) &= \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2 \\
&= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2 \\
&= \frac{1}{2} * (0.9 - f(2.5))^2 + (0.2 - f(0.5))^2 \\
&= 0.073
\end{aligned}
$$



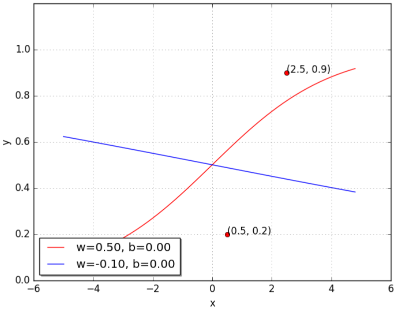
$$\mathscr{L}(w,b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$
$$= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2$$
$$= \frac{1}{2} * (0.9 - f(2.5))^2 + (0.2 - f(0.5))^2$$
$$= 0.073$$

We want $\mathscr{L}(w,b)$ to be as close to 0 as possible

Lets try some other values of w, b



w	b	$\mathscr{L}(w,b)$
0.50	0.00	0.0730

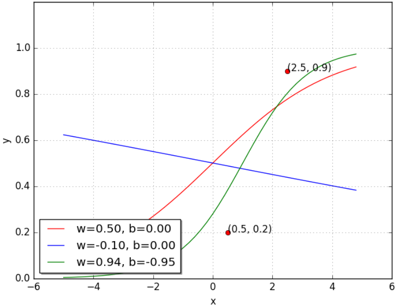


Lets try some other values of w, b

w	b	$\mathscr{L}(w,b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481

Oops!! this made things even worse...

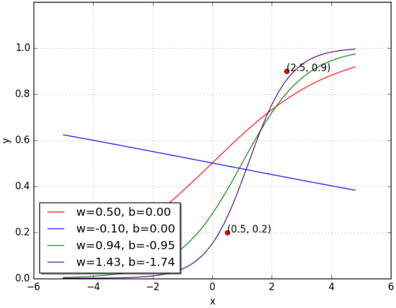
Lets try some other values of w, b



w	b	$\mathscr{L}(w, b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214

Perhaps it would help to push w and b in the other direction...

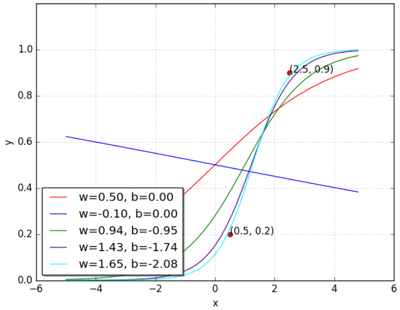
Lets try some other values of w, b



w	b	$\mathscr{L}(w,b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028

Lets keep going in this direction, *i.e.*, increase w and decrease b

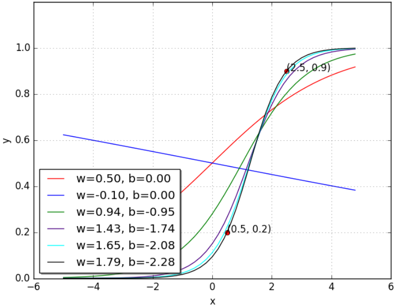
Lets try some other values of w, b



w	b	$\mathscr{L}(w,b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003

Lets keep going in this direction, *i.e.*, increase w and decrease b

Lets try some other values of w, b

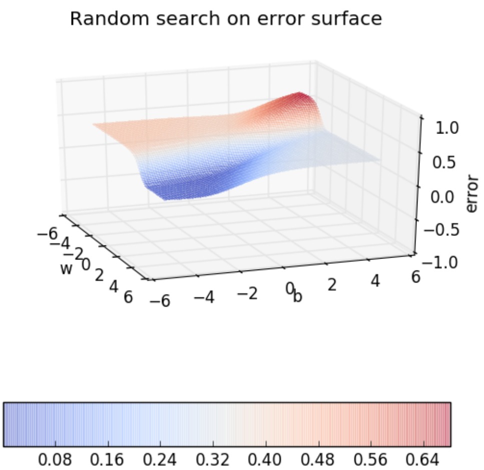


w	b	$\mathscr{L}(w,b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003
1.78	-2.27	0.0000

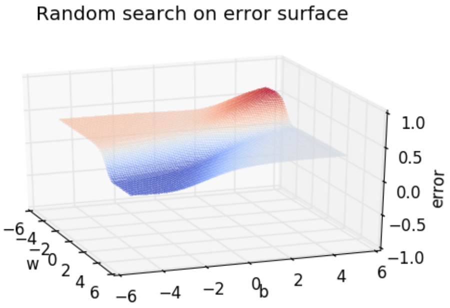
With some guess work and intuition we were able to find the right values for w and b

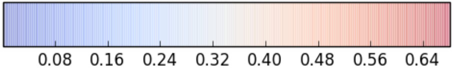
Lets look at something better than our "guess work" algorithm....

- Since we have only 2 points and 2 parameters (w, b) we can easily plot $\mathscr{L}(w, b)$ for different values of (w, b) and pick the one where $\mathscr{L}(w, b)$ is minimum

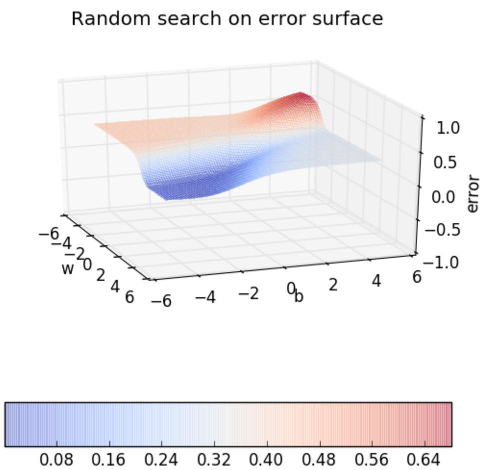


- Since we have only 2 points and 2 parameters (w, b) we can easily plot $\mathscr{L}(w, b)$ for different values of (w, b) and pick the one where $\mathscr{L}(w, b)$ is minimum



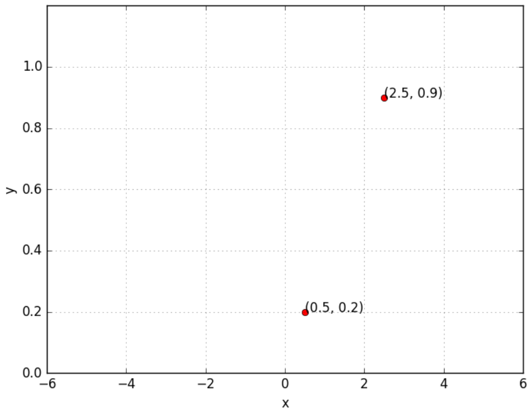


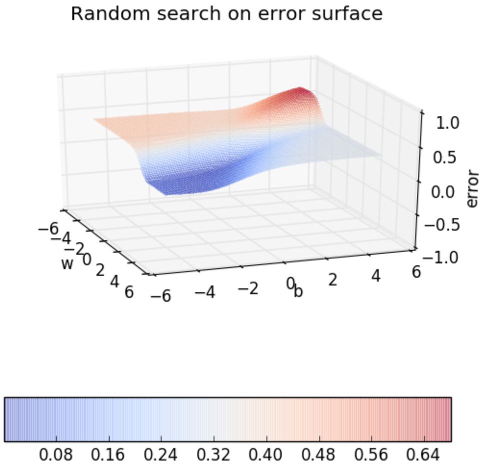
- Since we have only 2 points and 2 parameters (w, b) we can easily plot $\mathscr{L}(w, b)$ for different values of (w, b) and pick the one where $\mathscr{L}(w, b)$ is minimum
- But of course this becomes intractable once you have many more data points and many more parameters !!

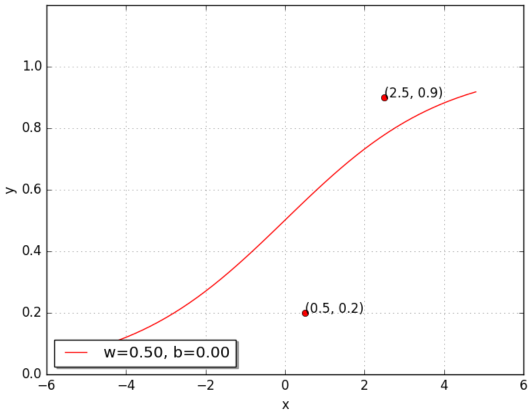


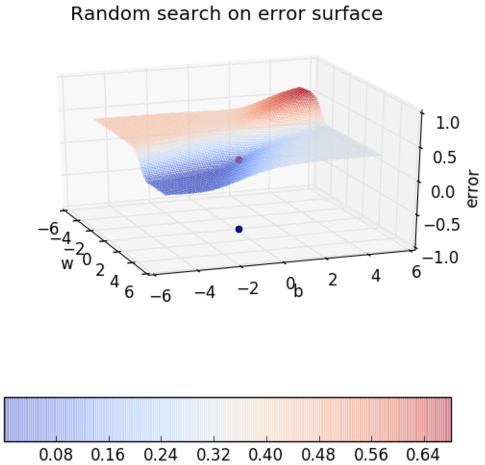
- Since we have only 2 points and 2 parameters $(w,\ b)$ we can easily plot $\mathscr{L}(w, b)$ for different values of $(w,\ b)$ and pick the one where $\mathscr{L}(w, b)$ is minimum
- But of course this becomes intractable once you have many more data points and many more parameters !!
- Further, even here we have plotted the error surface only for a small range of $(w,\ b)$ [from $(-6, 6)$ and not from $(-\inf, \inf)$]

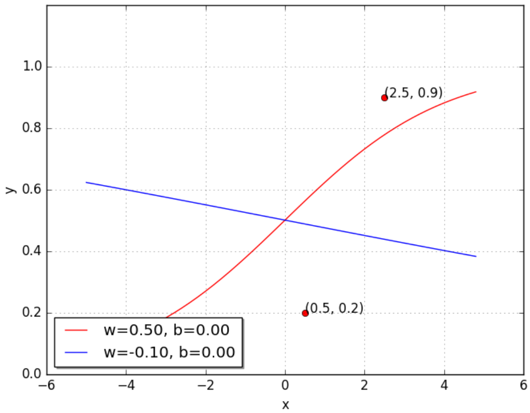
Lets look at the geometric interpretation of our "guess work" algorithm in terms of this error surface

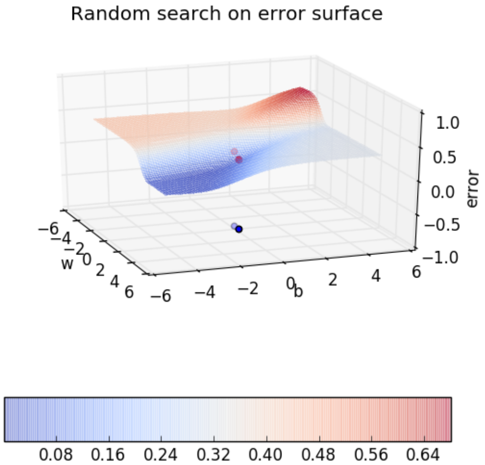

(2.5, 0.9)
(0.5, 0.2)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6

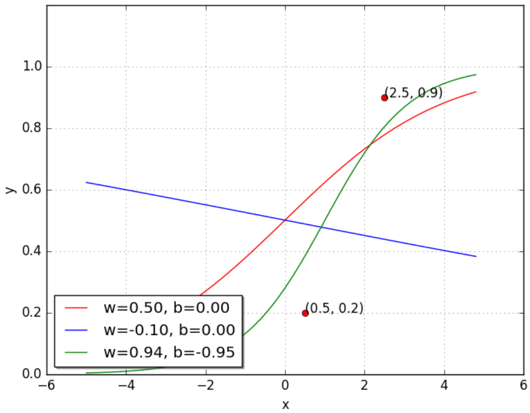

Random search on error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
y
x

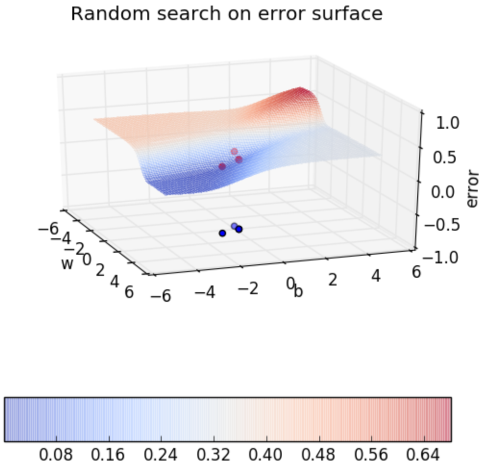

Random search on error surface
w
b
error


(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
x
y

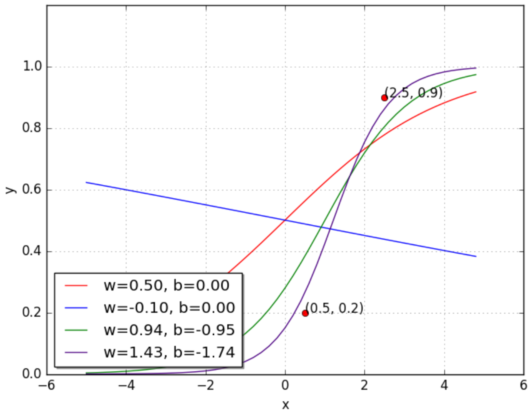

Random search on error surface
w
b
error
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



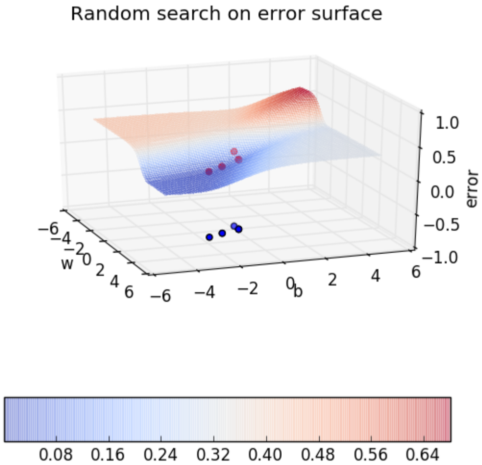
(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
y
x



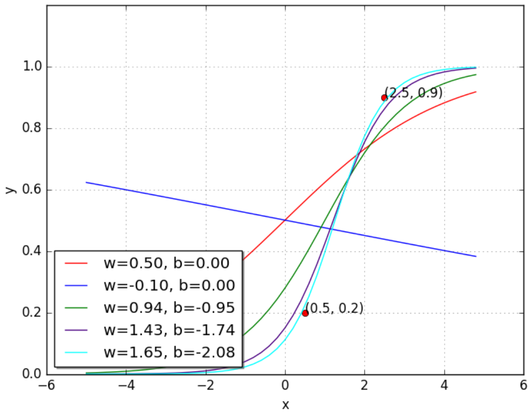
Random search on error surface
w
b
error

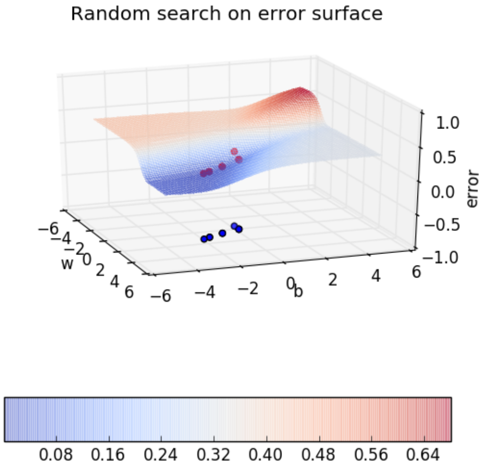


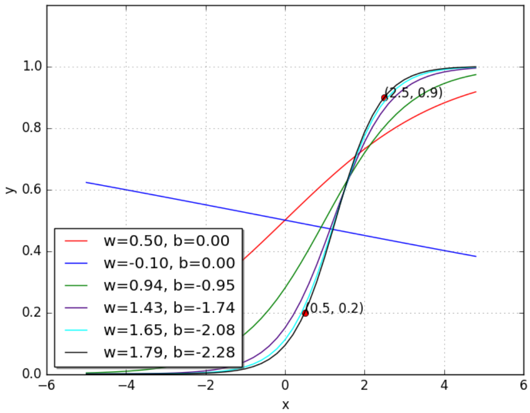
(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
w=1.43, b=-1.74
y
x



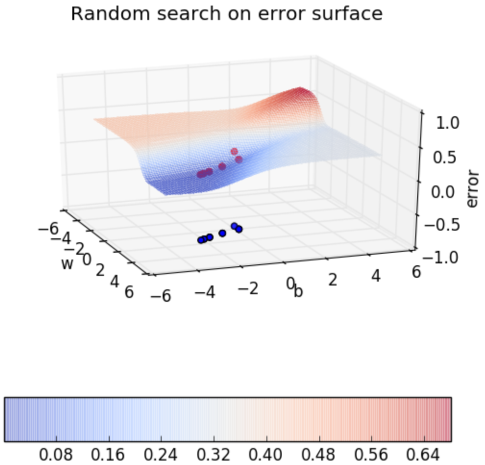
Random search on error surface
w
b
error
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
w=1.43, b=-1.74
w=1.65, b=-2.08
(2.5, 0.9)
(0.5, 0.2)
x
y


Random search on error surface
w
b
error
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
w=1.43, b=-1.74
w=1.65, b=-2.08
w=1.79, b=-2.28
(2.5, 0.9)
(0.5, 0.2)
x
y



Random search on error surface
w
b
error
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

Now lets see if there is a more efficient and principled way of doing this

Goal

Find a better way of traversing the error surface so that we can reach the minimum value quickly without resorting to brute force search!

vector of parameters,
say, randomly initialized

$$\theta = [w, b]$$

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

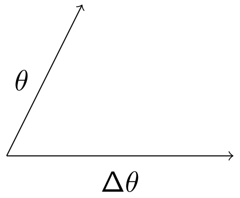
change in the values of w, b

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b

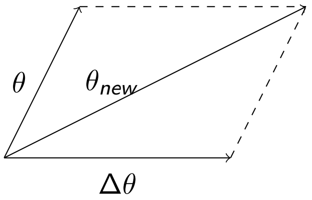


vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b

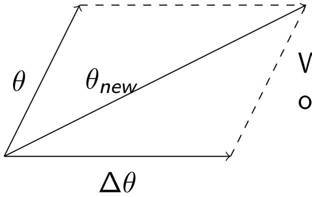


vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



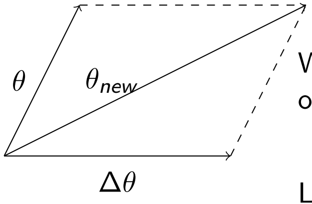
We moved in the direction of $\Delta\theta$

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



We moved in the direction of $\Delta\theta$

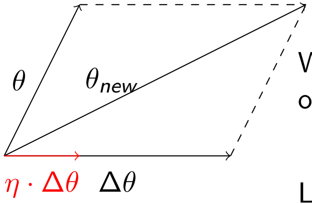
Lets be a bit conservative: move only by a small amount η

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



We moved in the direction of $\Delta\theta$

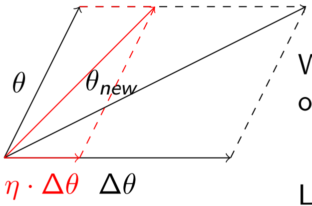
Lets be a bit conservative: move only by a small amount η

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



We moved in the direction of $\Delta\theta$

Lets be a bit conservative: move only by a small amount η

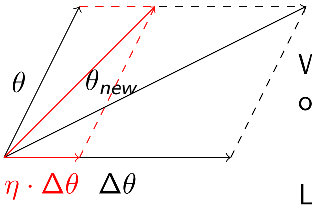
vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b

$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$



We moved in the direction of $\Delta\theta$

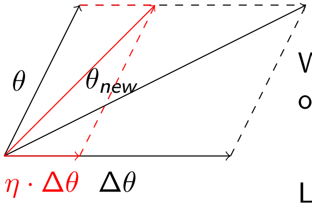
Lets be a bit conservative: move only by a small amount η

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



We moved in the direction of $\Delta\theta$

Lets be a bit conservative: move only by a small amount η

$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$

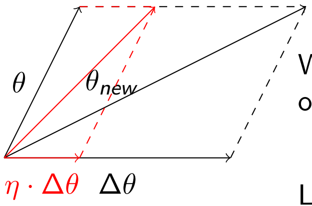
Question: What is the right $\Delta\theta$ to use ?

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



We moved in the direction of $\Delta\theta$

Lets be a bit conservative: move only by a small amount η

$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$

Question: What is the right $\Delta\theta$ to use ?

The answer comes from Taylor series

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

$$\mathscr{L}(\theta + \eta u) = \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathscr{L}(\theta) u + \frac{\eta^3}{3!} * \ldots + \frac{\eta^4}{4!} * \ldots$$

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

$$
\begin{aligned}
\mathscr{L}(\theta + \eta u) &= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathscr{L}(\theta) u + \frac{\eta^3}{3!} * \ldots + \frac{\eta^4}{4!} * \ldots \\
&= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) \ [\eta \textit{ is typically small, so } \eta^2, \eta^3, .. \rightarrow 0]
\end{aligned}
$$

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

$$\begin{aligned}
\mathscr{L}(\theta + \eta u) &= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathscr{L}(\theta) u + \frac{\eta^3}{3!} * \ldots + \frac{\eta^4}{4!} * \ldots \\
&= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) \ [\eta \textit{ is typically small, so } \eta^2, \eta^3, .. \rightarrow 0]
\end{aligned}$$

Note that the move (ηu) would be favorable only if,

$$\mathscr{L}(\theta + \eta u) - \mathscr{L}(\theta) < 0[\textit{i.e., if the new loss is less than the previous loss}]$$

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

$$\begin{aligned}\mathscr{L}(\theta + \eta u) &= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathscr{L}(\theta) u + \frac{\eta^3}{3!} * \ldots + \frac{\eta^4}{4!} * \ldots \\ &= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) \; [\eta \textit{ is typically small, so } \eta^2, \eta^3, .. \rightarrow 0]\end{aligned}$$

Note that the move (ηu) would be favorable only if,

$$\mathscr{L}(\theta + \eta u) - \mathscr{L}(\theta) < 0 [\textit{i.e., if the new loss is less than the previous loss}]$$

This implies,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$?

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Lets see....

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Lets see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Lets see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

$$-1 \leq cos(\beta) = \frac{u^T \nabla \mathscr{L}(\theta)}{||u|| * ||\nabla \mathscr{L}(\theta)||} \leq 1$$

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Lets see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

$$-1 \leq cos(\beta) = \frac{u^T \nabla \mathscr{L}(\theta)}{||u|| * ||\nabla \mathscr{L}(\theta)||} \leq 1$$

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Lets see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

$$-1 \leq cos(\beta) = \frac{u^T \nabla \mathscr{L}(\theta)}{||u|| * ||\nabla \mathscr{L}(\theta)||} \leq 1$$

multiply throughout by $k = ||u|| * ||\nabla \mathscr{L}(\theta)||$

$$-k \leq k * cos(\beta) = u^T \nabla \mathscr{L}(\theta) \leq k$$

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Lets see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

$$-1 \leq cos(\beta) = \frac{u^T \nabla \mathscr{L}(\theta)}{||u|| * ||\nabla \mathscr{L}(\theta)||} \leq 1$$

multiply throughout by $k = ||u|| * ||\nabla \mathscr{L}(\theta)||$

$$-k \leq k * cos(\beta) = u^T \nabla \mathscr{L}(\theta) \leq k$$

Thus, $\mathscr{L}(\theta + \eta u) - \mathscr{L}(\theta) = u^T \nabla \mathscr{L}(\theta) = k * cos(\beta)$ will be most negative when $cos(\beta) = -1$ *i.e.*, when β is $180°$

Gradient Descent Rule

- The direction u that we intend to move in should be at 180° w.r.t. the gradient

Gradient Descent Rule

- The direction u that we intend to move in should be at 180° w.r.t. the gradient
- In other words, move in a direction opposite to the gradient

Gradient Descent Rule

- The direction u that we intend to move in should be at 180° w.r.t. the gradient
- In other words, move in a direction opposite to the gradient

Parameter Update Equations

$$
\begin{aligned}
w_{t+1} &= w_t - \eta \nabla w_t \\
b_{t+1} &= b_t - \eta \nabla b_t \\
where, \nabla w_t &= \frac{\partial \mathscr{L}(w, b)}{\partial w}_{at\ w = w_t, b = b_t}, \nabla b = \frac{\partial \mathscr{L}(w, b)}{\partial b}_{at\ w = w_t, b = b_t}
\end{aligned}
$$

Gradient Descent Rule

- The direction u that we intend to move in should be at 180° w.r.t. the gradient
- In other words, move in a direction opposite to the gradient

Parameter Update Equations

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$b_{t+1} = b_t - \eta \nabla b_t$$

$$where, \nabla w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}_{at\ w = w_t, b = b_t}, \nabla b = \frac{\partial \mathscr{L}(w,b)}{\partial b}_{at\ w = w_t, b = b_t}$$

So we now have a more principled way of moving in the w-b plane than our "guess work" algorithm

- Lets create an algorithm from this rule ...

- Lets create an algorithm from this rule ...

Algorithm 1: gradient_descent()

$t \leftarrow 0$;

$max\_iterations \leftarrow 1000$;

while $t < max\_iterations$ **do**

  $w_{t+1} \leftarrow w_t - \eta \nabla w_t$;

  $b_{t+1} \leftarrow b_t - \eta \nabla b_t$;

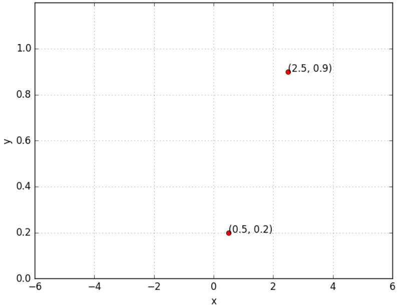
end

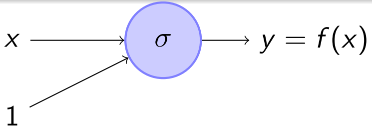
- To see this algorithm in practice lest first derive ∇w and ∇b for our toy neural network

$$x \longrightarrow \sigma \longrightarrow y = f(x)$$

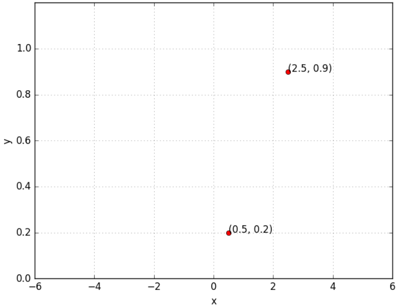
$$1$$

$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

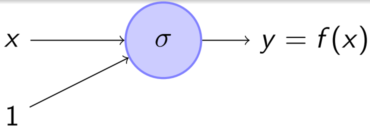




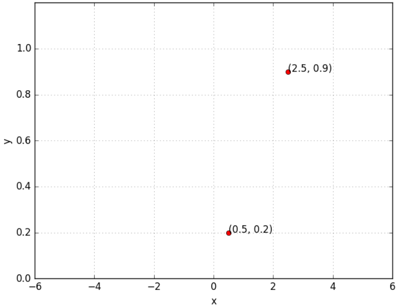
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



Let's assume there is only 1 point to fit (x, y)

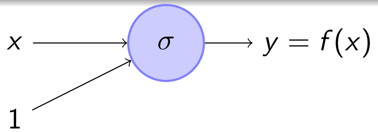


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

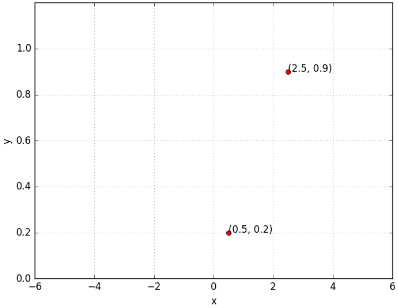


Let's assume there is only 1 point to fit (x, y)

$$\mathscr{L}(w, b) = \frac{1}{2} * (f(x) - y)^2$$



$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



Let's assume there is only 1 point to fit (x, y)

$$\mathscr{L}(w, b) = \frac{1}{2} * (f(x) - y)^2$$

$$\nabla w = \frac{\partial \mathscr{L}(w, b)}{\partial w} = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$

$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$

$$
\begin{aligned}
\nabla w &= \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2] \\
&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)] \\
&= (f(x) - y) * \frac{\partial}{\partial w}(f(x))
\end{aligned}
$$

$$
\begin{aligned}
\nabla w &= \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2] \\
&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)] \\
&= (f(x) - y) * \frac{\partial}{\partial w}(f(x)) \\
&= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)
\end{aligned}
$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2}\frac{\partial}{\partial w}(e^{-(wx+b)}))$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2}\frac{\partial}{\partial w}(e^{-(wx+b)}))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)})\frac{\partial}{\partial w}(-(wx + b)))$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

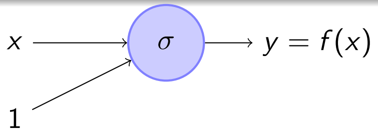
$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w}(e^{-(wx+b)}))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w}(-(wx + b)))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x)$$
$$= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x)$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

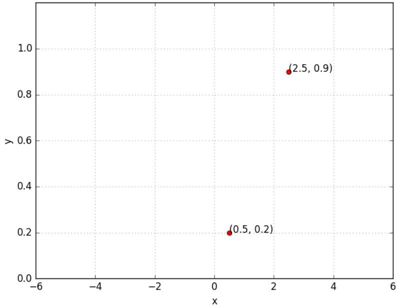
$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w}(e^{-(wx+b)}))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w}(-(wx + b)))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x)$$
$$= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x)$$
$$= f(x) * (1 - f(x)) * x$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$
$$= (f(x) - y) * f(x) * (1 - f(x)) * x$$

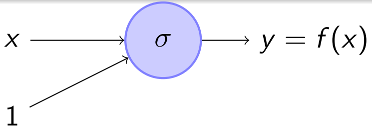
$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w}(e^{-(wx+b)}))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w}(-(wx + b)))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x)$$
$$= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x)$$
$$= f(x) * (1 - f(x)) * x$$



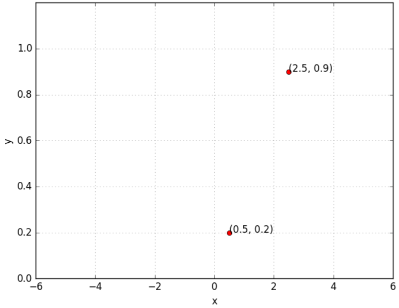
$$f(x) = \frac{1}{1+e^{-(w \cdot x+b)}}$$



So if there is only 1 point (x, y), we have,

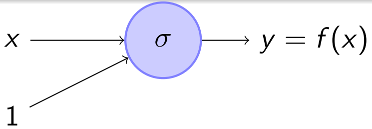


$$f(x) = \frac{1}{1+e^{-(w \cdot x+b)}}$$

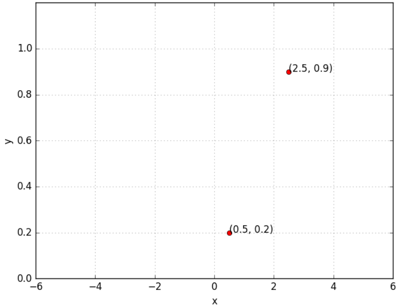


So if there is only 1 point (x, y), we have,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$



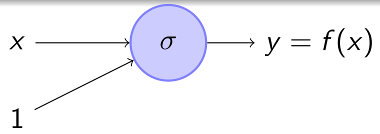
$$f(x) = \frac{1}{1+e^{-(w \cdot x+b)}}$$



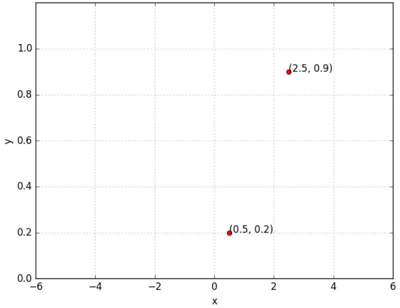
So if there is only 1 point (x, y), we have,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

For two points,



$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

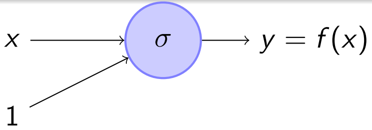


So if there is only 1 point (x, y), we have,

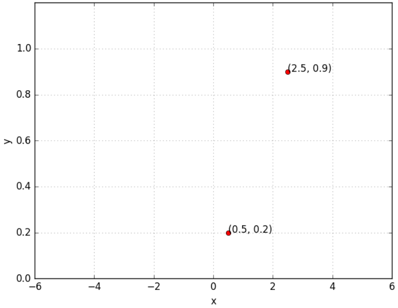
$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

For two points,

$$\nabla w = \sum_{i=1}^{2} (f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) * x_i$$



$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



So if there is only 1 point (x, y), we have,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

For two points,

$$\nabla w = \sum_{i=1}^{2} (f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) * x_i$$

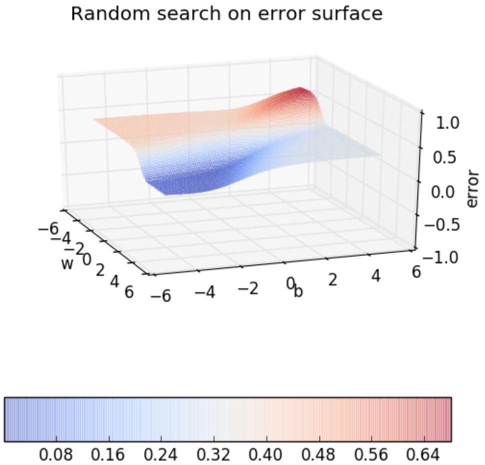
$$\nabla b = \sum_{i=1}^{2} (f(x_i) - y_i) * f(x_i) * (1 - f(x_i))$$

```
X = [0.5, 2.5]
Y = [0.2, 0.9]
```

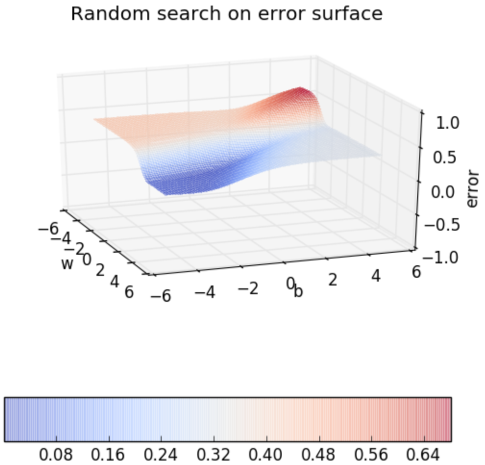
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))
```

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err
```

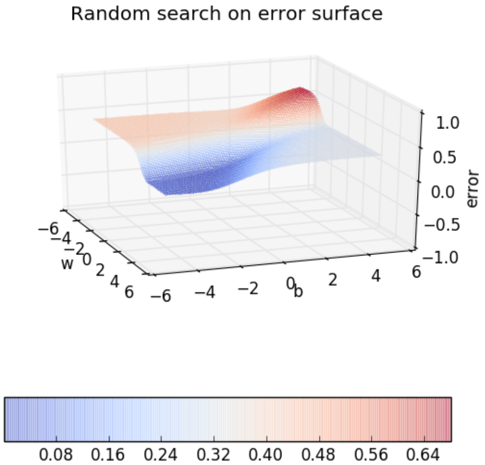
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)
```

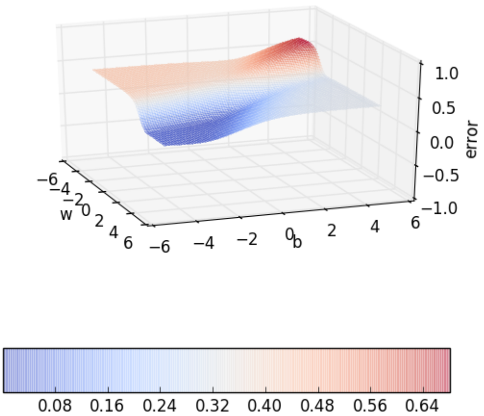


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x
```

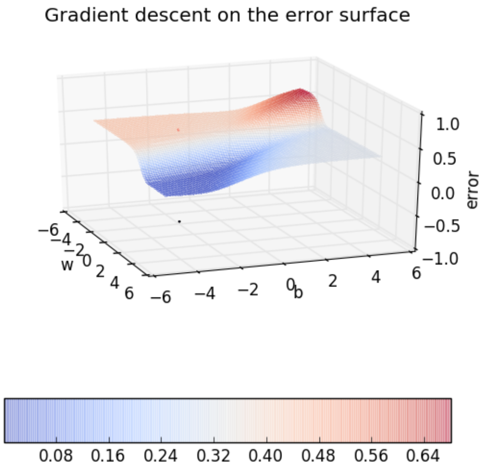


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

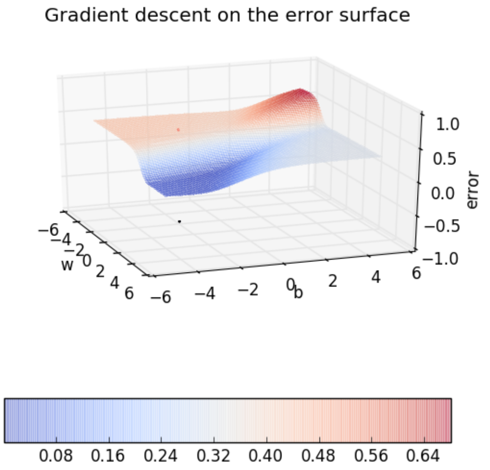
Random search on error surface



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

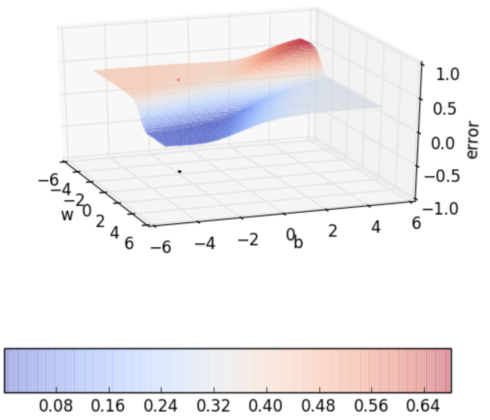


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



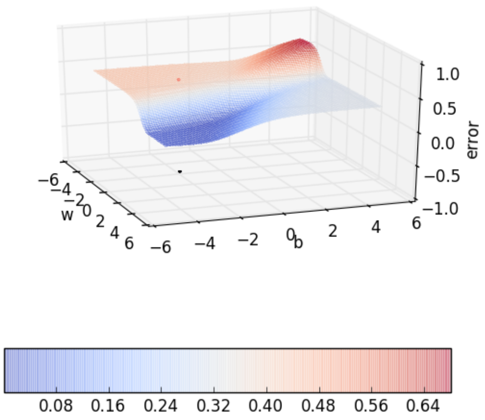
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



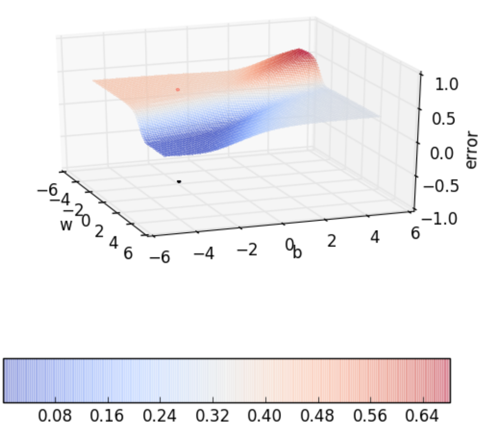
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

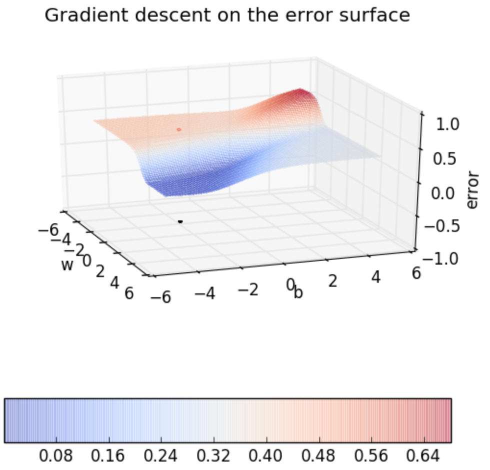


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

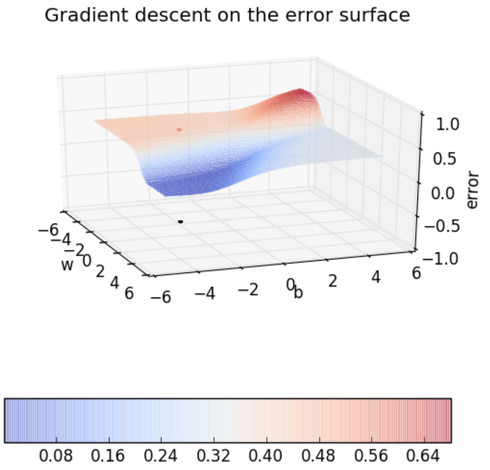
Gradient descent on the error surface



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

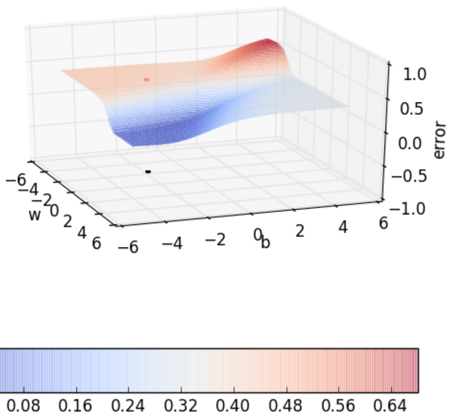


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

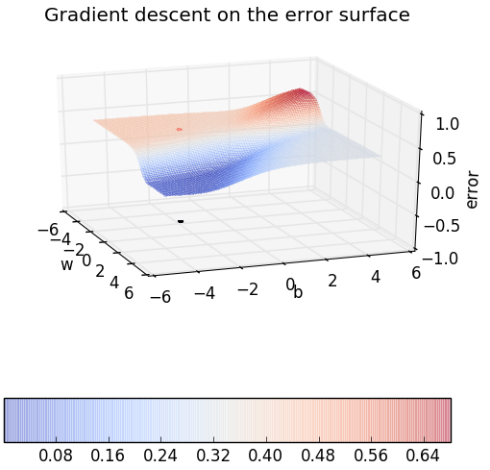


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

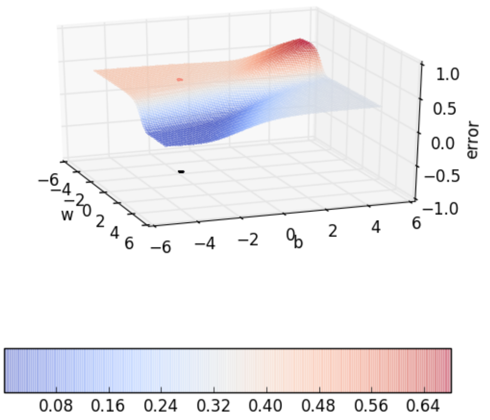


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



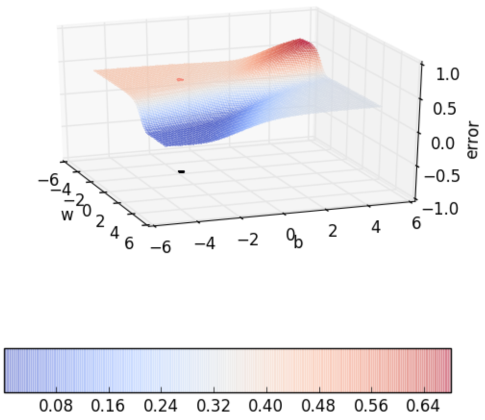
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

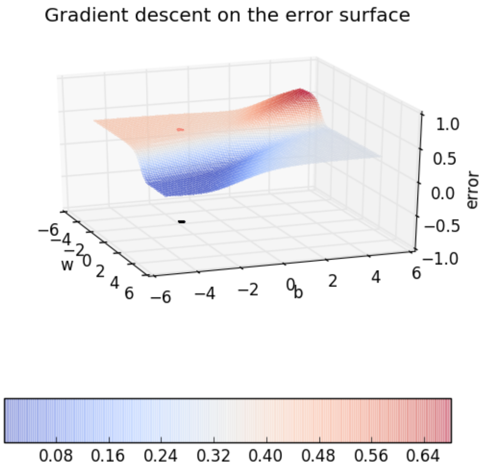


```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

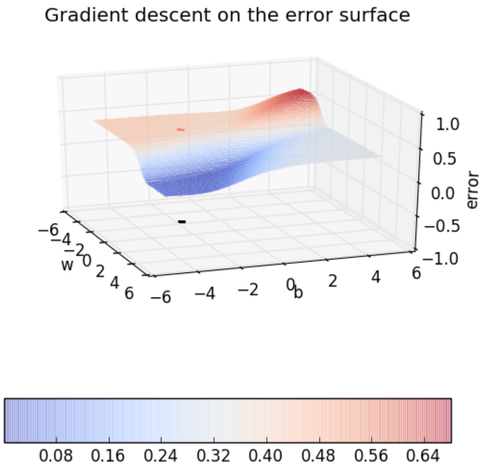
Gradient descent on the error surface



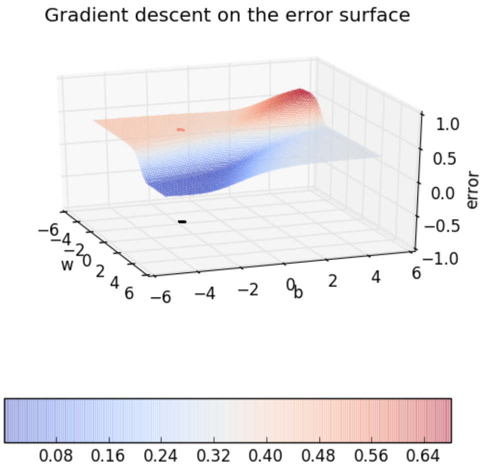
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

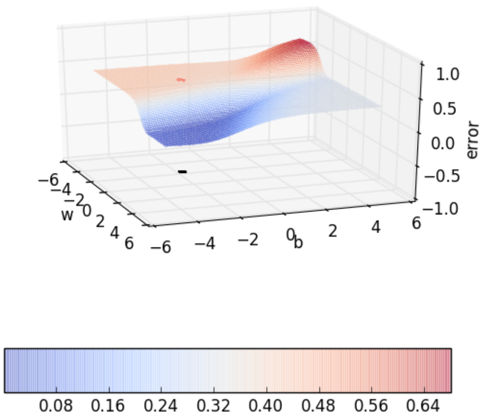


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

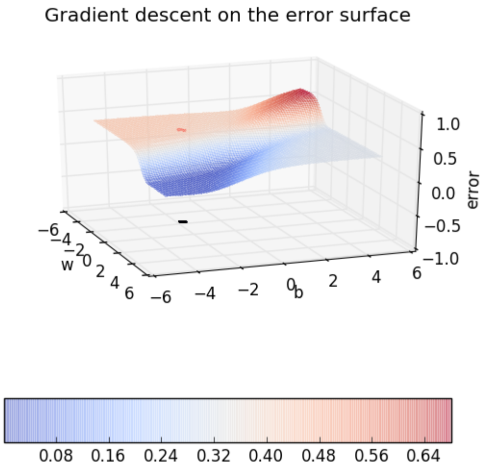


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

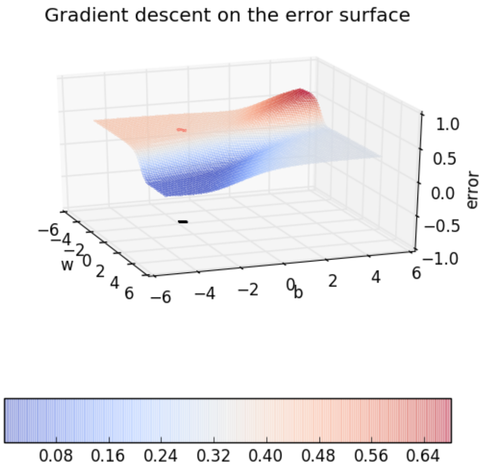
Gradient descent on the error surface



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

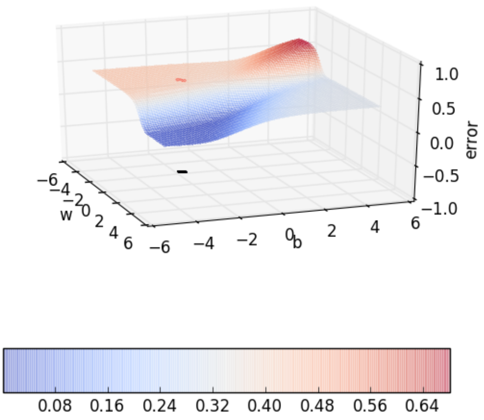


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

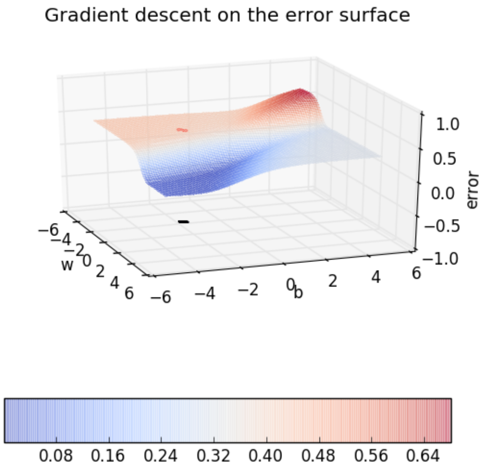


```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

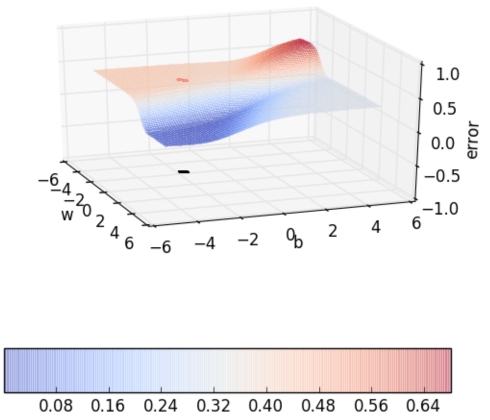


```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

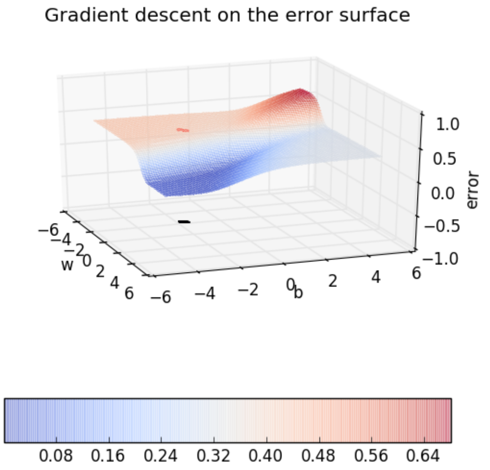


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

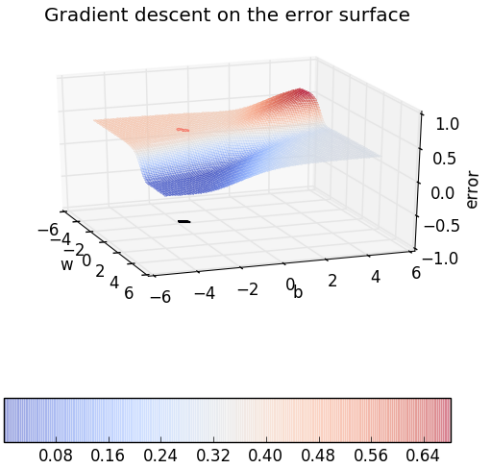
Gradient descent on the error surface



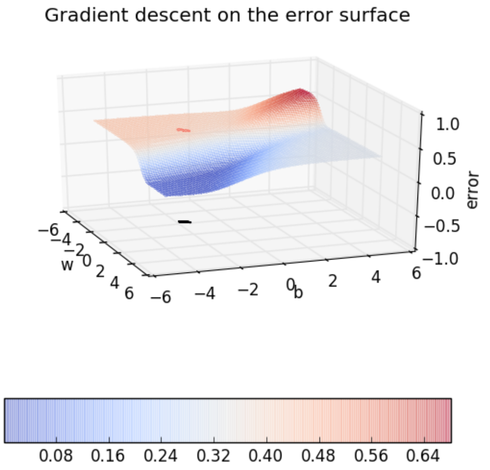
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



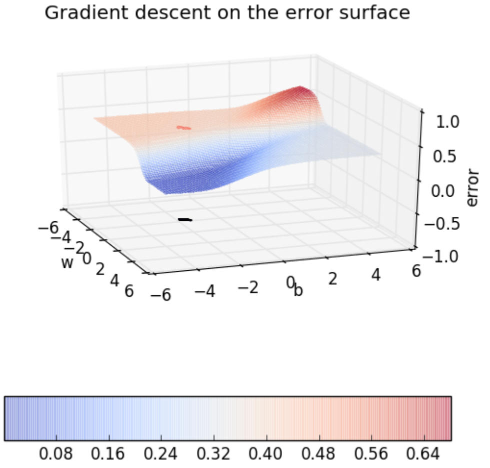
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

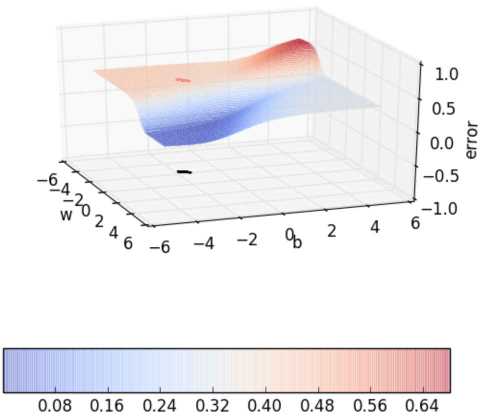


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



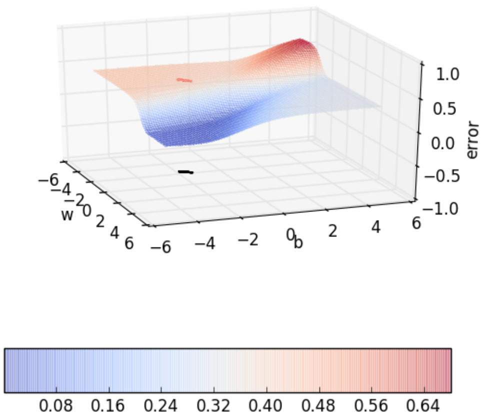
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



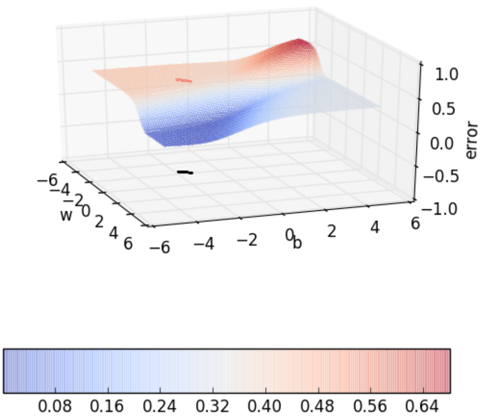
```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



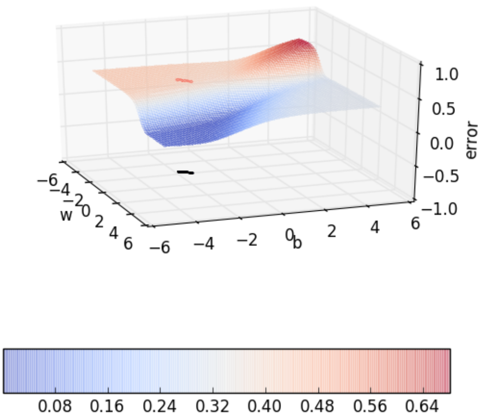
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

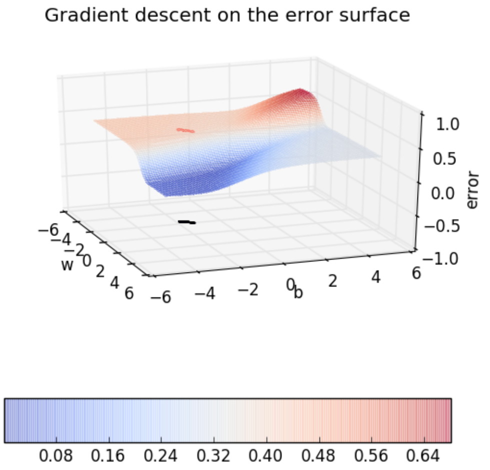


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

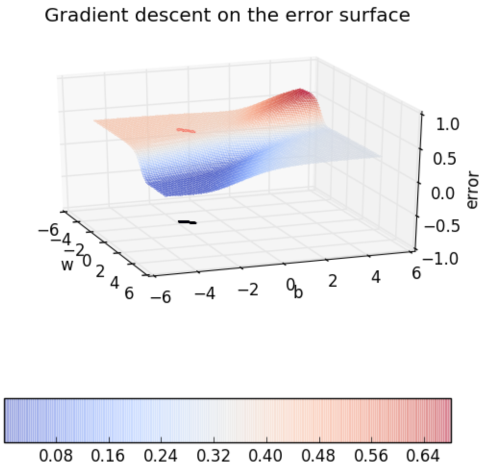
Gradient descent on the error surface



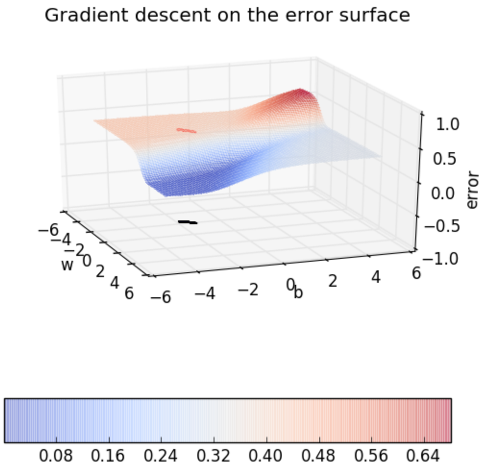
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



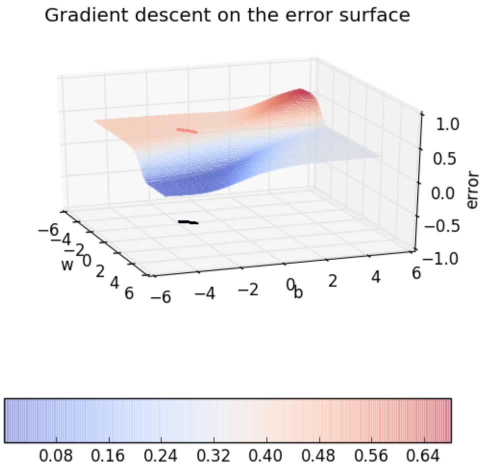
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



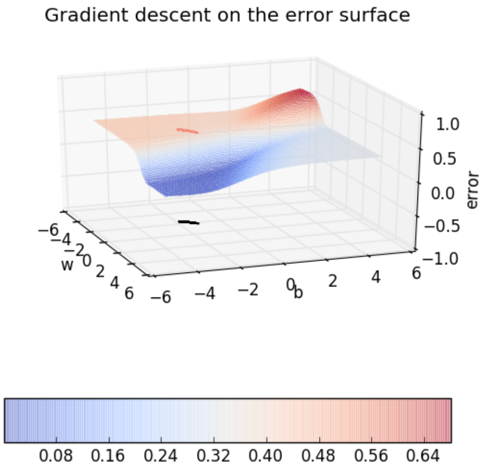
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



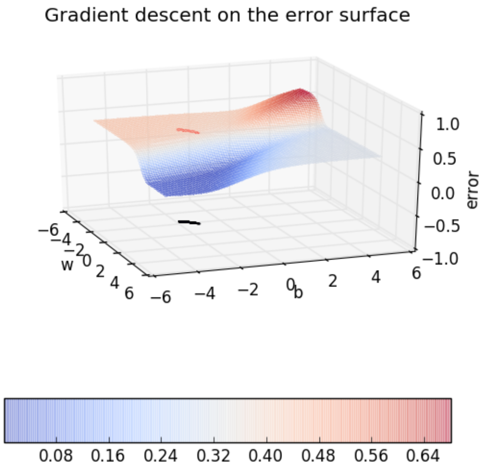
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

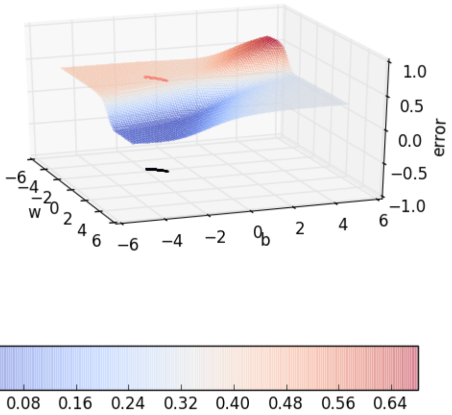


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

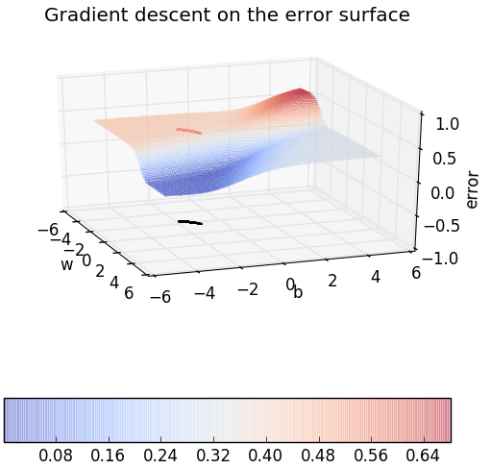


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

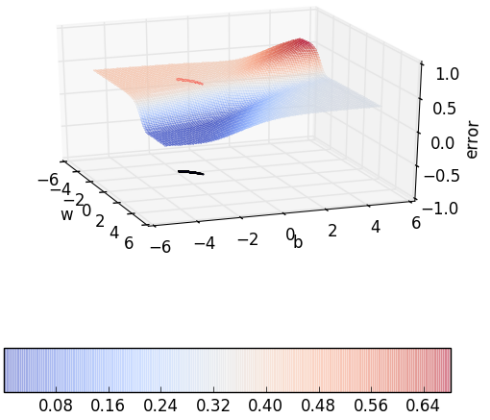


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

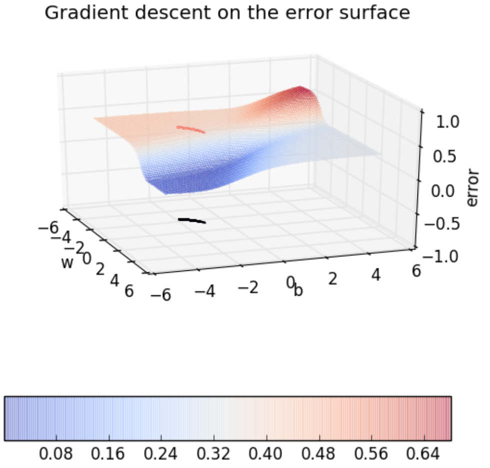


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

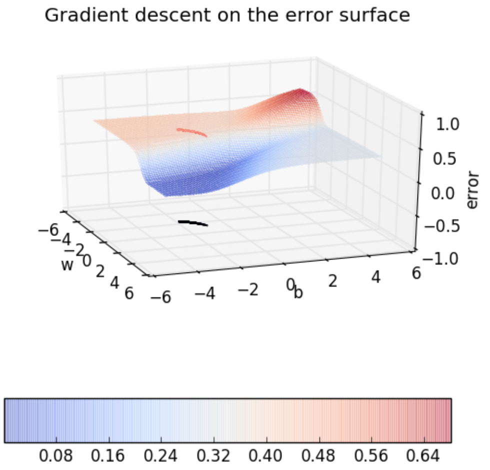
Gradient descent on the error surface



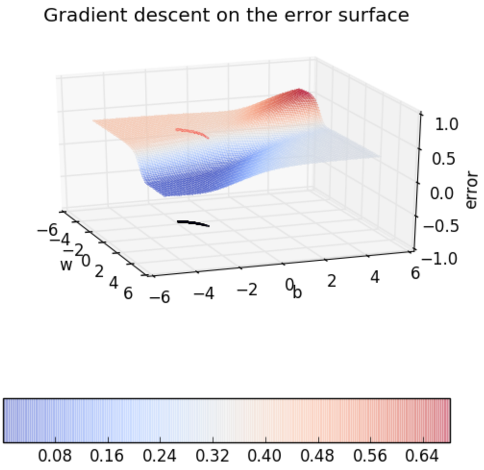
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



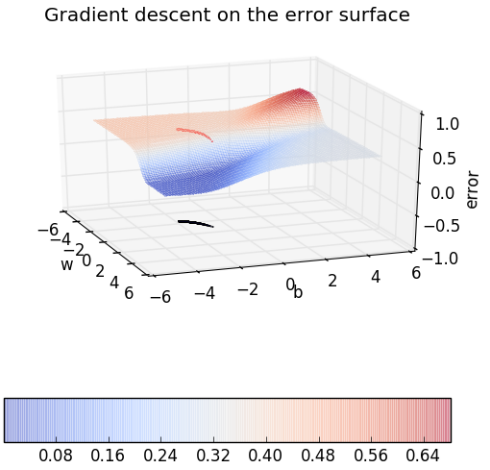
```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



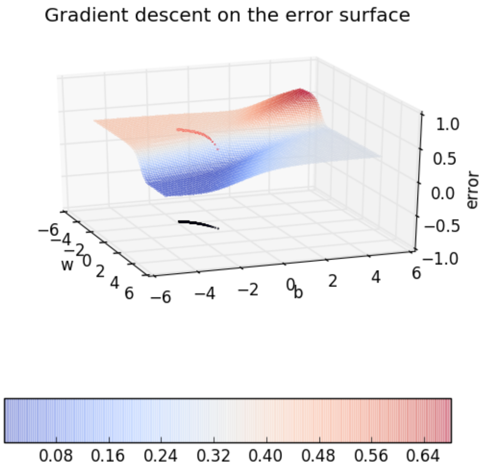
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



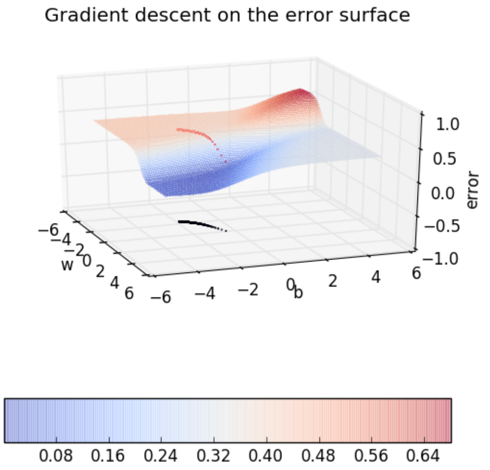
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



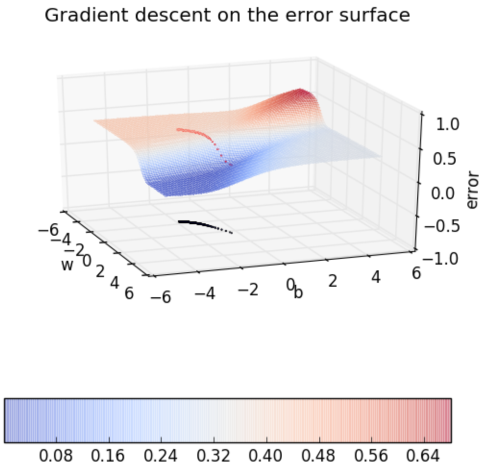
```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

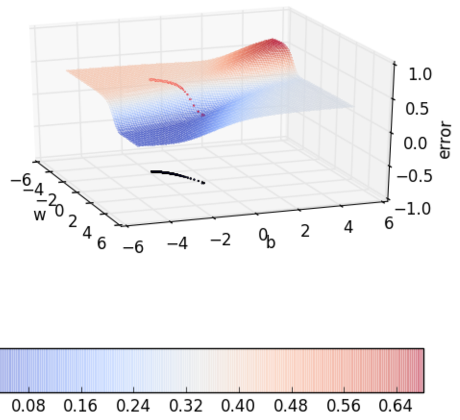


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

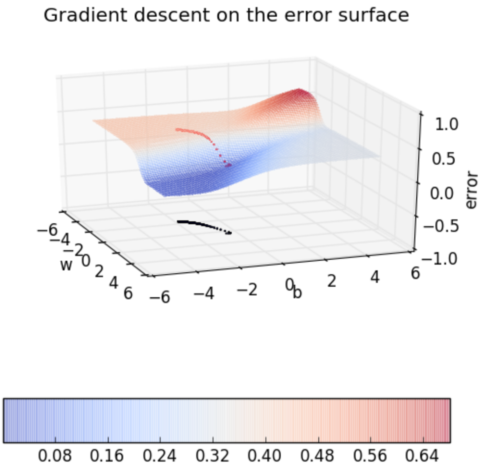


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

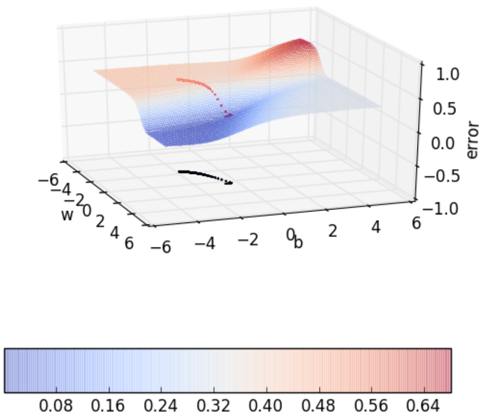


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

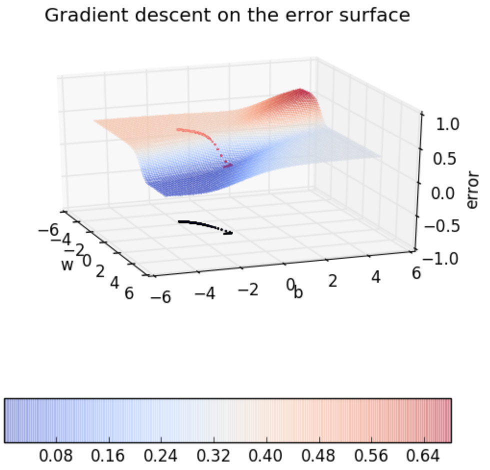


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

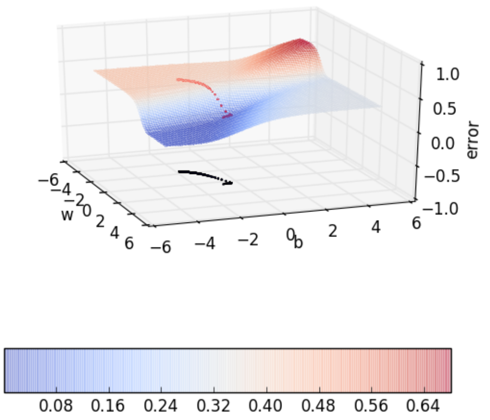


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

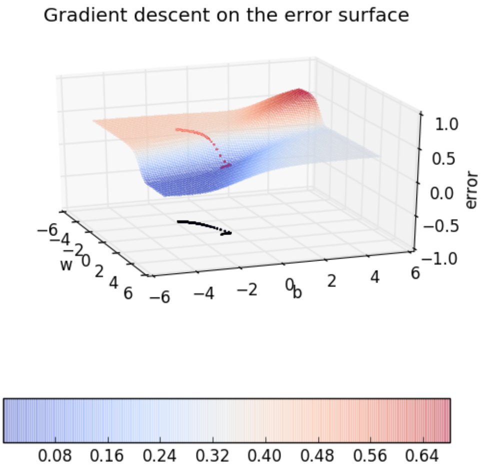


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

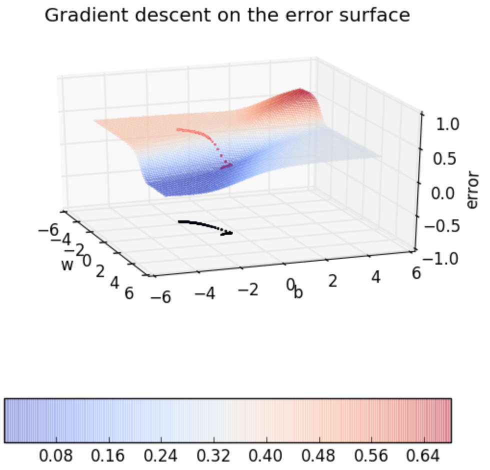
Gradient descent on the error surface



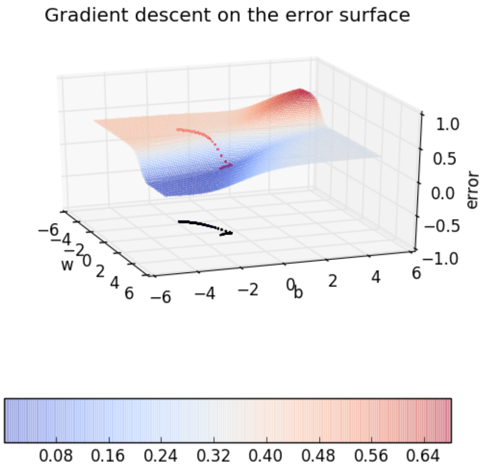
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



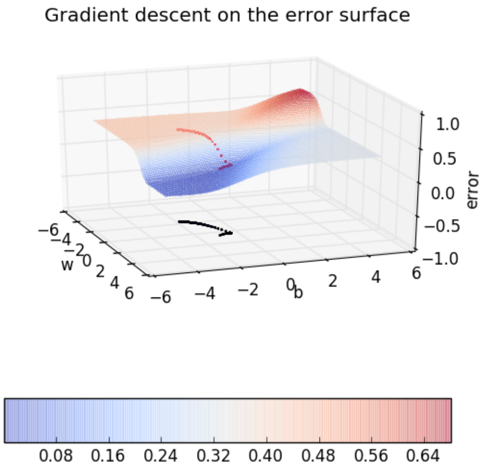
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



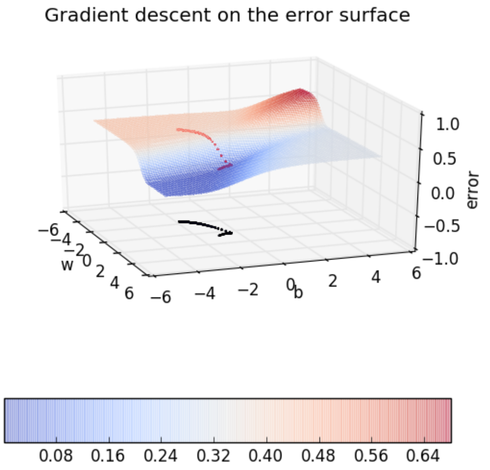
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



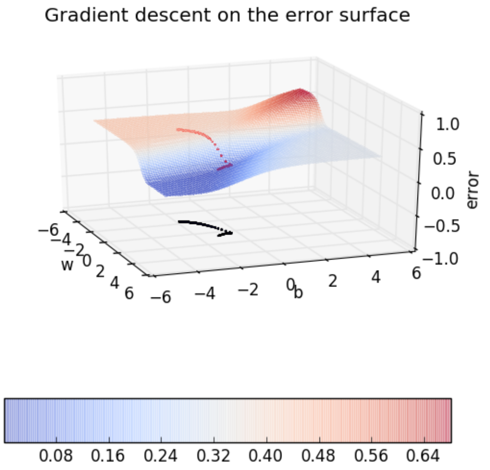
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



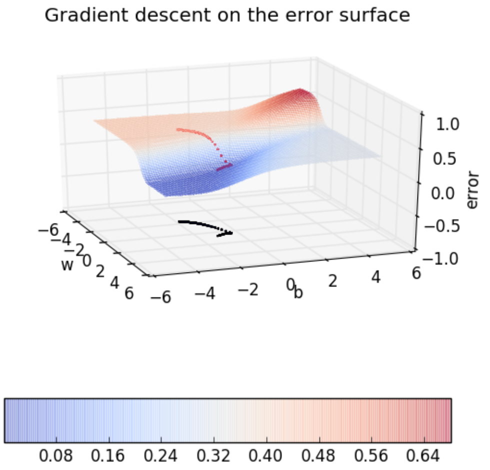
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



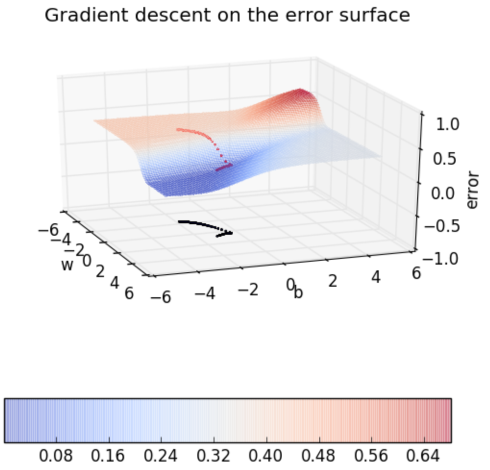
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



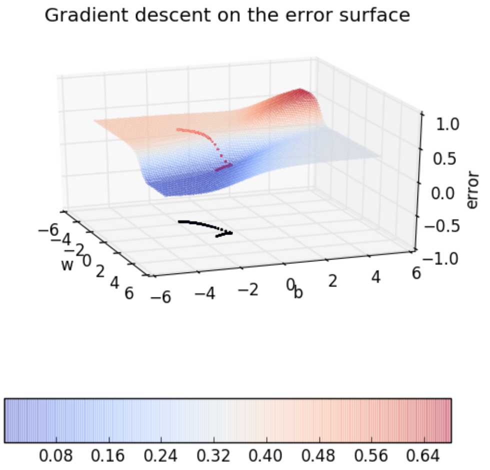
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



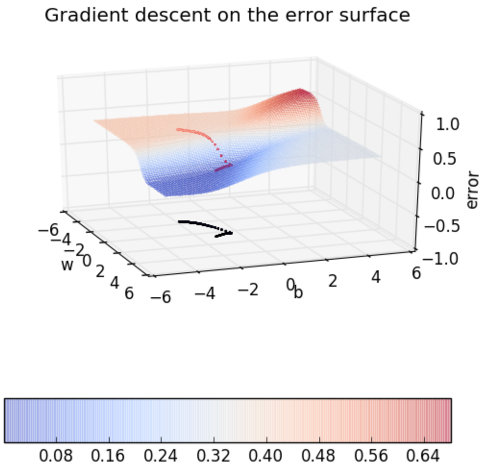
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

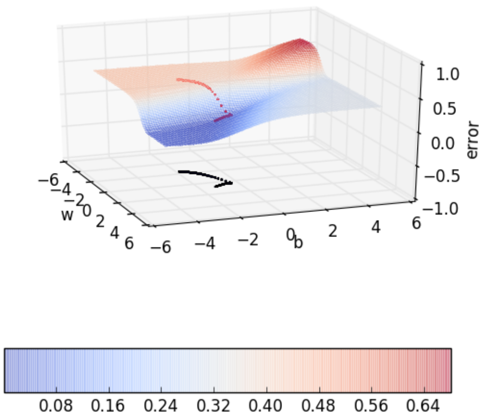


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

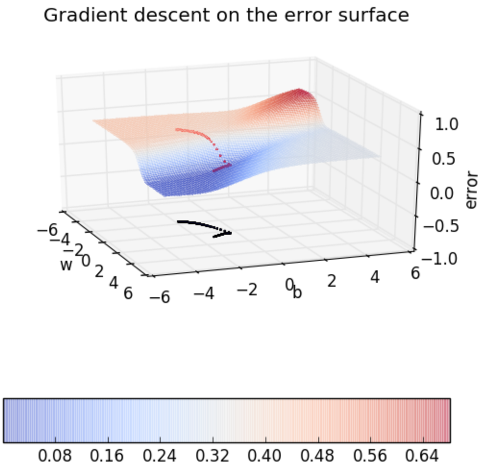


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

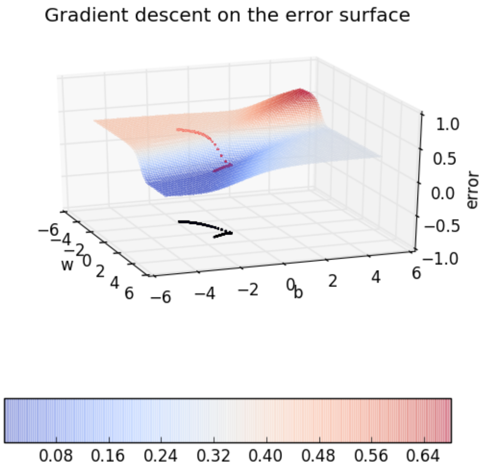
Gradient descent on the error surface



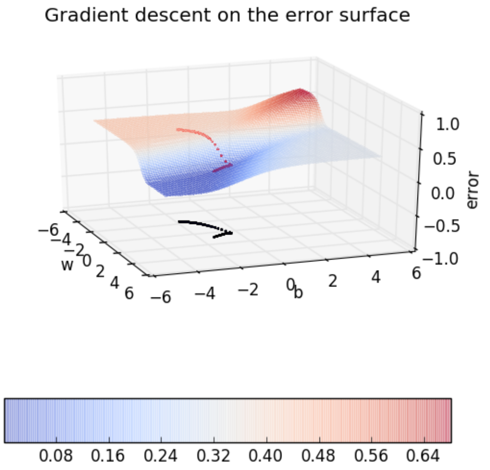
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



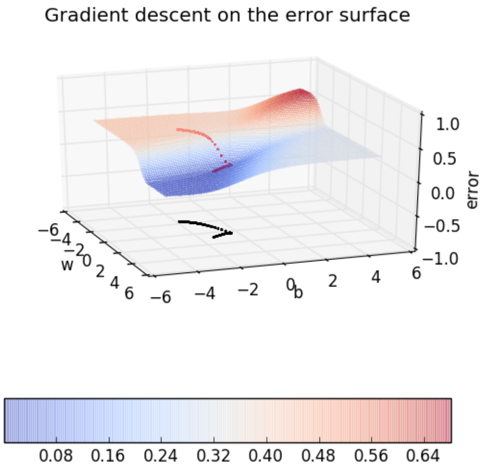
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



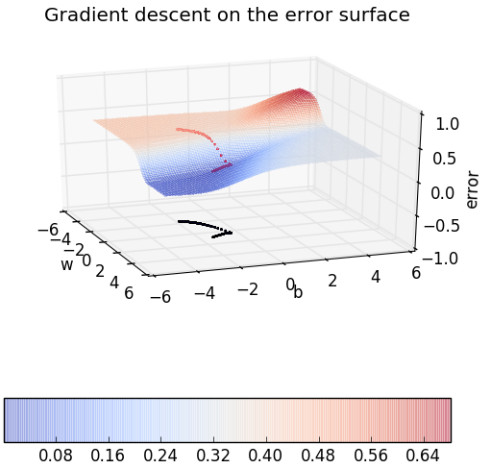
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



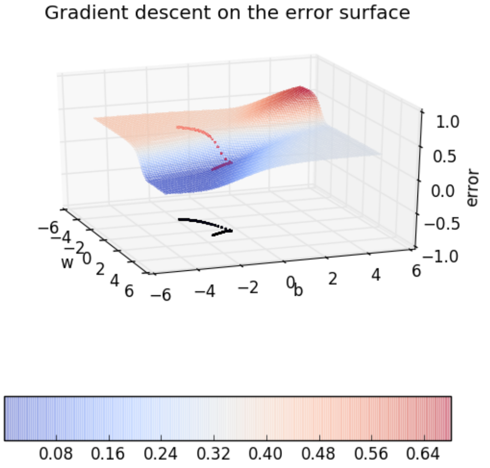
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

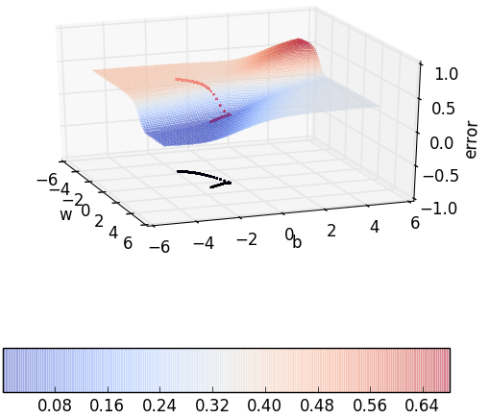


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

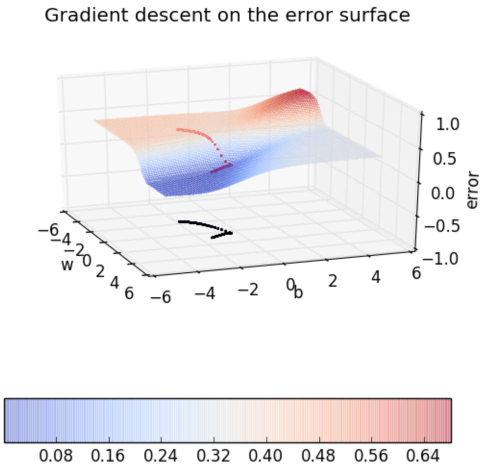


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

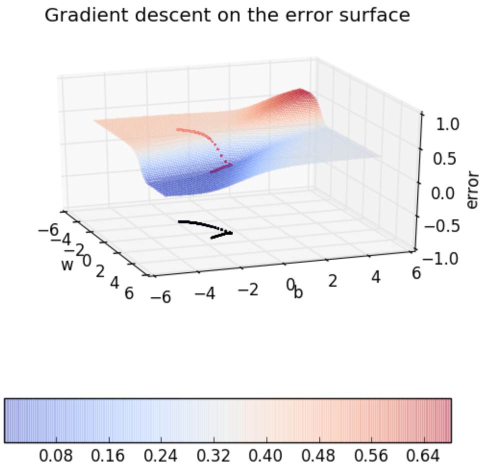
Gradient descent on the error surface



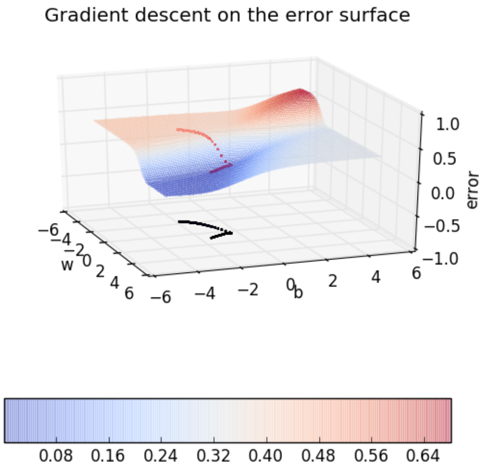
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



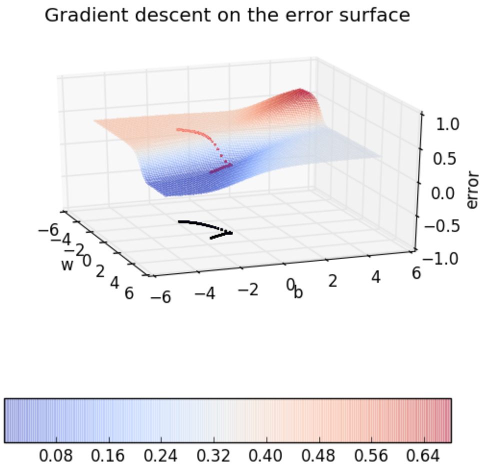
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



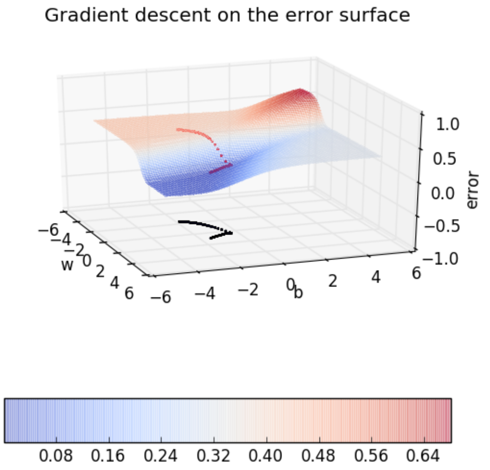
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



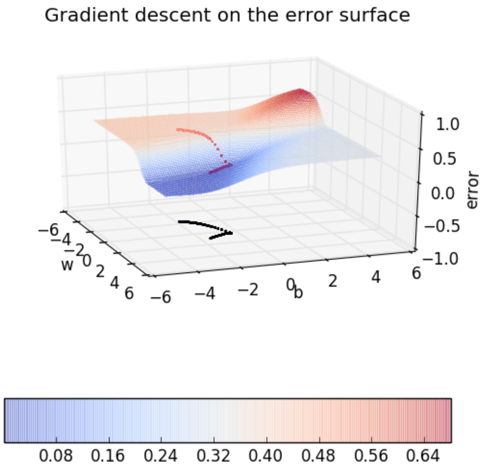
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



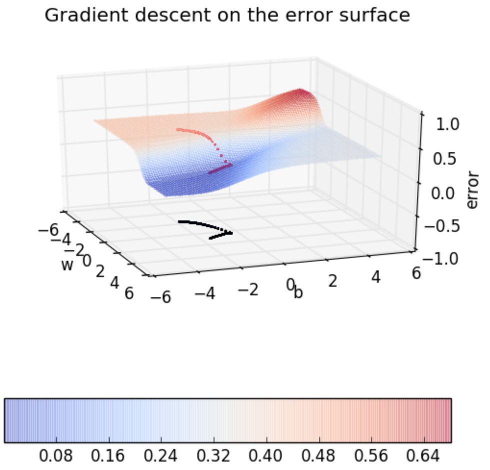
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



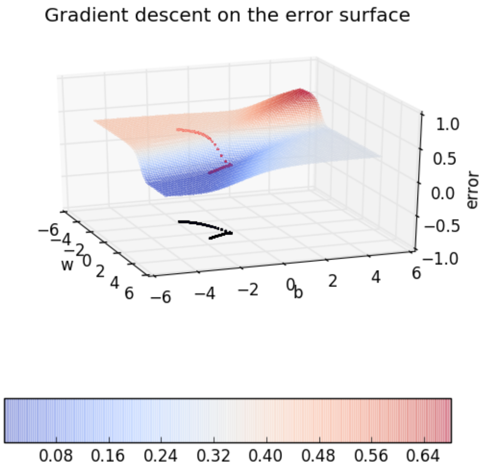
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



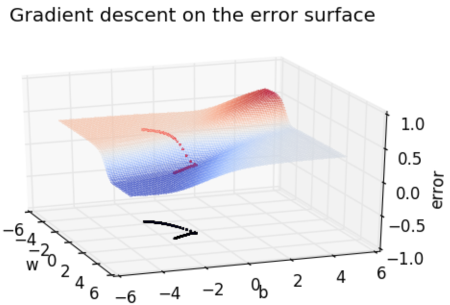
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

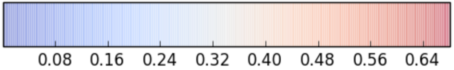


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



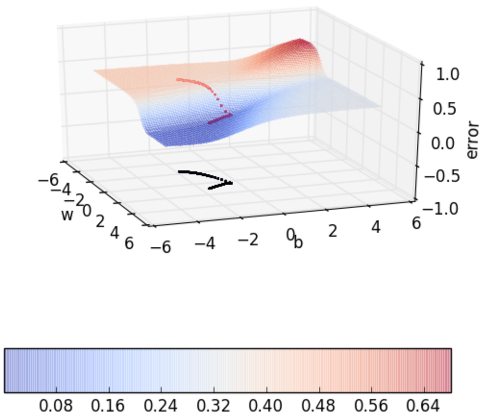
```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```





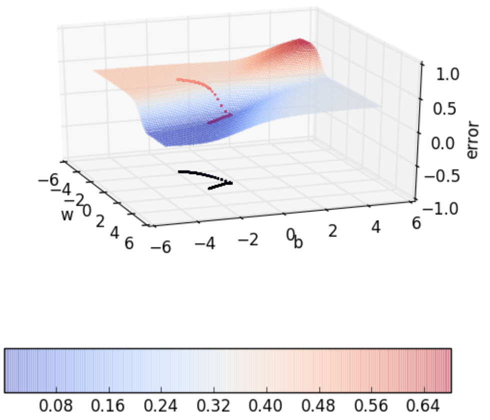
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

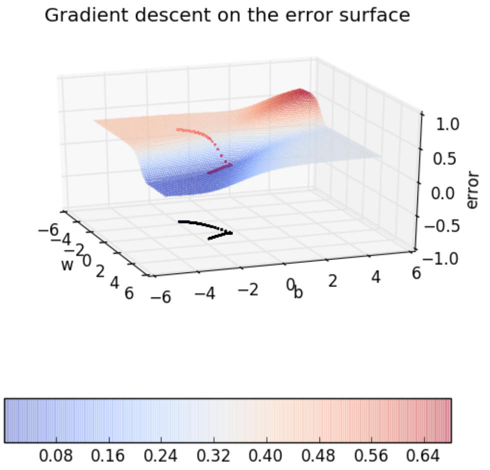


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

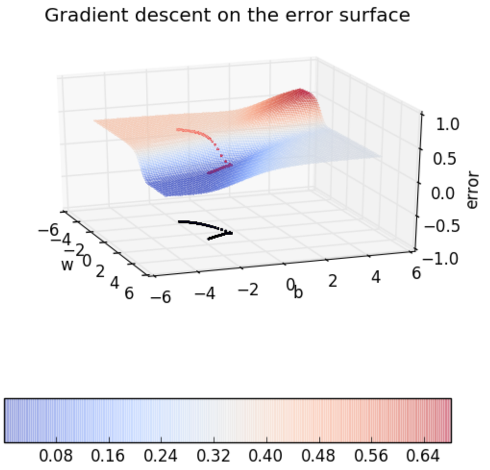
Gradient descent on the error surface



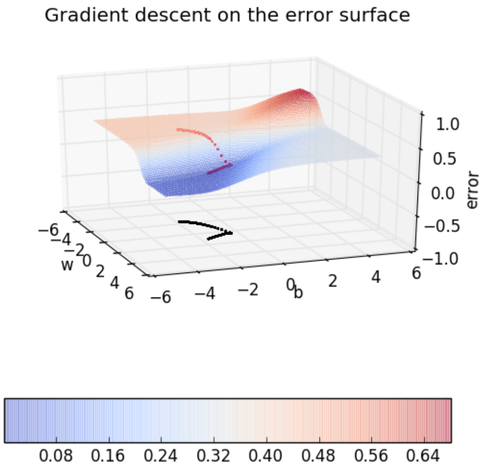
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



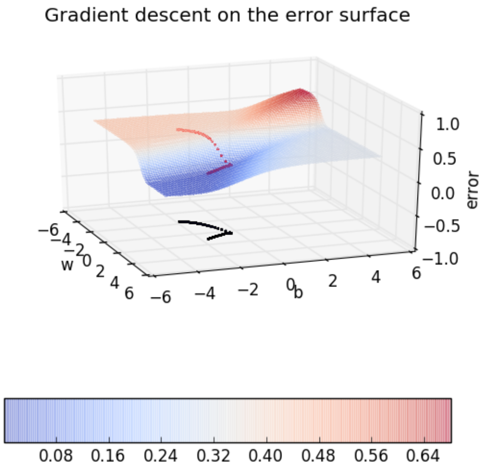
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



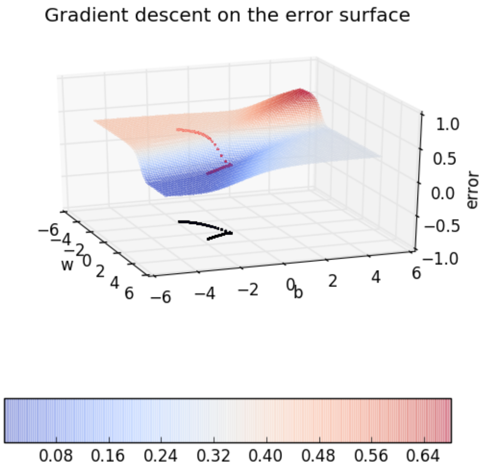
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

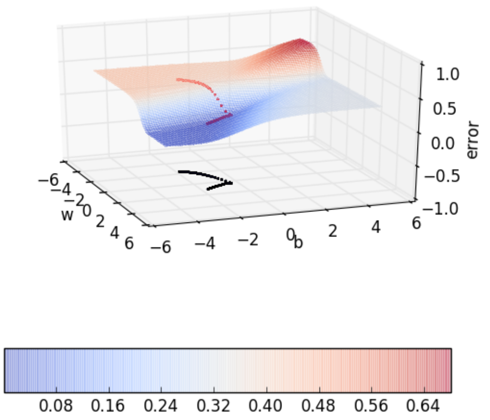


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

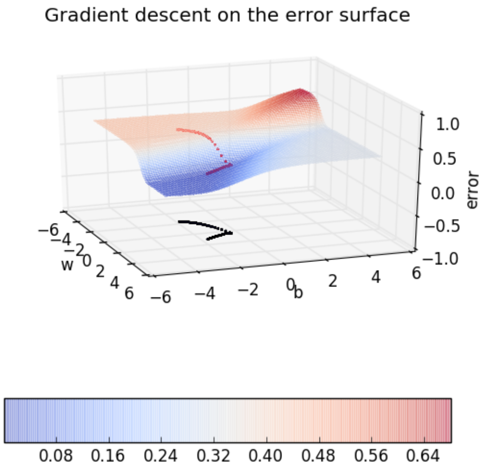


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

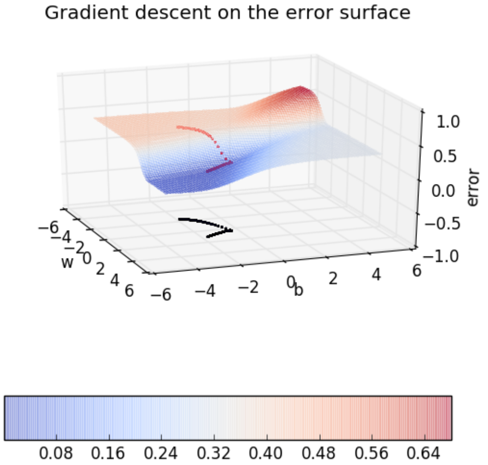
Gradient descent on the error surface



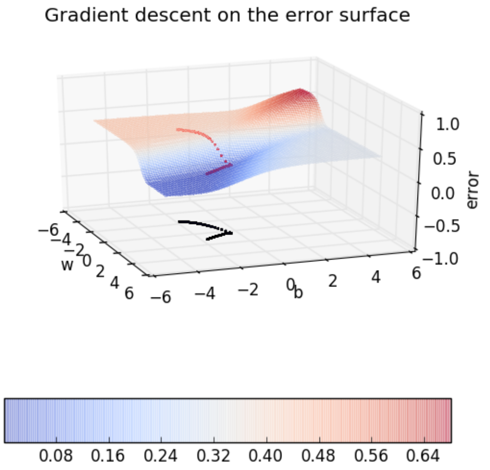
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

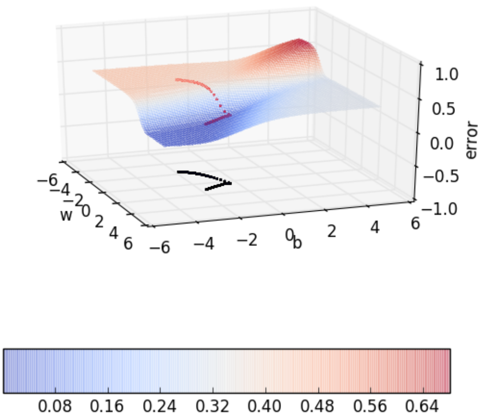


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

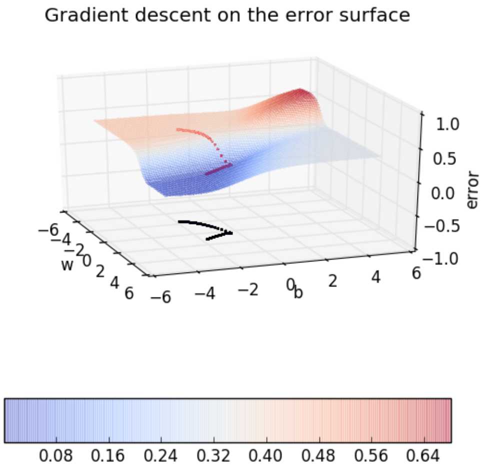


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

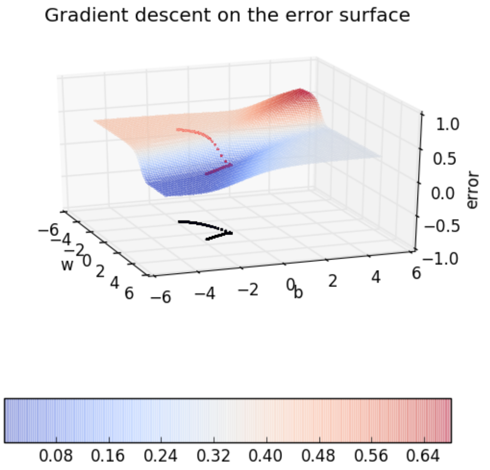
Gradient descent on the error surface



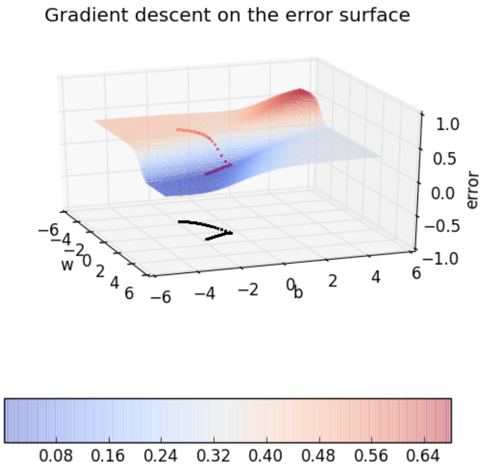
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



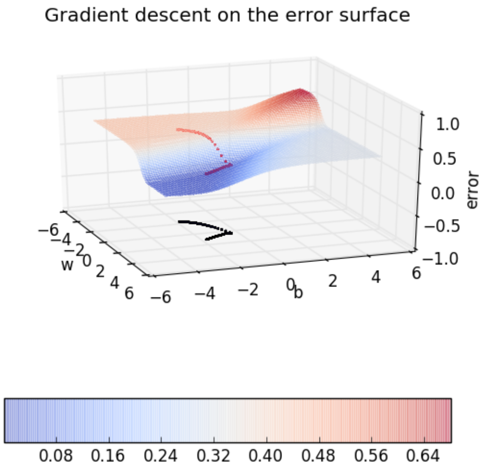
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



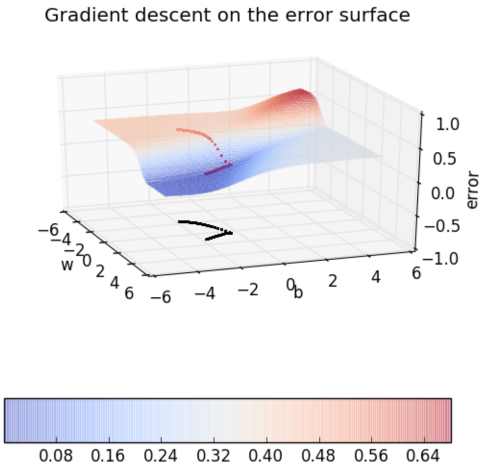
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



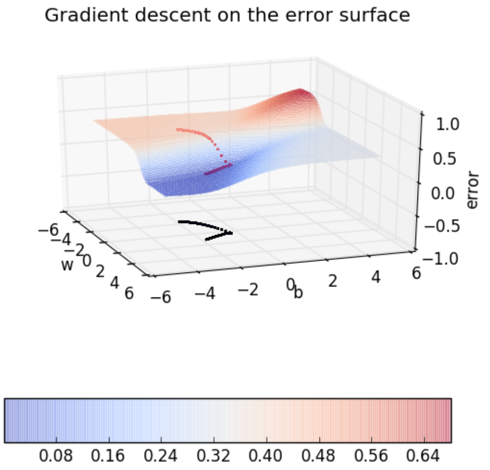
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



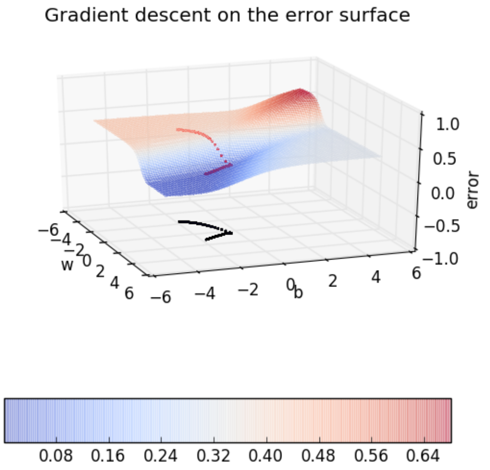
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



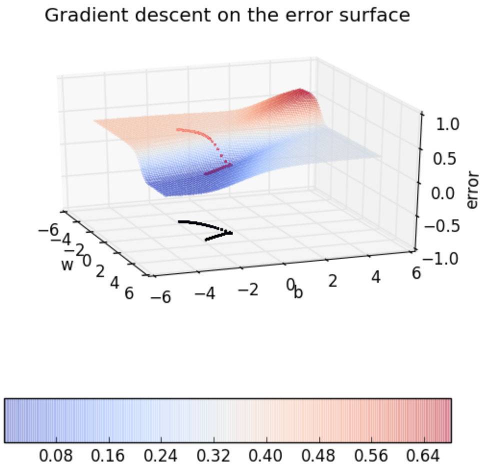
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



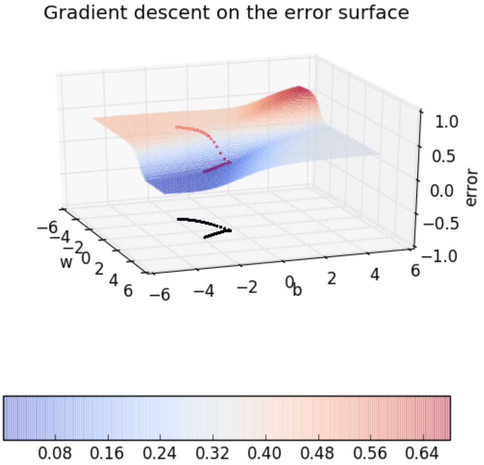
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



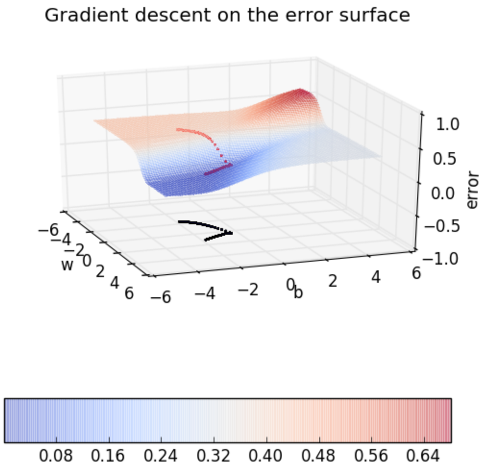
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



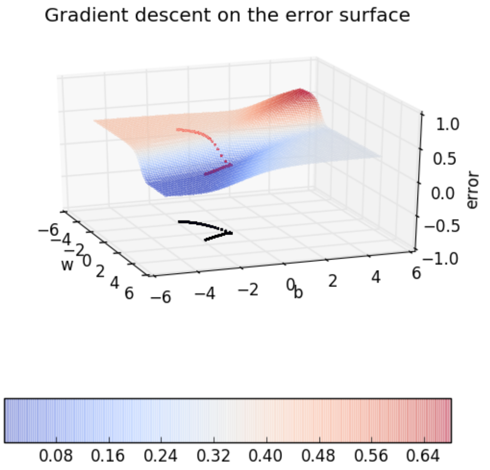
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

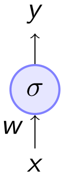


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

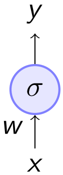


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

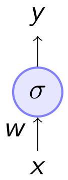




y
σ
w
x

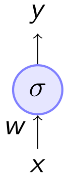


- We already saw how to train this network



- We already saw how to train this network

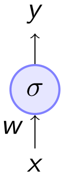
$$w = w - \eta \nabla w \quad where,$$



- We already saw how to train this network

$$w = w - \eta \nabla w \quad where,$$

$$\nabla w = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial w}$$

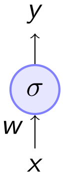


- We already saw how to train this network

$$w = w - \eta \nabla w \quad where,$$

$$\nabla w = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial w}$$

$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$

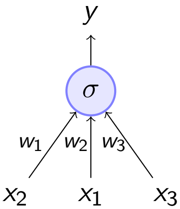


- We already saw how to train this network

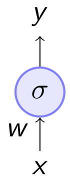
$$w = w - \eta \nabla w \quad where,$$

$$\nabla w = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial w}$$

$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



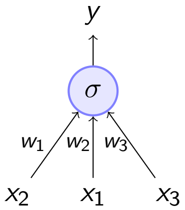
- What about a wider network with more inputs:



- We already saw how to train this network

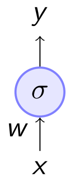
$$w = w - \eta \nabla w \quad where,$$

$$\nabla w = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial w}$$
$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



- What about a wider network with more inputs:

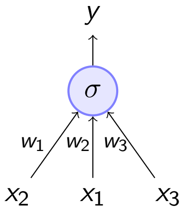
$$w_1 = w_1 - \eta \nabla w_1$$



- We already saw how to train this network

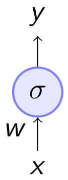
$$w = w - \eta \nabla w \quad where,$$

$$\nabla w = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial w}$$
$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



- What about a wider network with more inputs:

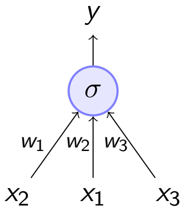
$$w_1 = w_1 - \eta \nabla w_1$$
$$w_2 = w_2 - \eta \nabla w_2$$



- We already saw how to train this network

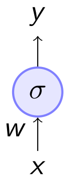
$$w = w - \eta \nabla w \quad where,$$

$$\nabla w = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial w}$$
$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



- What about a wider network with more inputs:

$$w_1 = w_1 - \eta \nabla w_1$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_3 = w_3 - \eta \nabla w_3$$

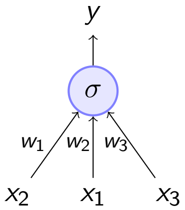


- We already saw how to train this network

$$w = w - \eta \nabla w \quad where,$$

$$\nabla w = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial w}$$

$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



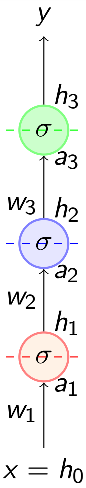
- What about a wider network with more inputs:

$$w_1 = w_1 - \eta \nabla w_1$$

$$w_2 = w_2 - \eta \nabla w_2$$

$$w_3 = w_3 - \eta \nabla w_3$$

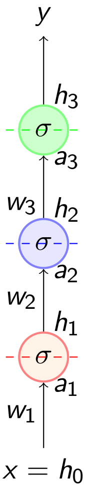
$$where, \nabla w_i = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * {\color{red}x_i}$$



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

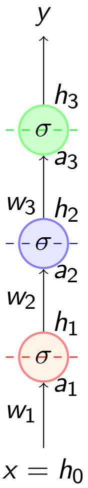
- What if we have a deeper network ?



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

- What if we have a deeper network ?
- We can now calculate ∇w_1 using chain rule:

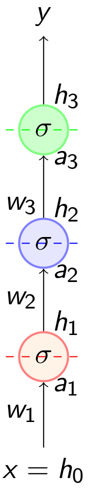


$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

- What if we have a deeper network ?
- We can now calculate ∇w_1 using chain rule:

$$\frac{\partial \mathscr{L}(\mathbf{w})}{\partial w_1} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$



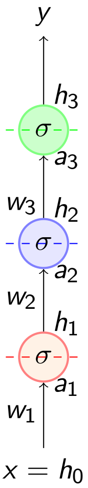
$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

- What if we have a deeper network ?
- We can now calculate ∇w_1 using chain rule:

$$\frac{\partial \mathscr{L}(\mathbf{w})}{\partial w_1} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$= \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} * \ldots\ldots\ldots\ldots\ldots * h_0$$



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

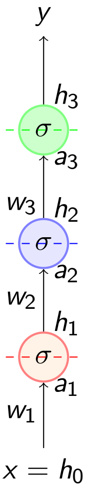
- What if we have a deeper network ?
- We can now calculate ∇w_1 using chain rule:

$$\frac{\partial \mathscr{L}(\mathbf{w})}{\partial w_1} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$= \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} * \ldots\ldots\ldots\ldots\ldots * h_0$$

- In general,

$$\nabla w_i = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} * \ldots\ldots\ldots\ldots\ldots * h_{i-1}$$



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

- What if we have a deeper network ?
- We can now calculate ∇w_1 using chain rule:

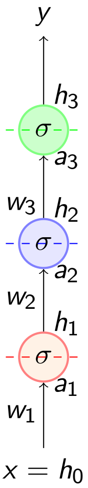
$$\frac{\partial \mathscr{L}(\mathbf{w})}{\partial w_1} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$= \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} * \ldots\ldots\ldots\ldots\ldots * h_0$$

- In general,

$$\nabla w_i = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} * \ldots\ldots\ldots\ldots\ldots * h_{i-1}$$

- Notice that ∇w_i is proportional to the corresponding input h_{i-1}



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

- What if we have a deeper network ?
- We can now calculate ∇w_1 using chain rule:

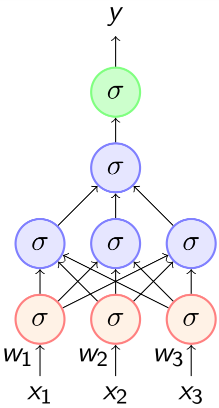
$$\frac{\partial \mathscr{L}(\mathbf{w})}{\partial w_1} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$= \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} * \ldots\ldots\ldots\ldots\ldots * h_0$$

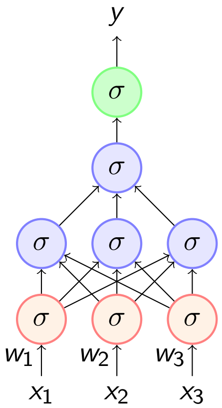
- In general,

$$\nabla w_i = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} * \ldots\ldots\ldots\ldots\ldots * h_{i-1}$$

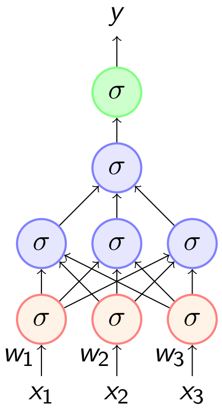
- Notice that ∇w_i is proportional to the corresponding input h_{i-1} (we will use this fact later)



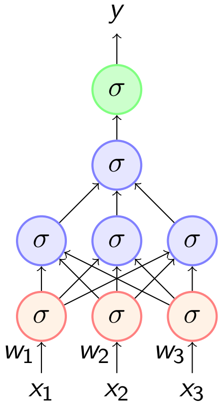
- What happens if we have a network which is deep and wide?



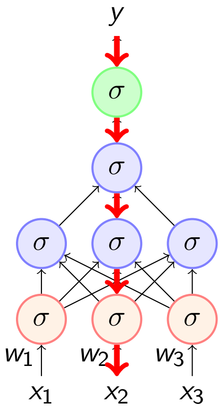
- What happens if we have a network which is deep and wide?



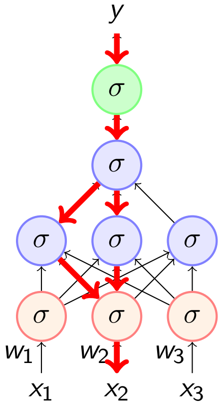
- What happens if we have a network which is deep and wide?
- How do you calculate $\nabla w_2 =$?



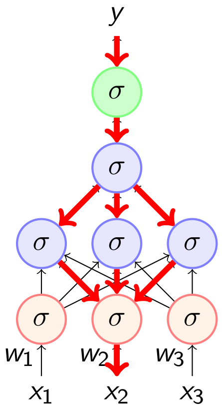
- What happens if we have a network which is deep and wide?
- How do you calculate $\nabla w_2 =?$
- It will be given by chain rule applied across multiple paths



- What happens if we have a network which is deep and wide?
- How do you calculate $\nabla w_2 = ?$
- It will be given by chain rule applied across multiple paths
- This is called the backpropagation algorithm

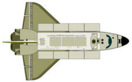


- What happens if we have a network which is deep and wide?
- How do you calculate $\nabla w_2 = ?$
- It will be given by chain rule applied across multiple paths
- This is called the backpropagation algorithm



- What happens if we have a network which is deep and wide?
- How do you calculate $\nabla w_2 = ?$
- It will be given by chain rule applied across multiple paths
- This is called the backpropagation algorithm

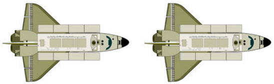
Convolutional Neural Networks



x_0

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

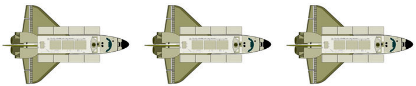
66/113



x_0 x_1

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

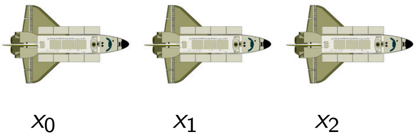
 66/113



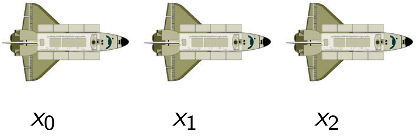
x_0 x_1 x_2

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

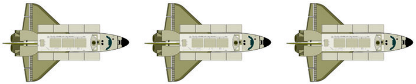
66/113



- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy

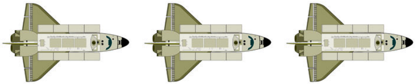


- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements



x_0 x_1 x_2

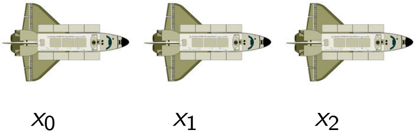
- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average



$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} =$$

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

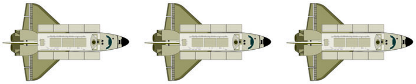
66/113



$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)_t$$

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

 66/113



x_0 x_1 x_2

$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (\underset{\text{input}}{x} \underset{\text{convolution}}{*} \underset{\text{filter}}{w})_t$$

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

66/113

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window

67/113

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter

67/113

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array (**w**) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0					
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5					
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80					

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96				

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11			

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16		

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

 67/113

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array (**w**) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	1	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70

S	1.80	1.96	2.11	2.16	2.28	2.42

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	1	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	2.42

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	1	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	2.42

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2d input also?

- We can think of images as 2d inputs

- We can think of images as 2d inputs
- We would now like to use a 2d filter (mxn)

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a,j-b} K_{a,b}$$

- We can think of images as 2d inputs
- We would now like to use a 2d filter (mxn)
- First let us see what the 2d formula looks like

- We can think of images as 2d inputs
- We would now like to use a 2d filter (mxn)
- First let us see what the 2d formula looks like
- This formula looks at all the preceding neighbours $(i-a, j-b)$

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a,j-b} K_{a,b}$$

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a,j+b} K_{a,b}$$

- We can think of images as 2d inputs
- We would now like to use a 2d filter (mxn)
- First let us see what the 2d formula looks like
- This formula looks at all the preceding neighbours $(i - a, j - b)$
- In practice, we use the following formula which looks at the succeeding neighbours

- Let us apply this idea to a toy example and see the results

69/113

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz		

- Let us apply this idea to a toy example and see the results

69/113

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	

- Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz

- Let us apply this idea to a toy example and see the results

 69/113

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz		

- Let us apply this idea to a toy example and see the results

- Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz	fw+gx+jy+kz	

- Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz	fw+gx+jy+kz	gw+hx+ky+ℓz

 69/113

- For the rest of the discussion we will use the following formula for convolution

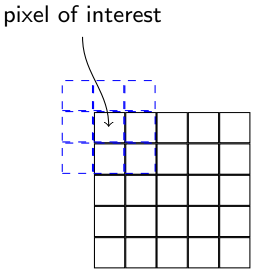
70/113

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

- For the rest of the discussion we will use the following formula for convolution

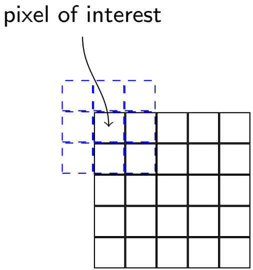
70/113

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$



- For the rest of the discussion we will use the following formula for convolution
- In other words we will assume that the kernel is centered on the pixel of interest

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$



- For the rest of the discussion we will use the following formula for convolution
- In other words we will assume that the kernel is centered on the pixel of interest
- So we will be looking at both preceeding and succeeding neighbors

 70/113

Let us see some examples of 2d convolutions applied to images

71/113

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

72/113

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

blurs the image

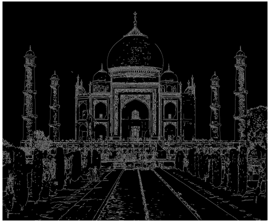
$$* \begin{array}{ccc} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{array} =$$

$$* \begin{array}{ccc} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{array} =$$

sharpens the image

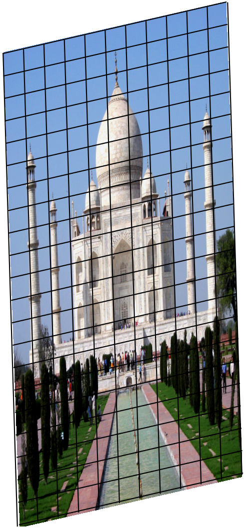
$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



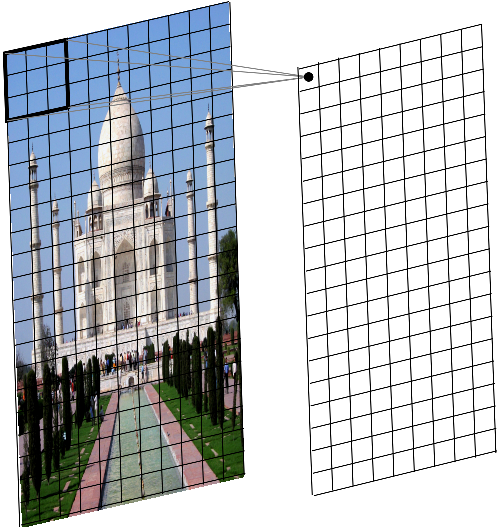
detects the edges

We will now see a working example of 2D convolution.

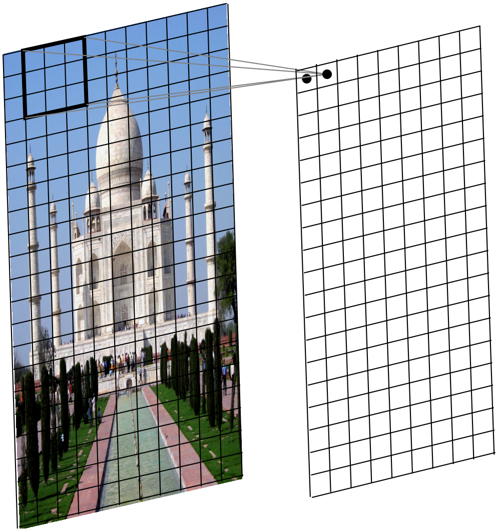


- We just slide the kernel over the input image

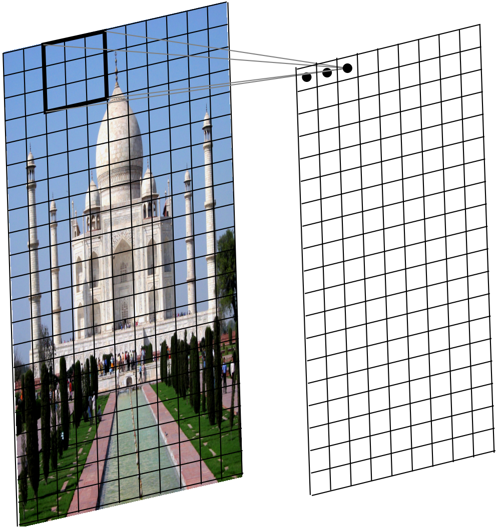
76/113



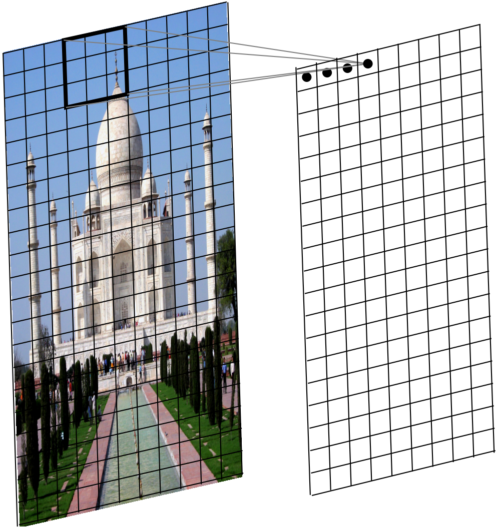
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

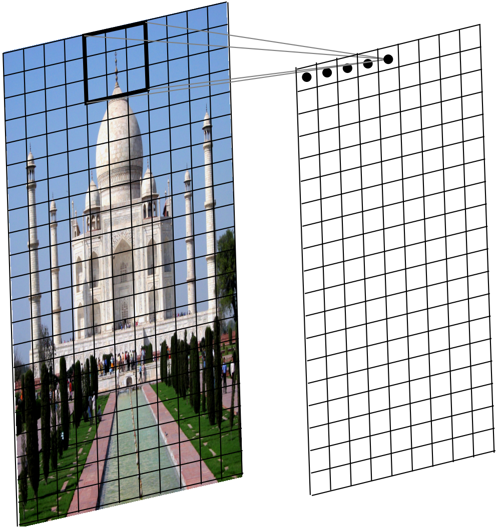


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



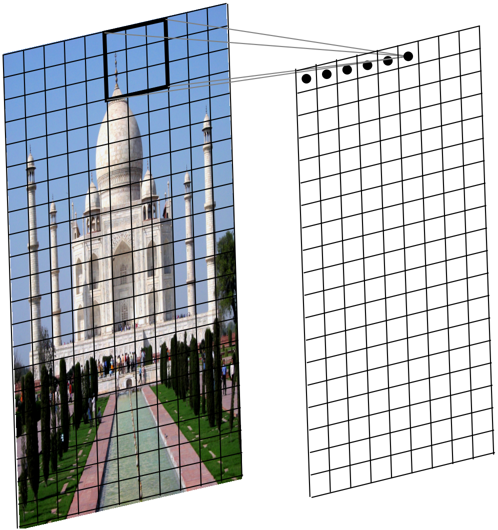
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

76/113

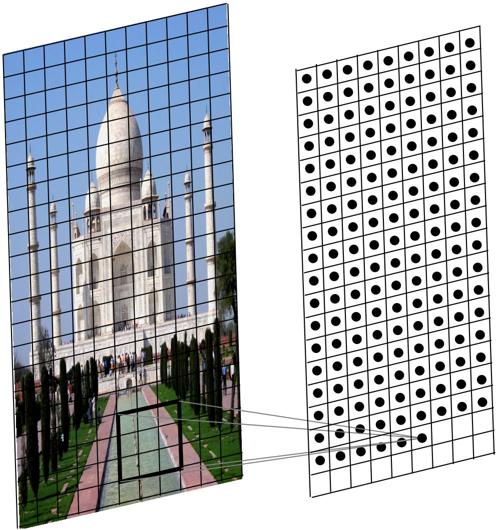


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

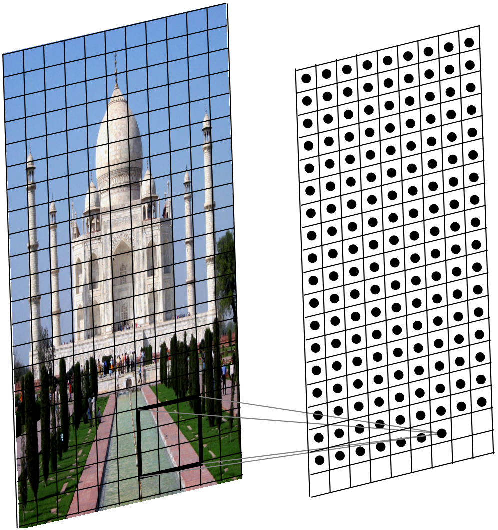
 76/113



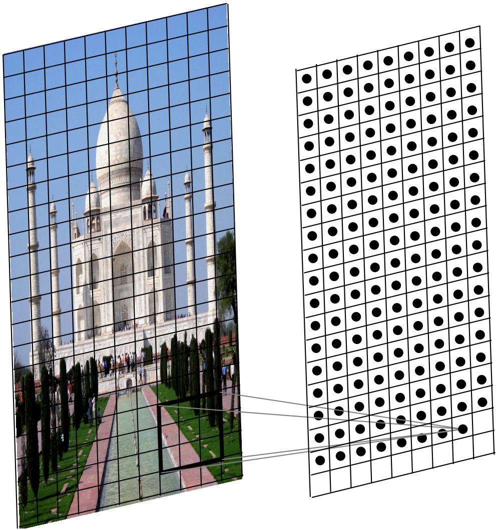
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



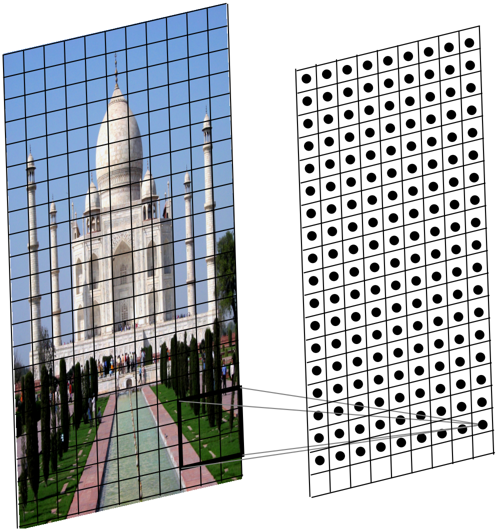
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

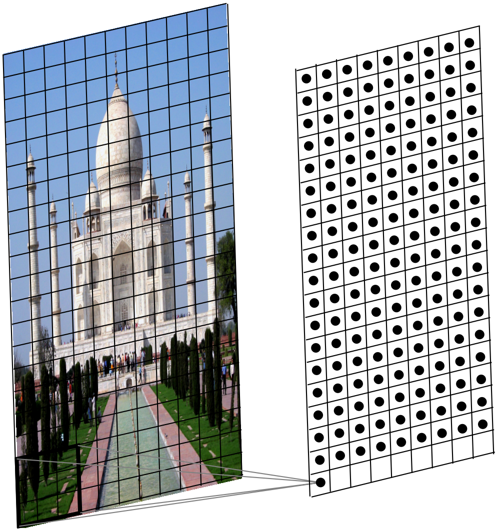


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



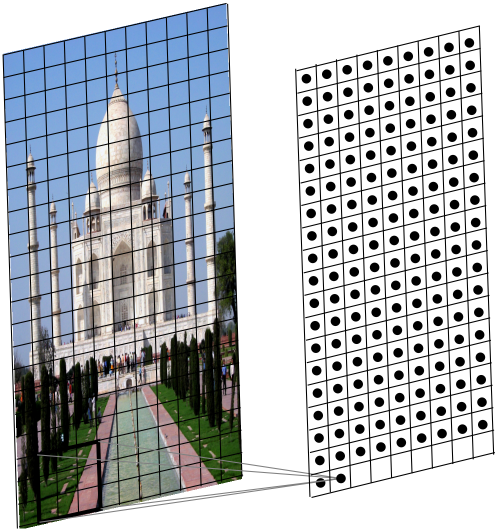
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

76/113



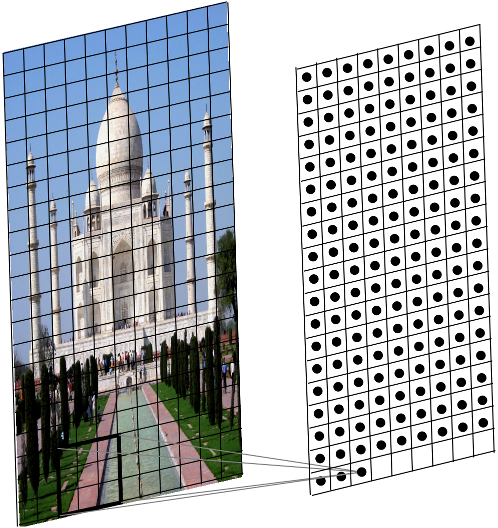
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

76/113

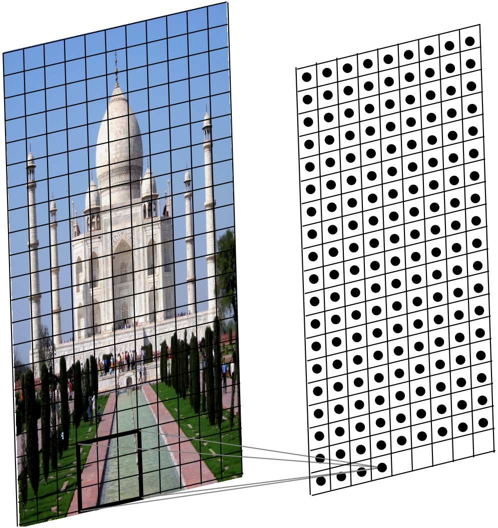


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

76/113

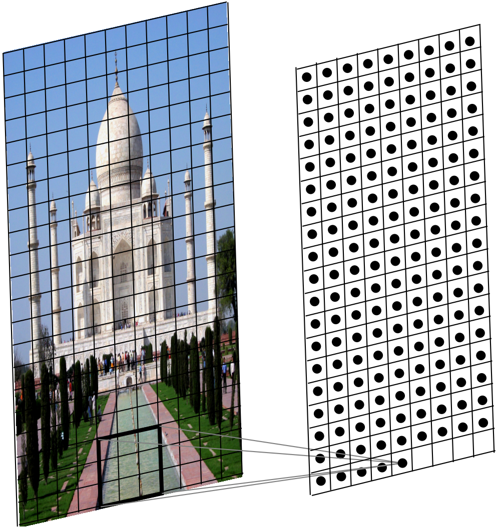


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

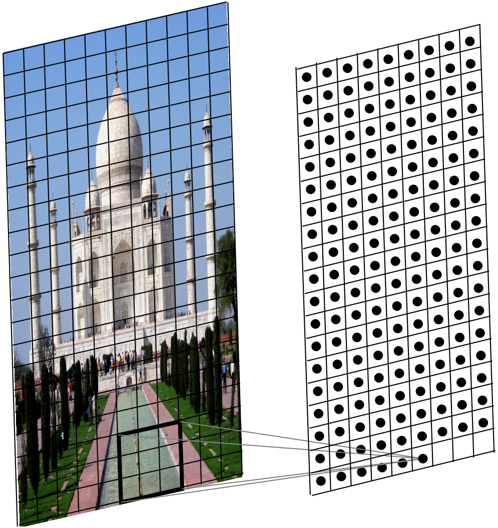


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

76/113

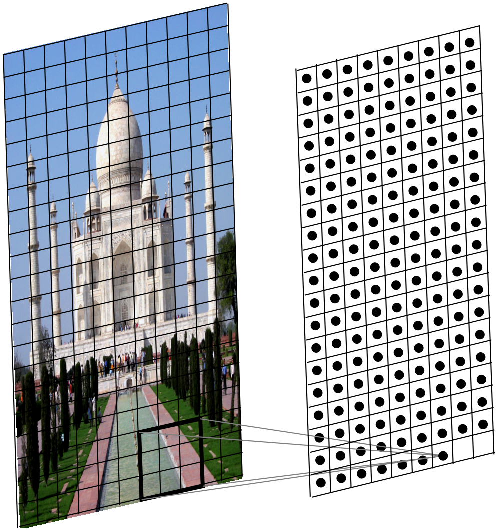


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

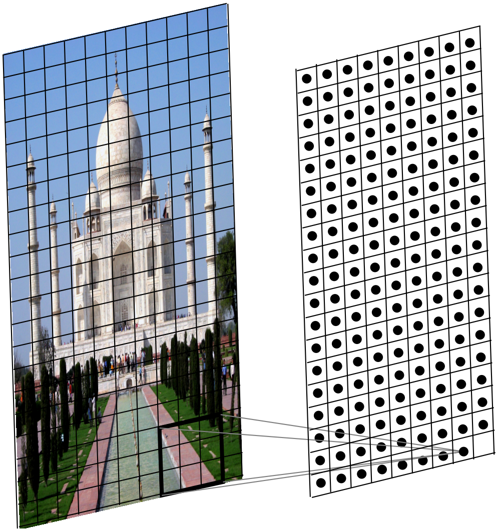


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

76/113

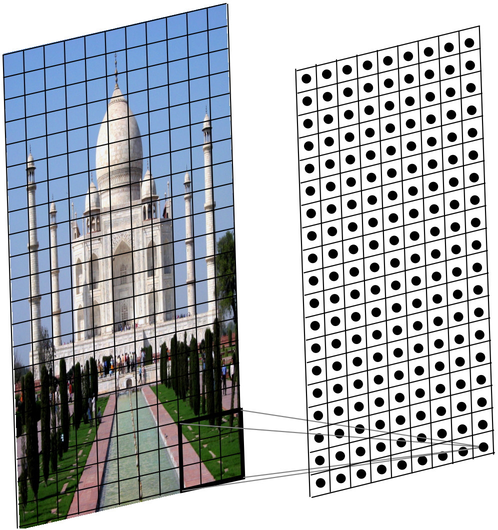


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



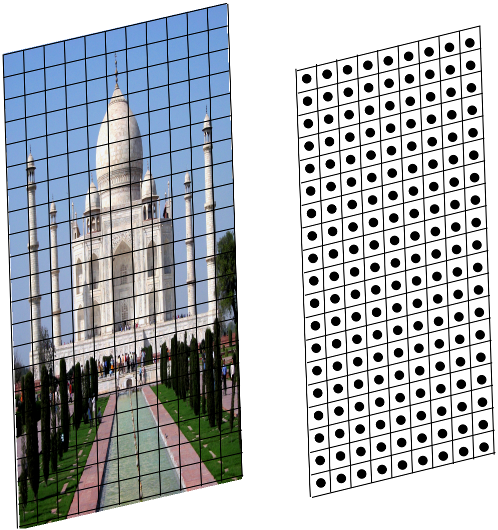
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

76/113



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.

76/113



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

76/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input

77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input

A	B	C	B	A	B	C

 77/113

Question

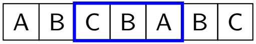
- In 1D convolution, we slide a one dimensional filter over a one dimensional input

A	B	C	B	A	B	C

77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input



77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input

A	B	C	B	A	B	C

 77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input

A	B	C	B	$\boxed{A}$	$\boxed{B}$	$\boxed{C}$

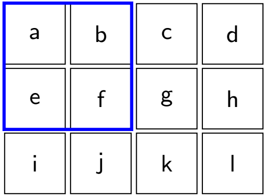
 77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input
- In 2D convolution, we slide a two dimenstional filter over a two dimensional output

Question

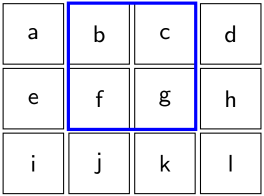
- In 1D convolution, we slide a one dimensional filter over a one dimensional input
- In 2D convolution, we slide a two dimensional filter over a two dimensional output



77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input
- In 2D convolution, we slide a two dimensional filter over a two dimensional output



77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input
- In 2D convolution, we slide a two dimensional filter over a two dimensional output

a	b	c	d
e	f	g	h
i	j	k	l

77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input
- In 2D convolution, we slide a two dimensional filter over a two dimensional output

a	b	c	d
e	f	g	h
i	j	k	l

77/113

a	b	c	d
e	f	g	h
i	j	k	l

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input
- In 2D convolution, we slide a two dimensional filter over a two dimensional output

77/113

Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input
- In 2D convolution, we slide a two dimensional filter over a two dimensional output

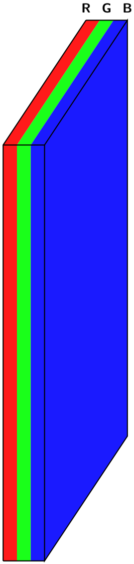
a	b	c	d
e	f	g	h
i	j	k	l

77/113

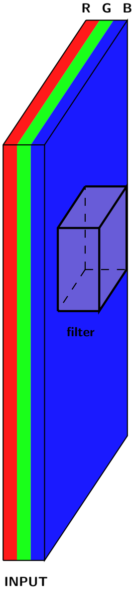
Question

- In 1D convolution, we slide a one dimensional filter over a one dimensional input
- In 2D convolution, we slide a two dimenstional filter over a two dimensional output
- What would a 3D convolution look like?

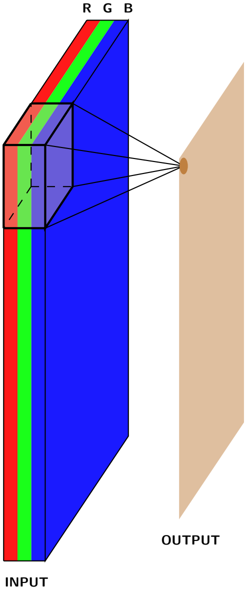
77/113



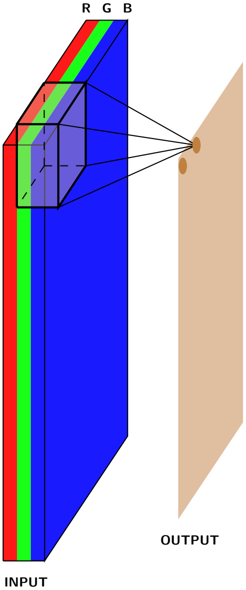
- What would a 3D filter look like?



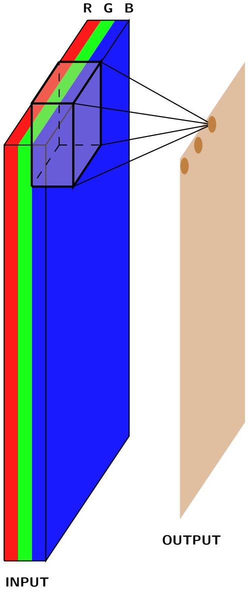
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume



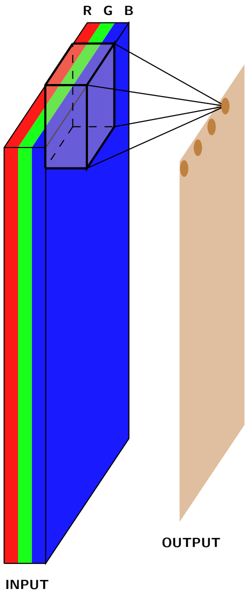
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



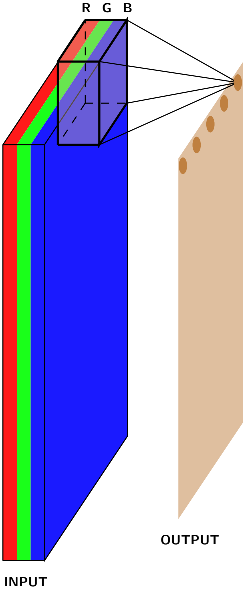
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



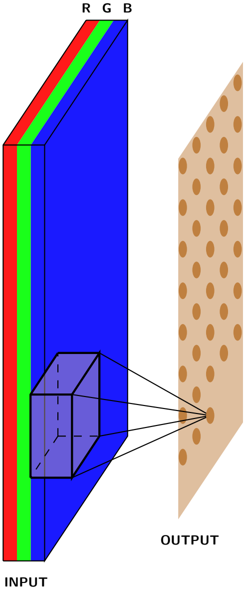
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



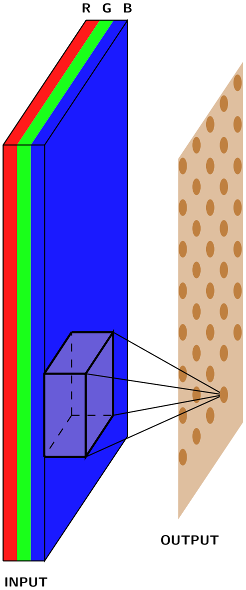
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



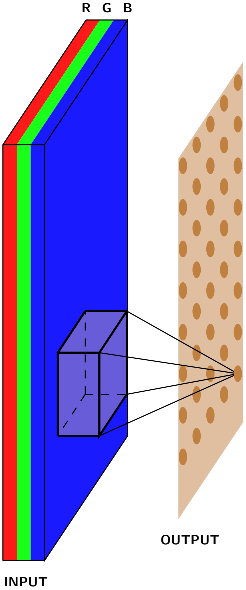
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



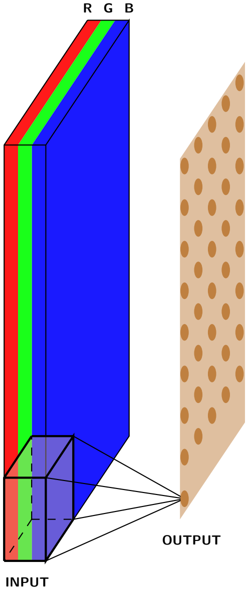
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



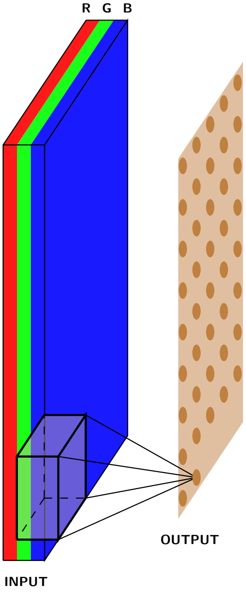
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



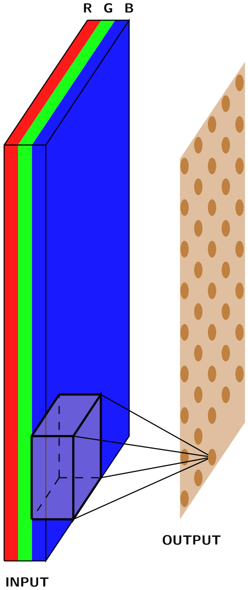
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



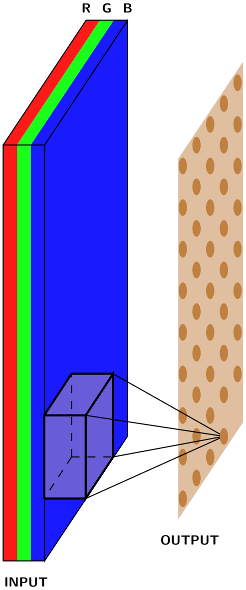
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



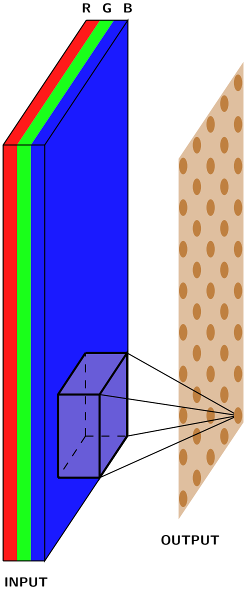
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



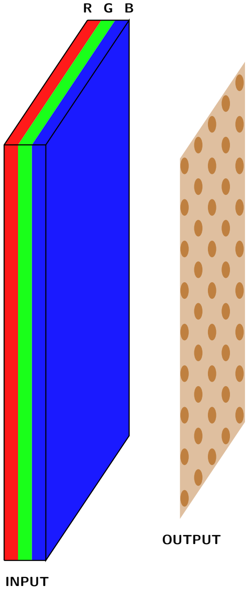
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



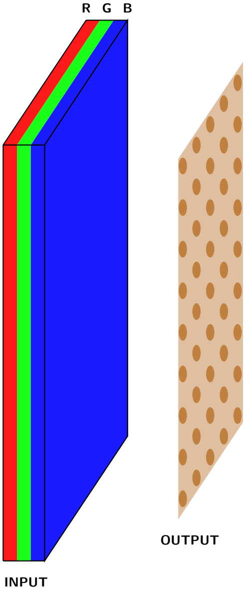
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.



- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.
- Note that the filter always extends the depth of the image.



- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.
- Note that the filter always extends the depth of the image.
- Also note that 3D filter applied to a 3D input results in a 2D output.



- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation.
- Note that the filter always extends the depth of the image.
- Also note that 3D filter applied to a 3D input results in a 2D output.
- Once again we can apply multiple filters to get multiple feature maps.

- So far we have not said anything explicit about the dimensions of the

 79/113

- So far we have not said anything explicit about the dimensions of the
    1. inputs

 79/113

- So far we have not said anything explicit about the dimensions of the
  1. inputs
  2. filters

79/113

- So far we have not said anything explicit about the dimensions of the
  1. inputs
  2. filters
  3. outputs

79/113

- So far we have not said anything explicit about the dimensions of the
  1. inputs
  2. filters
  3. outputs

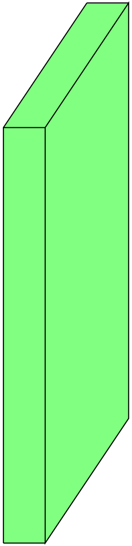
  and the relations between them

 79/113

- So far we have not said anything explicit about the dimensions of the
  1. inputs
  2. filters
  3. outputs

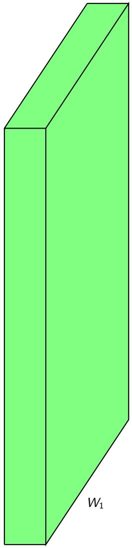
  and the relations between them
- We will see how they are related but before that we will define a few quantities

79/113



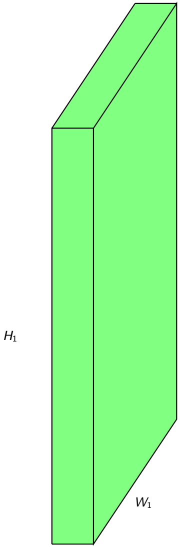
- We first define the following quantities

80/113



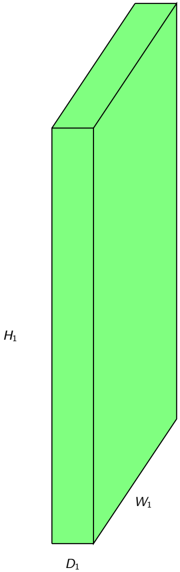
- We first define the following quantities
- Width (W_1),

80/113



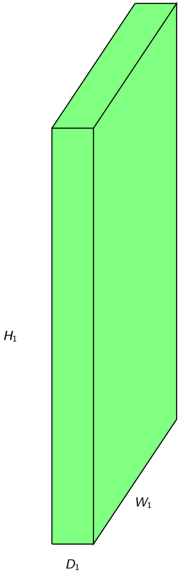
- We first define the following quantities
- Width (W_1), Height (H_1)

80/113



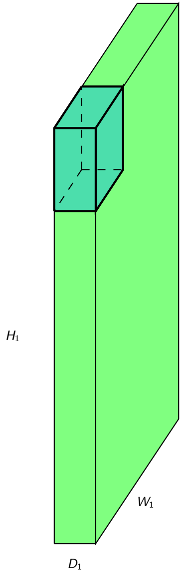
- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input

 80/113



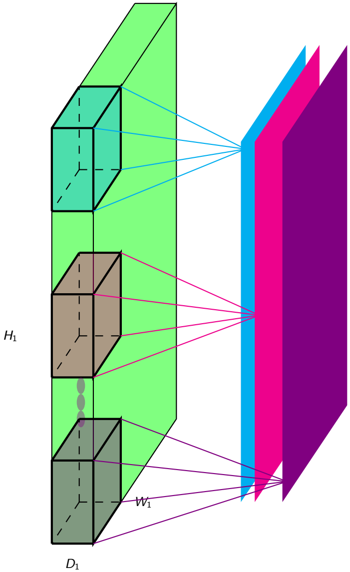
- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)

80/113

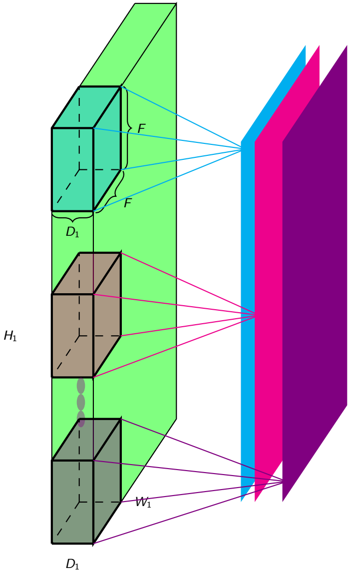


- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)

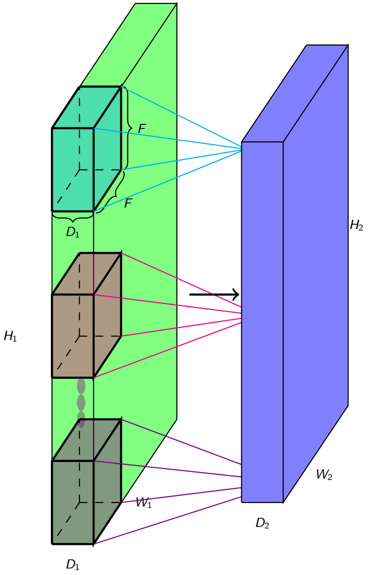
80/113



- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K



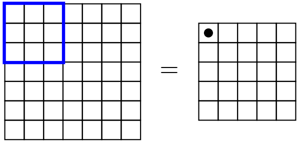
- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extend (F) of each filter (the depth of each filter is same as the depth of each input



- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extend (F) of each filter (the depth of each filter is same as the depth of each input
- The output is $W_2 \times H_2 \times D_2$ (we will soon see a formula for computing W_2, H_2 and D_2

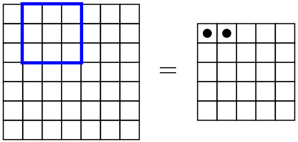
 80/113

- Let us compute the dimension (W_2, H_2) of the output

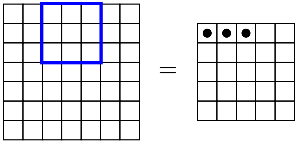


- Let us compute the dimension (W_2, H_2) of the output

81/113

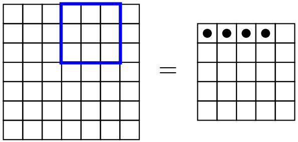


- Let us compute the dimension (W_2, H_2) of the output

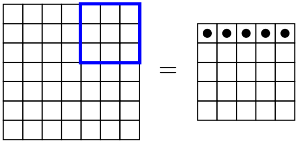


- Let us compute the dimension (W_2, H_2) of the output

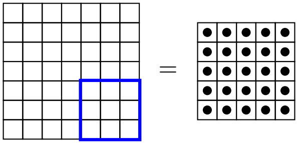
81/113



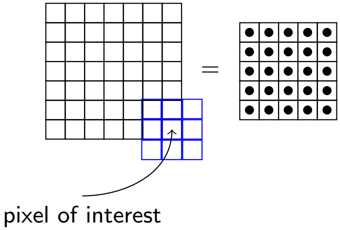
- Let us compute the dimension (W_2, H_2) of the output



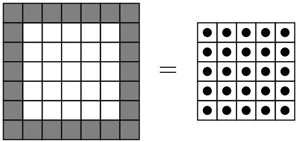
- Let us compute the dimension (W_2, H_2) of the output



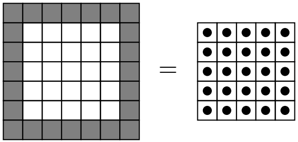
- Let us compute the dimension (W_2, H_2) of the output



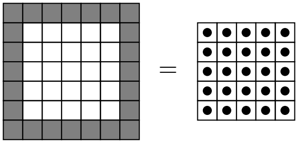
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary



- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary

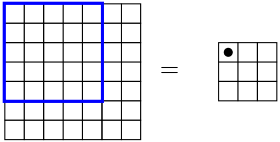


- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)

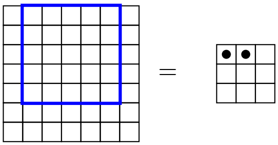


- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input

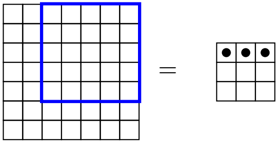
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels



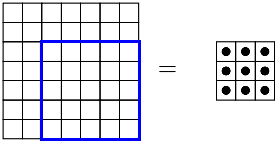
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel



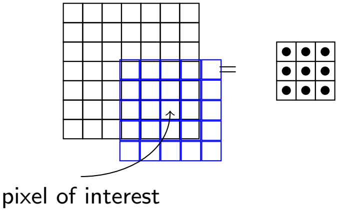
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller ouput now



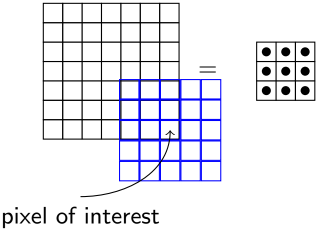
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller ouput now



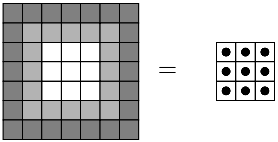
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller ouput now



- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller ouput now



- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller ouput now



$$\textit{In general,} W_2 = W_1 - F + 1$$
$$H_2 = H_1 - F + 1$$

We will refine this formula further

- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller ouput now

82/113

- What if we want the output to be of same size as the input?

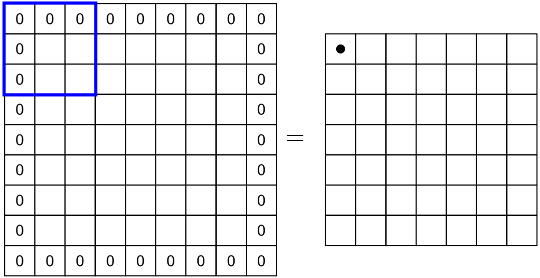
83/113

- What if we want the output to be of same size as the input?
- We can use something known as padding

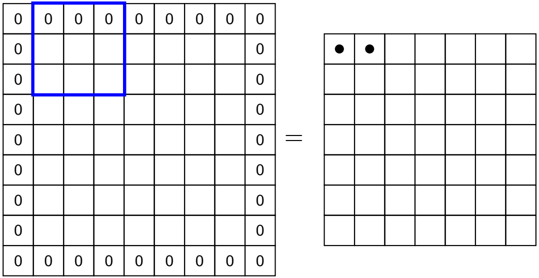
83/113

- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners

83/113

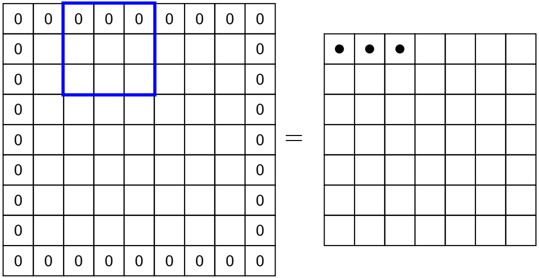


- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad P $= 1$ with a 3×3 kernel

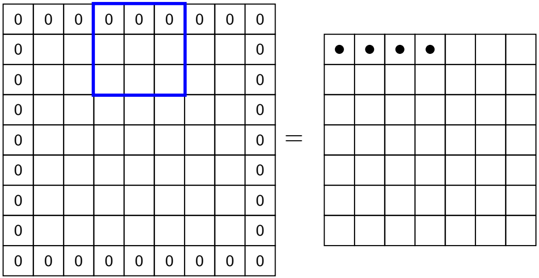


- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

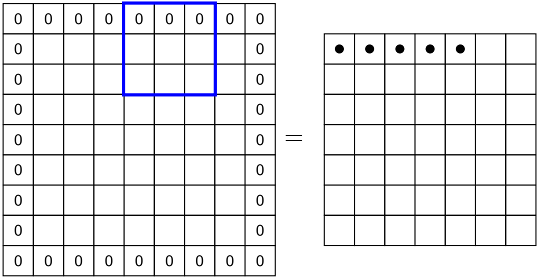
83/113



- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

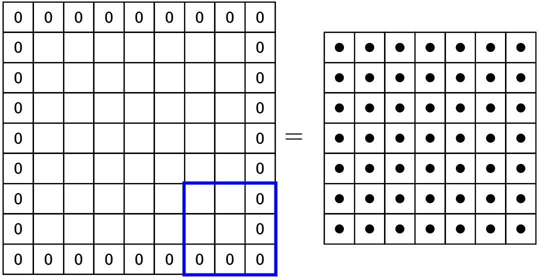


- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right



- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

83/113



We now have,

$$W_2 = W_1 - F + 2P + 1$$

$$H_2 = H_1 - F + 2P + 1$$

We will refine this formula further

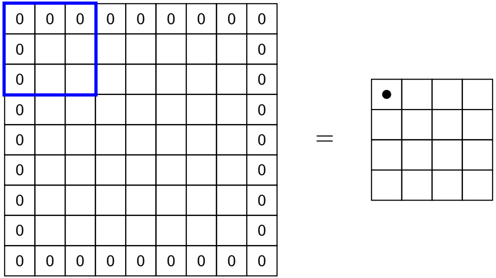
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad P = 1 with a 3 × 3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

- What does the stride S do?

84/113

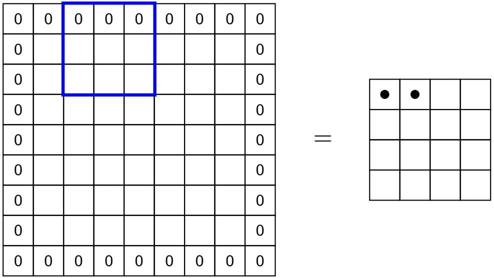
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)

84/113



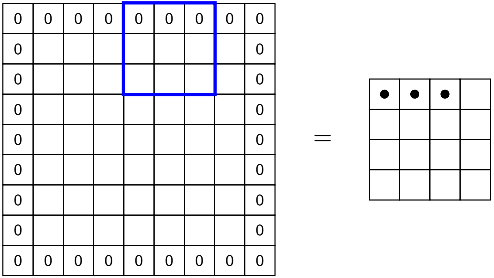
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

84/113

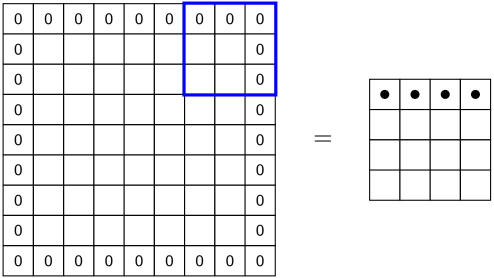


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

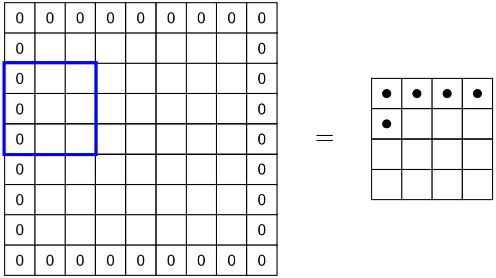
84/113



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

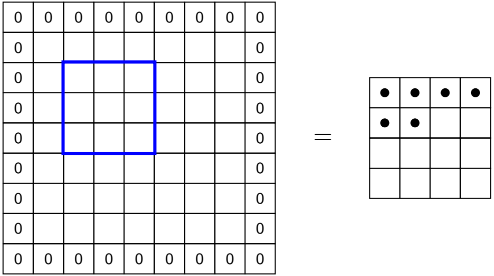


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



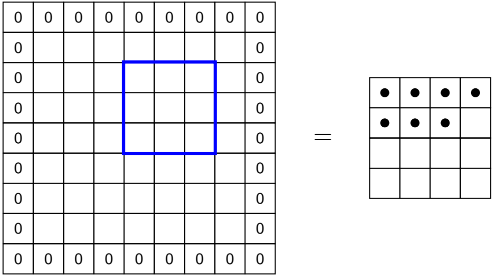
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

 84/113



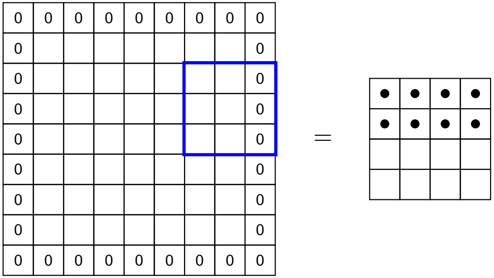
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

84/113



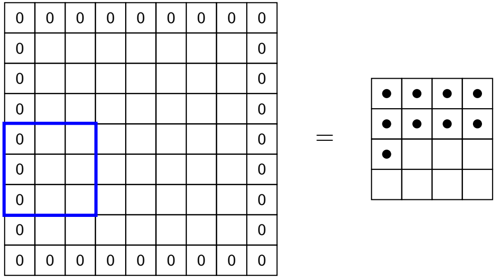
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

84/113



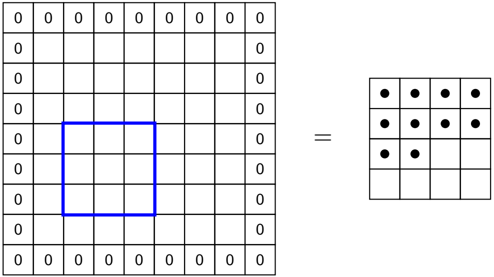
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

 84/113



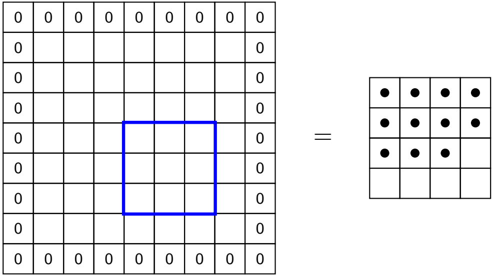
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

 84/113

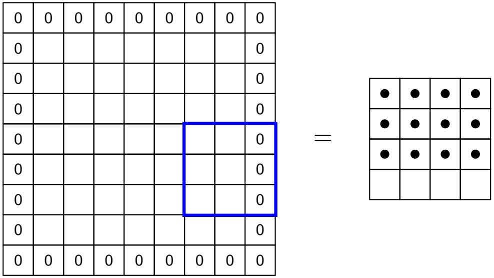


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

84/113

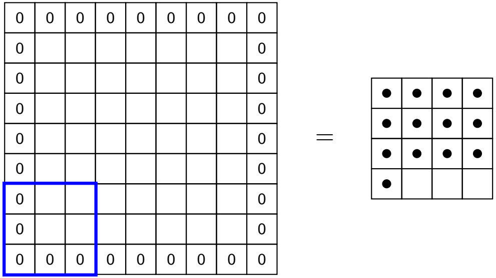


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

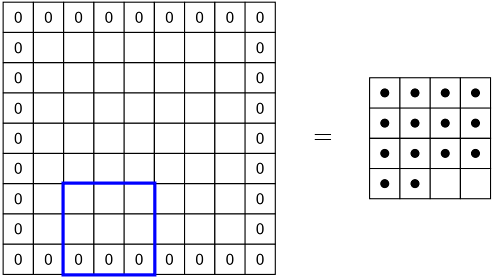


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

84/113

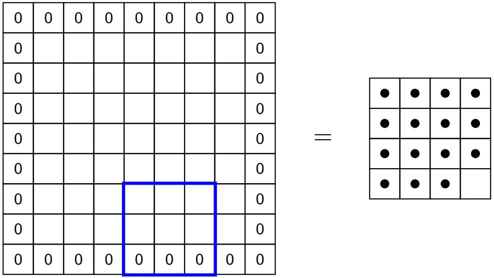


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

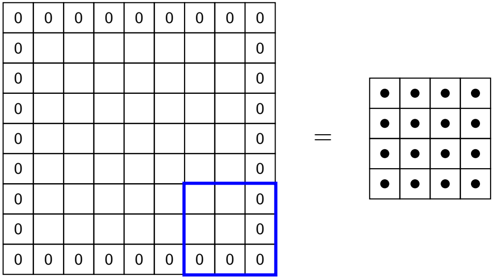
 84/113



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

So what should our final formula look like,

84/113

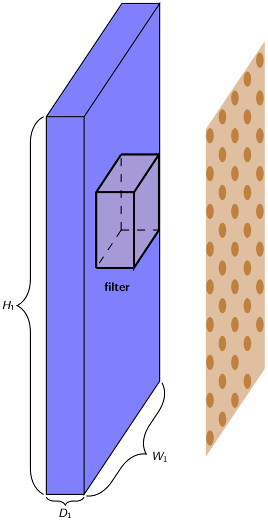


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

So what should our final formula look like,

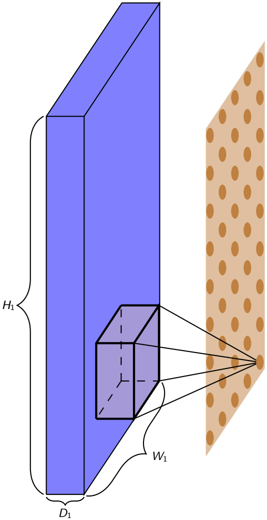
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$



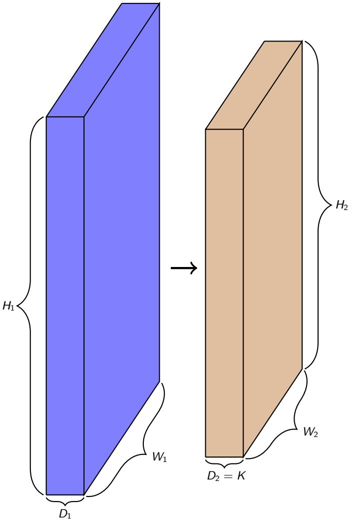
- Finally, coming to the depth of the output.

85/113



- Finally, coming to the depth of the output.
- Each filter gives us one 2d output.

 85/113

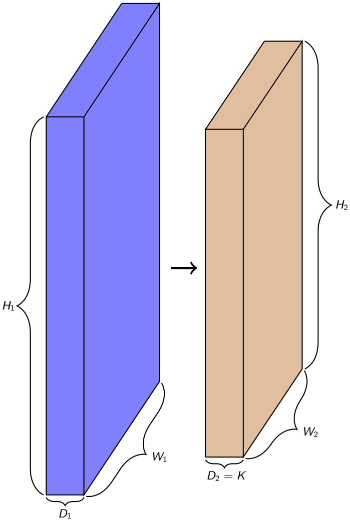


$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

- Finally, coming to the depth of the output.
- Each filter gives us one 2d output.
- K filters will give us K such 2D outputs

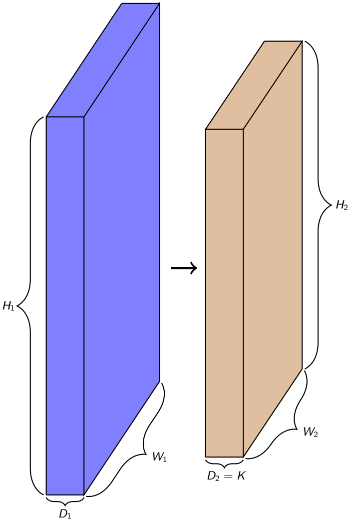


$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

- Finally, coming to the depth of the output.
- Each filter gives us one 2d output.
- K filters will give us K such 2D outputs
- We can think of the resulting output as $K \times W_2 \times H_2$ volume



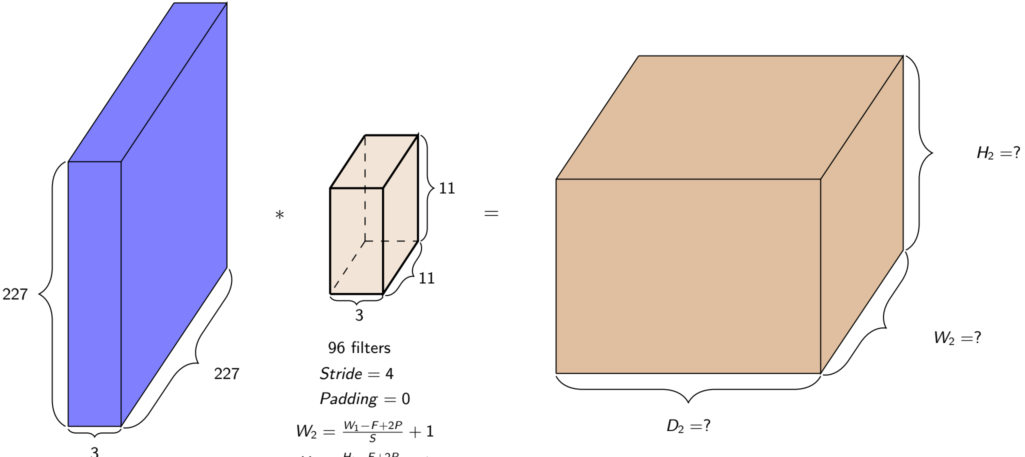
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

- Finally, coming to the depth of the output.
- Each filter gives us one 2d output.
- K filters will give us K such 2D outputs
- We can think of the resulting output as $K \times W_2 \times H_2$ volume
- Thus $D_2 = K$

Let us do a few exercises



96 filters

$Stride = 4$

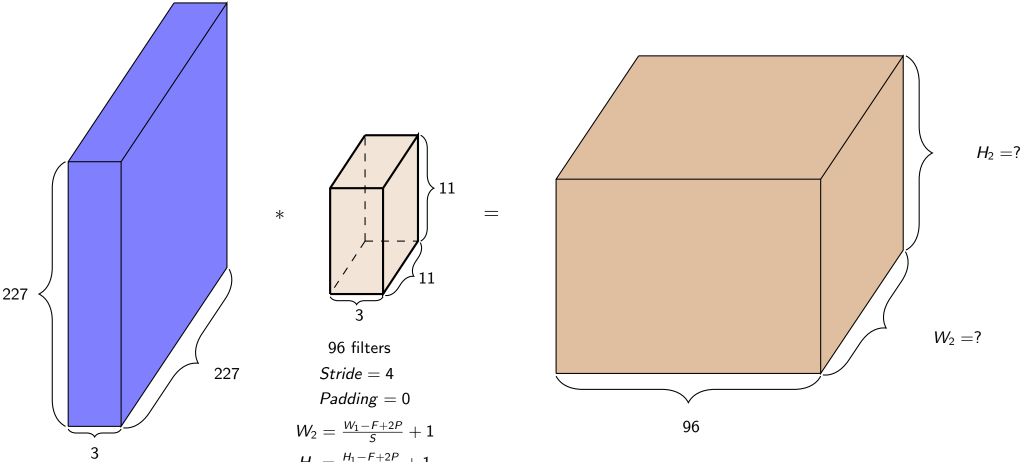
$Padding = 0$

$W_2 = \frac{W_1 - F + 2P}{S} + 1$

$H_2 = \frac{H_1 - F + 2P}{S} + 1$

86/113

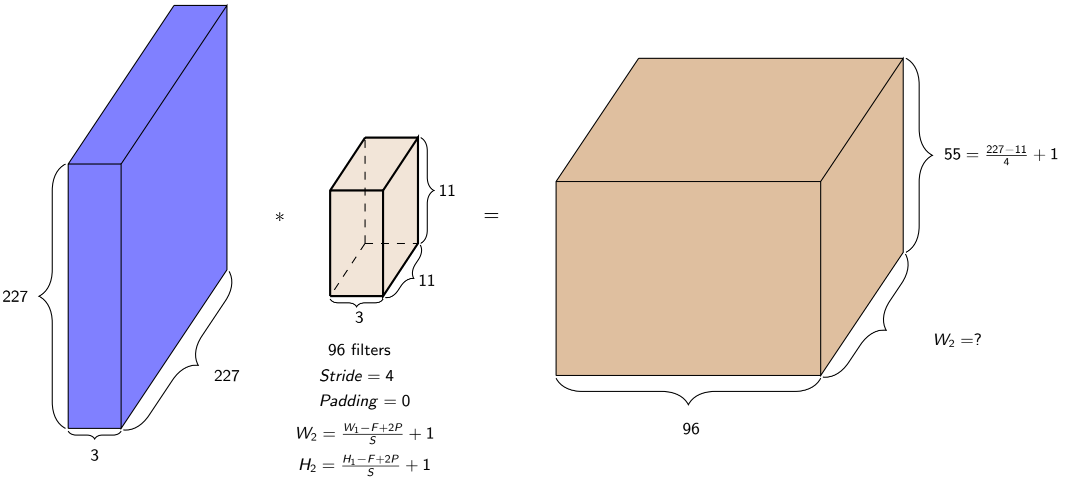
Let us do a few exercises



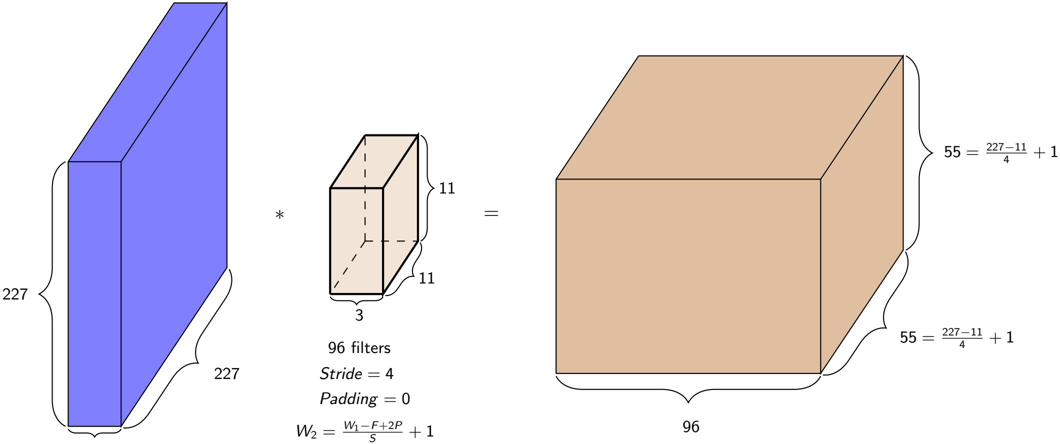
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Let us do a few exercises



Let us do a few exercises



96 filters

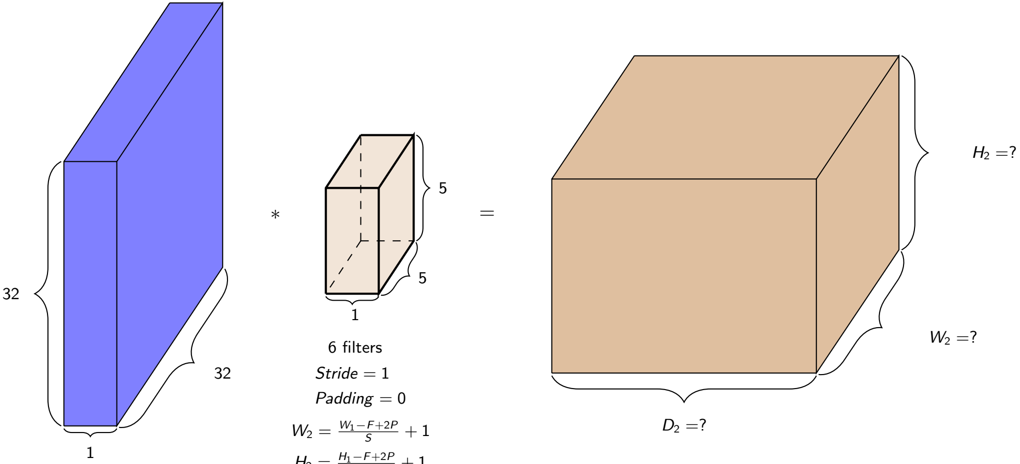
$Stride = 4$

$Padding = 0$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Let us do a few exercises



6 filters

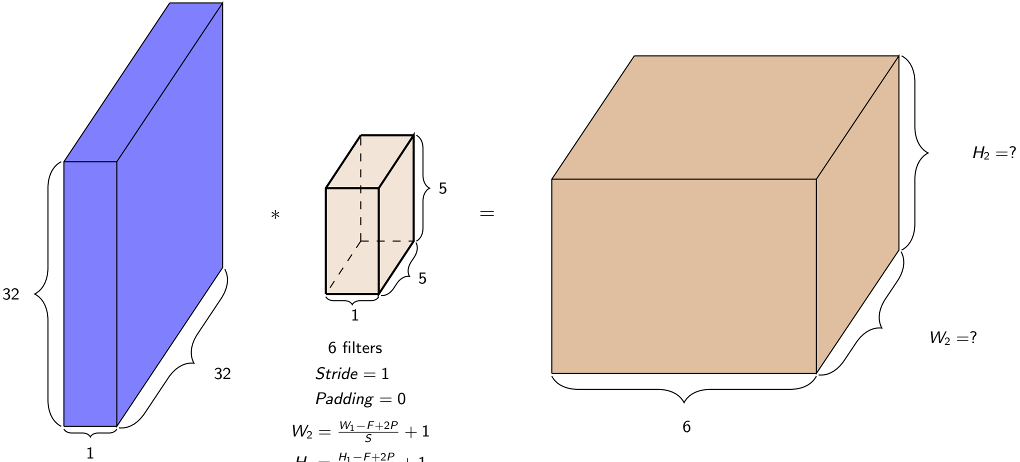
$Stride = 1$

$Padding = 0$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Let us do a few exercises



6 filters

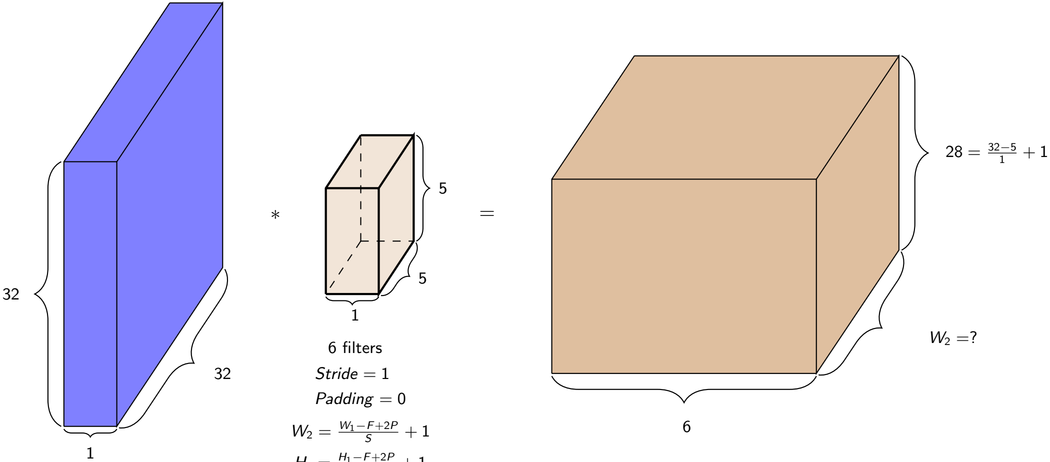
$Stride = 1$

$Padding = 0$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Let us do a few exercises



6 filters

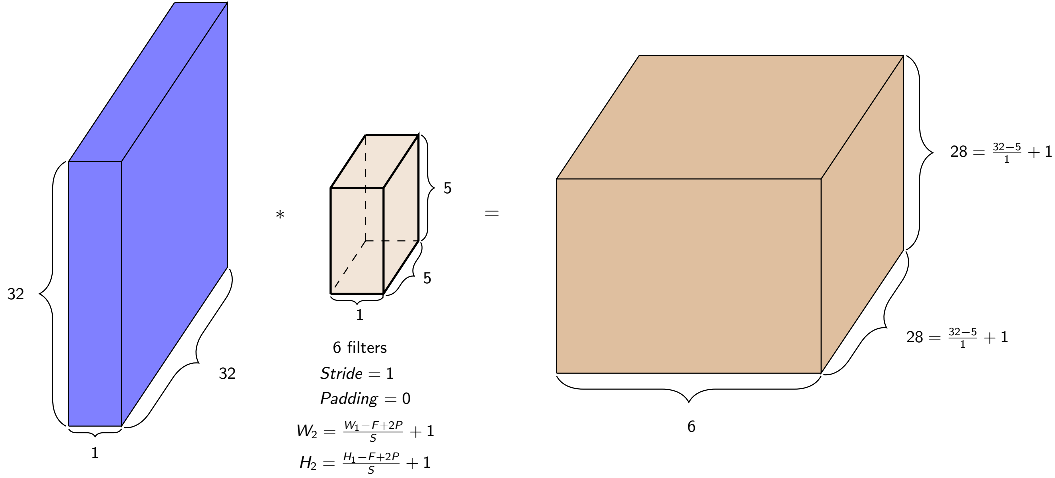
$Stride = 1$

$Padding = 0$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Let us do a few exercises



 87/113

Putting things into perspective

- What is the connection between this operation (convolution) and neural networks?

88/113

Putting things into perspective

- What is the connection between this operation (convolution) and neural networks?
- We will try to understand this by considering the task of "image classification".

88/113

 89/113

Features

Raw pixels

89/113

Features

$\xrightarrow{\textit{Raw pixels}}$

$\longrightarrow$ car, bus, monument, flower

Features

Raw pixels ⟶

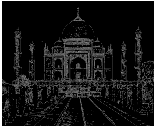
⟶ car, bus, monument, flower

Features

Raw pixels ⟶

⟶ car, bus, monument, flower

Edge Detector ⟶



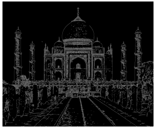
 89/113

Features

Raw pixels →

→ car, bus, monument, flower

Edge Detector →



→ car, bus, monument, flower

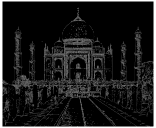
89/113

Features

Raw pixels $\longrightarrow$

$\longrightarrow$ car, bus, monument, flower

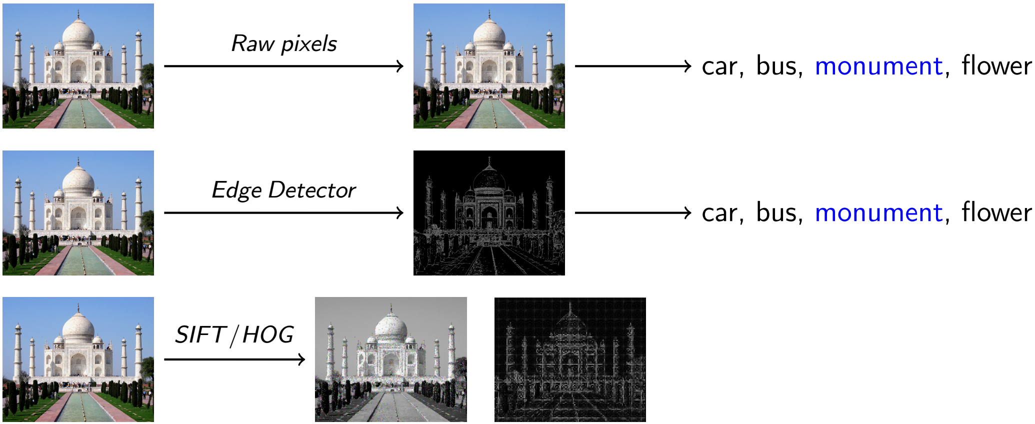
Edge Detector $\longrightarrow$



$\longrightarrow$ car, bus, monument, flower

 89/113

Features

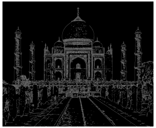

Raw pixels
car, bus, monument, flower
Edge Detector
car, bus, monument, flower
SIFT / HOG

Features

Raw pixels $\longrightarrow$

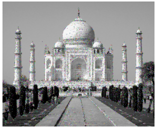
$\longrightarrow$ car, bus, monument, flower

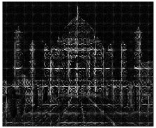
Edge Detector $\longrightarrow$



$\longrightarrow$ car, bus, monument, flower

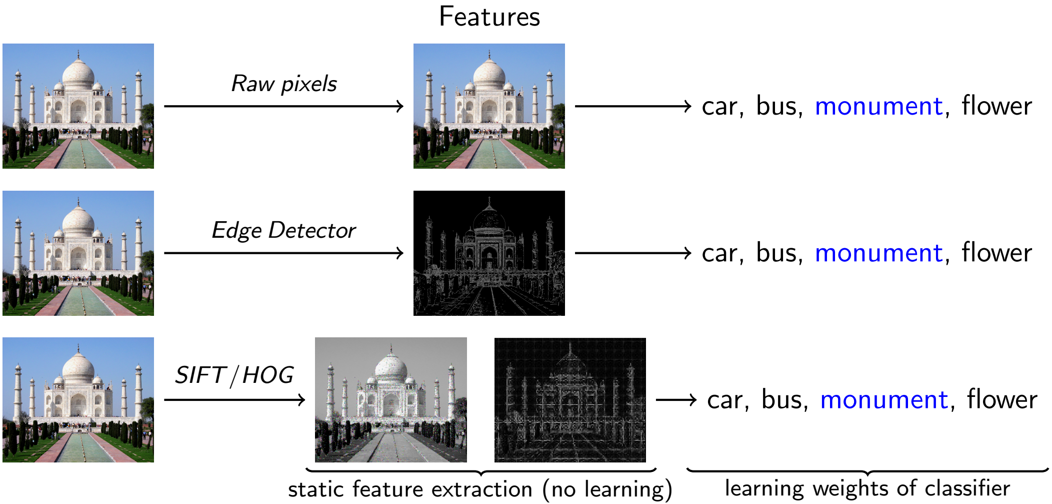
SIFT / HOG $\longrightarrow$





$\longrightarrow$ car, bus, monument, flower

89/113


Features
Raw pixels
car, bus, monument, flower
Edge Detector
car, bus, monument, flower
SIFT / HOG
car, bus, monument, flower
static feature extraction (no learning)
learning weights of classifier

89/113

Input | Features | Classifier

$\longrightarrow$

$\longrightarrow$ car, bus, monument, flower

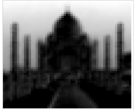
$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

- Instead of using handcrafted kernels such as edge detectors **can we learn meaningful kernels/filters in addition to learning the weights of the classifier?**

Input | Features | Classifier

$\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$



$\longrightarrow$ car, bus, monument, flower

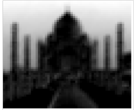
$$\begin{matrix} -1.21358689e-03 & 3.23652686e-03 & \cdots & \cdots & -2.06615720e-02 \\ -1.52757822e-03 & 2.36130832e-03 & \cdots & \cdots & -1.19824838e-02 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -8.25322699e-04 & -5.14897937e-03 & \cdots & \cdots & -9.90395527e-03 \end{matrix}$$

- Instead of using handcrafted kernels such as edge detectors **can we learn meaningful kernels/filters in addition to learning the weights of the classifier?**

Input $\qquad$ Features $\qquad$ Classifier

 $\longrightarrow$  $\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

 $\longrightarrow$  $\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} -1.21358689\text{e-}03 & 3.23652686\text{e-}03 & \cdots & \cdots & -2.06615720\text{e-}02 \\ -1.52757822\text{e-}03 & 2.36130832\text{e-}03 & \cdots & \cdots & -1.19824838\text{e-}02 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -8.25322699\text{e-}04 & -5.14897937\text{e-}03 & \cdots & \cdots & -9.90395527\text{e-}03 \end{matrix} \longleftarrow \text{Learn these weights}$$

- Instead of using handcrafted kernels such as edge detectors **can we learn meaningful kernels/filters in addition to learning the weights of the classifier?**

 90/113

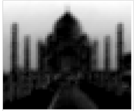
Input | Features | Classifier

$\longrightarrow$

$\longrightarrow$ car, bus, monument, flower

0	0	0	0	0
0	1	1	1	0
0	1	-8	1	0
0	1	1	1	0
0	0	0	0	0

$\longrightarrow$



$\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} -1.21358689e{-}03 & 3.23652686e{-}03 & \cdots & \cdots & -2.06615720e{-}02 \\ -1.52757822e{-}03 & 2.36130832e{-}03 & \cdots & \cdots & -1.19824838e{-}02 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -8.25322699e{-}04 & -5.14897937e{-}03 & \cdots & \cdots & -9.90395527e{-}03 \end{matrix}$$

- **Even better:** Instead of using handcrafted kernels (such as edge detectors) **can we learn multiple meaningful kernels/filters in addition to learning the weights of the classifier?**

Input	Features	Classifier

car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

car, bus, monument, flower

$$\begin{matrix} -0.02337041 & -0.03243878 & \cdots & \cdots & -0.04728875 \\ -0.05375158 & -0.05350766 & \cdots & \cdots & -0.04323674 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -0.00792501 & -0.00503319 & \cdots & \cdots & 0.00174674 \end{matrix}$$

- **Even better:** Instead of using handcrafted kernels (such as edge detectors) **can we learn multiple meaningful kernels/filters in addition to learning the weights of the classifier?**

 91/113

Input | Features | Classifier

car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

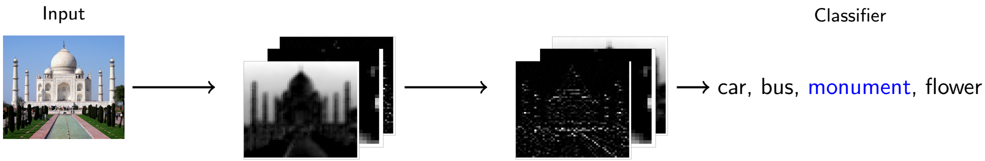
car, bus, monument, flower

$$\begin{matrix} -0.01871333 & -0.01075948 & \cdots & \cdots & 0.04684572 \\ 0.00104325 & 0.01935937 & \cdots & \cdots & 0.01016542 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ 0.03008777 & 0.00335217 & \cdots & \cdots & -0.02791128 \end{matrix}$$

- **Even better:** Instead of using handcrafted kernels (such as edge detectors)**can we learn** **multiple meaningful kernels/filters in addition to learning the weights of the classifier?**

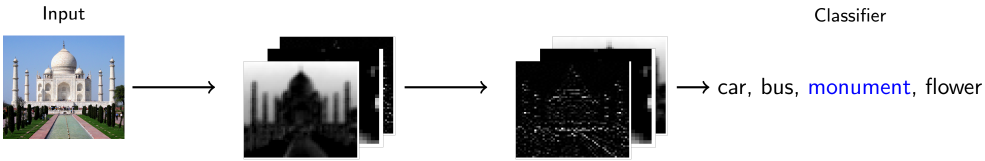
- Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?

92/113



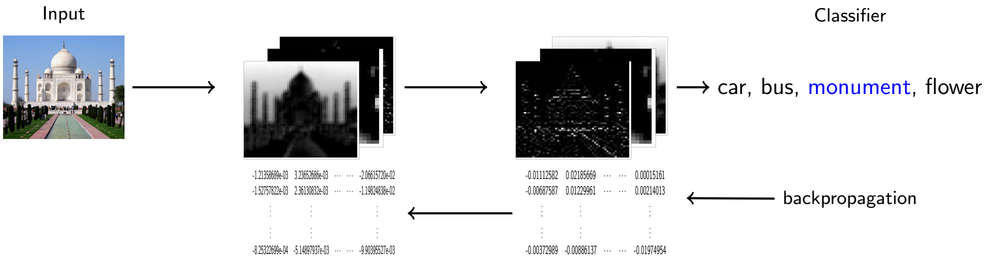
- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**
- Yes, we can !

 92/113

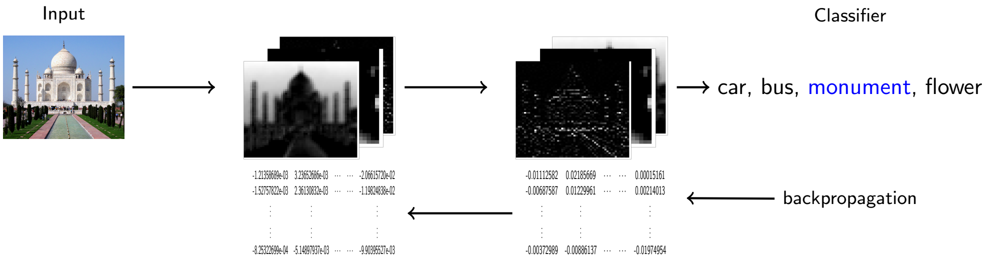


- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**
- Yes, we can !

 92/113



- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)



- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)
- Such a network is called a Convolutional Neural Network.

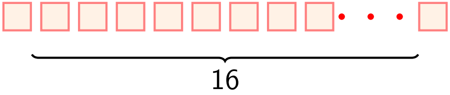
- Okay, I get it that the idea is to learn the kernel/filters by just treating them as parameters of the classification model

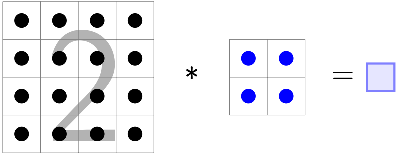
93/113

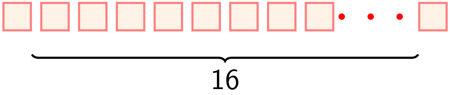
- Okay, I get it that the idea is to learn the kernel/filters by just treating them as parameters of the classification model
- But how is this different from a regular feedforward neural network

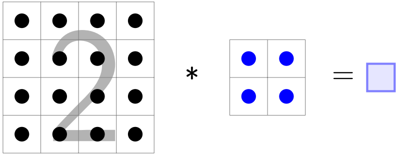
93/113

- Okay, I get it that the idea is to learn the kernel/filters by just treating them as parameters of the classification model
- But how is this different from a regular feedforward neural network
- Let us see

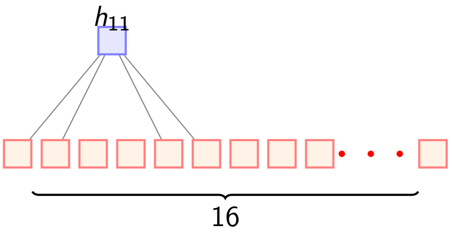

16

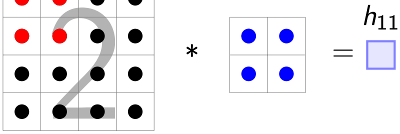

2



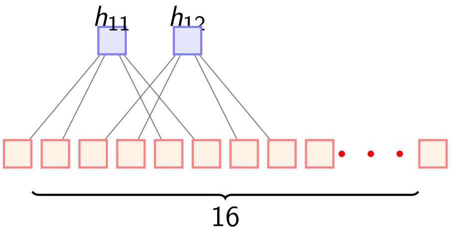


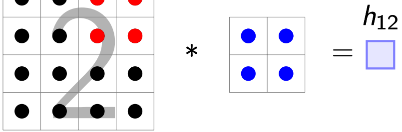
- Only a few local neurons participate in the computation of h_{11}



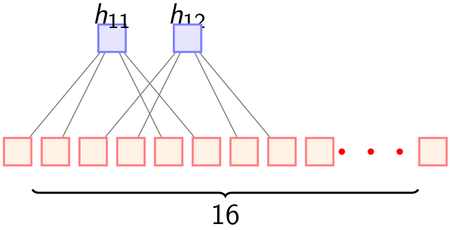


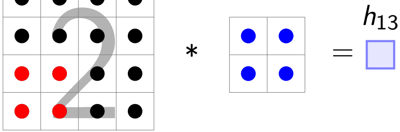
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}



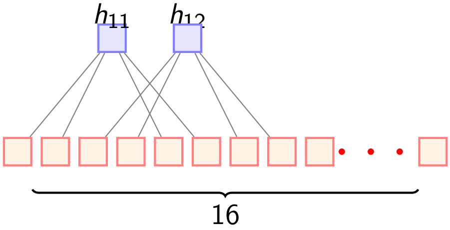


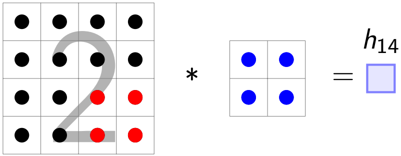
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}



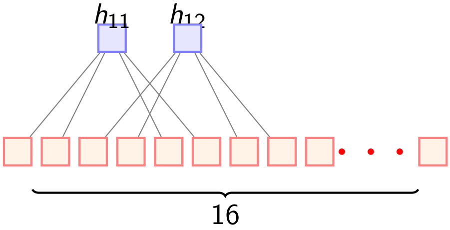


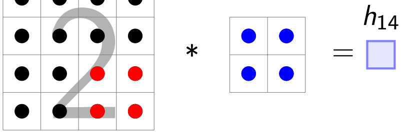
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}



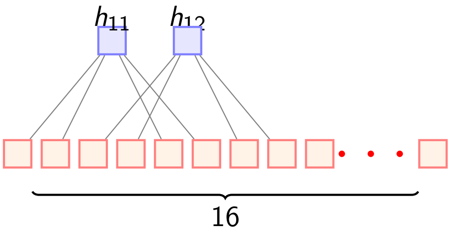


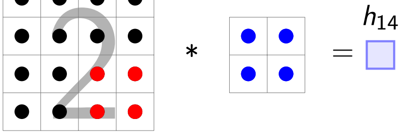
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}



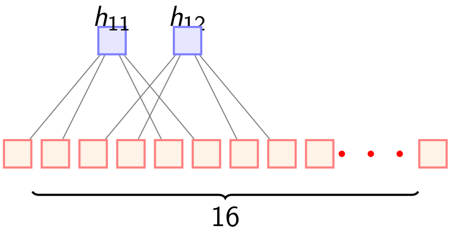


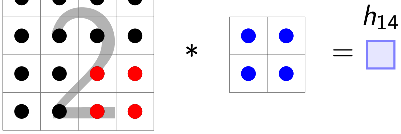
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser





- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser
- We are taking advantage of the structure of the image(interactions between neighboring pixels are more interesting)





- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser
- We are taking advantage of the structure of the image(interactions between neighboring pixels are more interesting)
- This **sparse connectivity** reduces the number of parameters in the model

- But is sparse connectivity really good thing ?

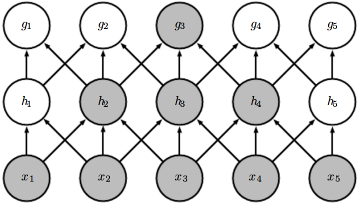
[a] **Goodfellow-et-al-2016**.

 95/113

- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)

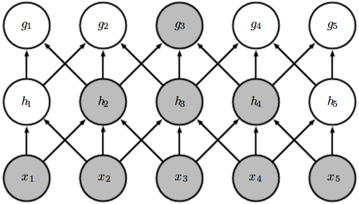
[a] **Goodfellow-et-al-2016**.

 95/113



- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really

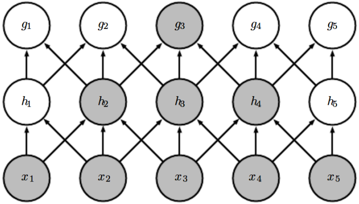
[a] **Goodfellow-et-al-2016**.



- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x_1 & x_5)[a] do not interact in *layer* 1

[a] **Goodfellow-et-al-2016**.

95/113



- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x_1 & x_5)[a] do not interact in *layer* 1
- But they indirectly contribute to the computation of g_3 and hence interact indirectly

[a] **Goodfellow-et-al-2016**.

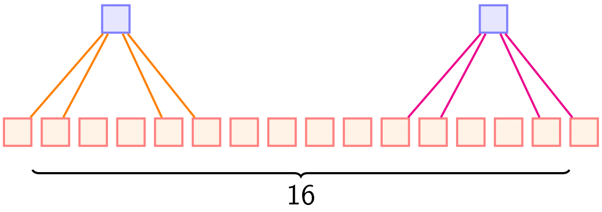
95/113

- Another characteristic of CNNs is **weight sharing**

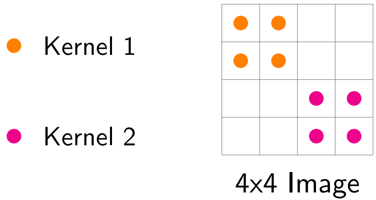
96/113

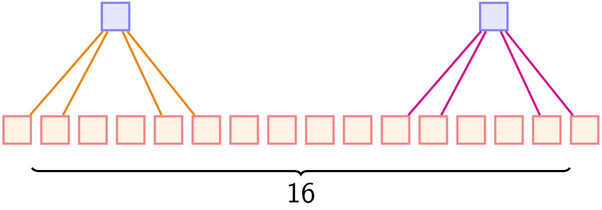
- Another characteristic of CNNs is **weight sharing**
- Consider the following network

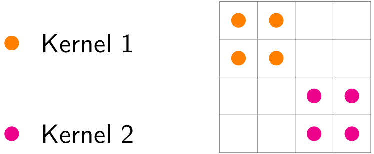
96/113



- Another characteristic of CNNs is **weight sharing**
- Consider the following network

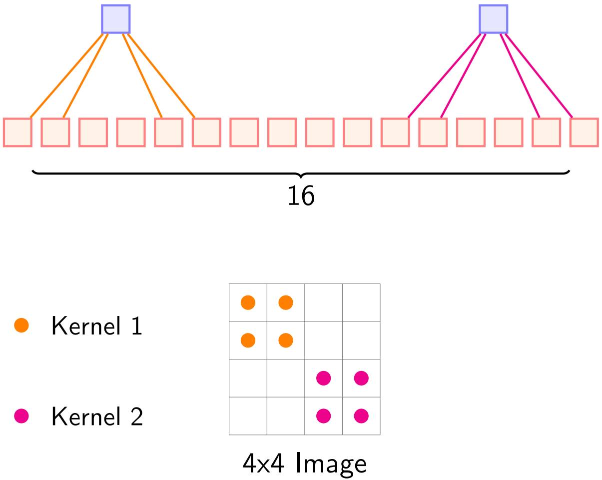




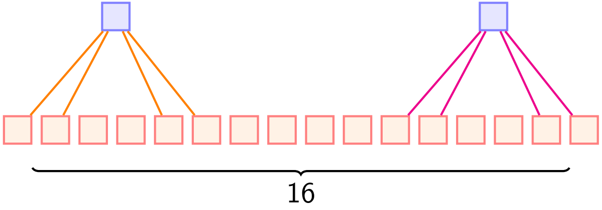


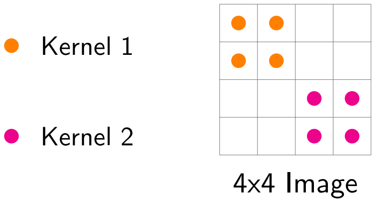
4x4 Image

- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image ?



- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image ?
- Imagine that we are trying to learn a kernel that detects edges

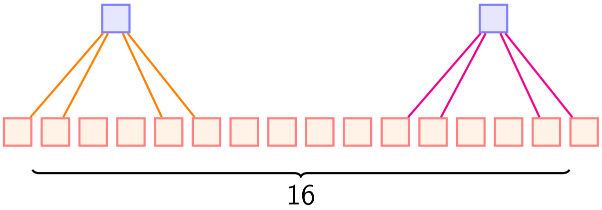


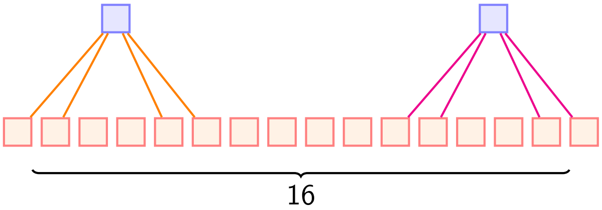


- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image ?
- Imagine that we are trying to learn a kernel that detects edges
- Shouldn't we be applying the same kernel at all the portions of the edge

 96/113

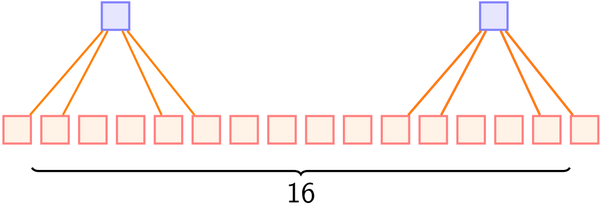
- In other words shouldn't the *orange* and *pink* kernels be the same

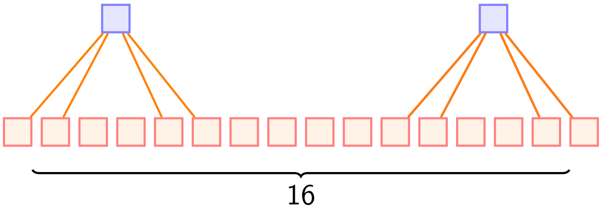




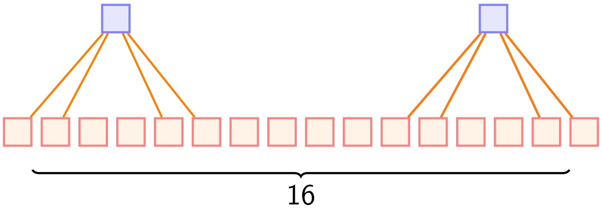
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed

- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed

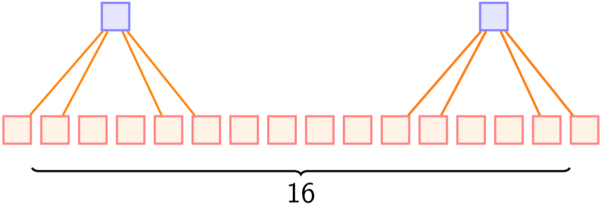




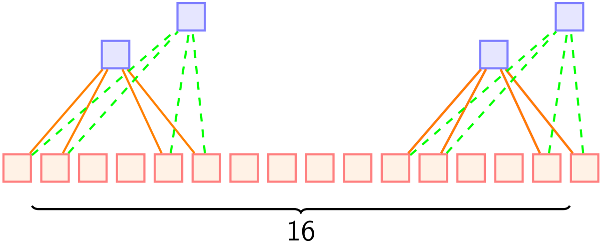
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)



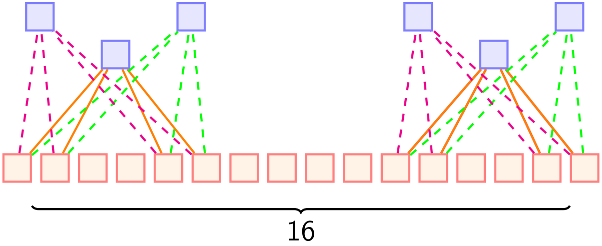
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?



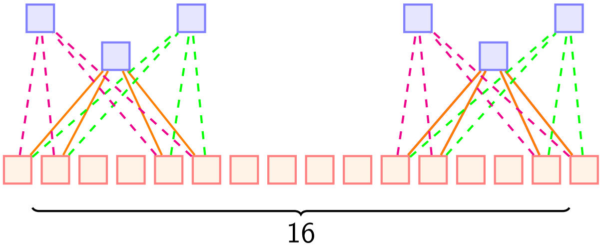
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image



- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image



- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image

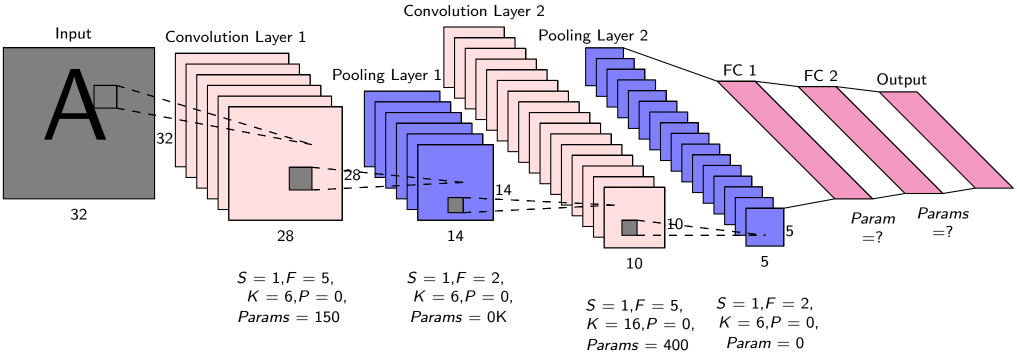


- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image
- This is called "weight sharing"

- So far, we have focused only on the convolution operation.

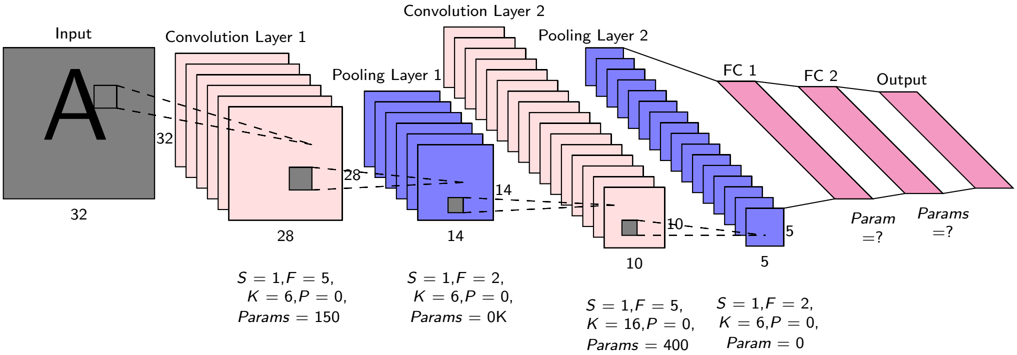
98/113

- So far, we have focused only on the convolution operation.
- Let us see what a full convolutional neural network looks like.

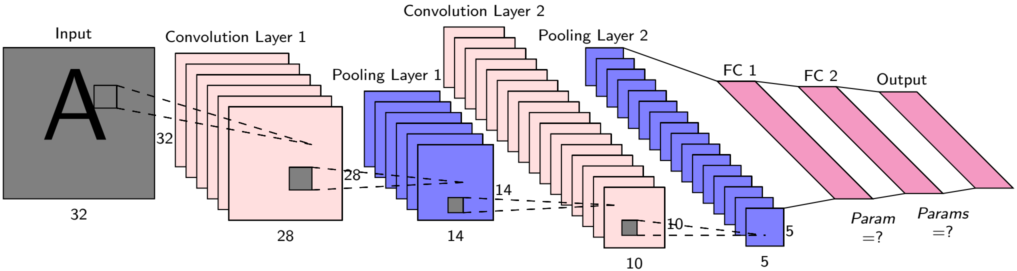


Input
32
32
A
Convolution Layer 1
28
28
Pooling Layer 1
14
14
Convolution Layer 2
10
10
Pooling Layer 2
5
5
FC 1
FC 2
Output
S = 1, F = 5, K = 6, P = 0, Params = 150
S = 1, F = 2, K = 6, P = 0, Params = 0K
S = 1, F = 5, K = 16, P = 0, Params = 400
S = 1, F = 2, K = 6, P = 0, Param = 0
Param =?
Params =?

99/113


Input
A
32
32
Convolution Layer 1
28
28
Pooling Layer 1
14
14
Convolution Layer 2
10
10
Pooling Layer 2
5
5
FC 1
FC 2
Output
$S = 1, F = 5,$
$K = 6, P = 0,$
$Params = 150$
$S = 1, F = 2,$
$K = 6, P = 0,$
$Params = 0$K
$S = 1, F = 5,$
$K = 16, P = 0,$
$Params = 400$
$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$
$Param$
$=?$
$Params$
$=?$

99/113

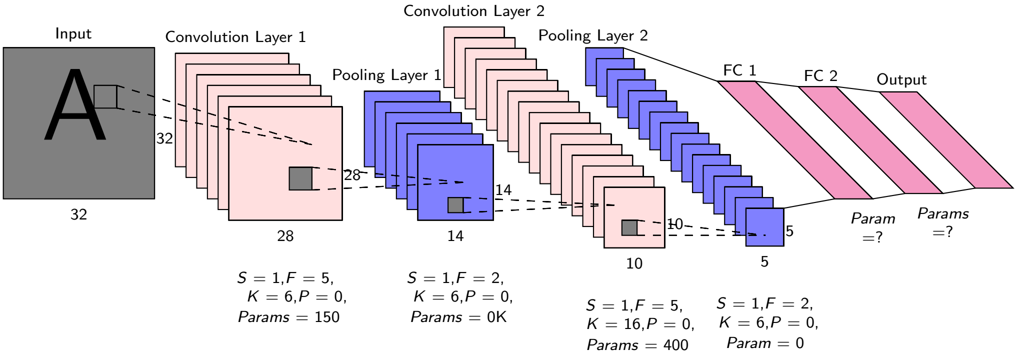


$S = 1, F = 5,$
$K = 6, P = 0,$
$Params = 150$

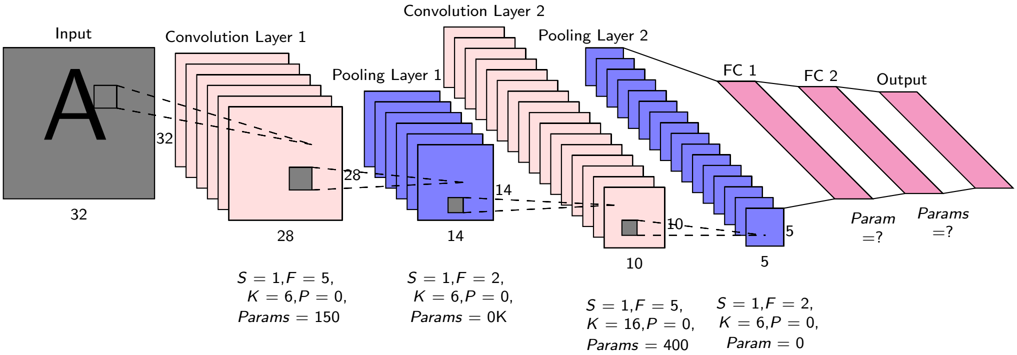
$S = 1, F = 2,$
$K = 6, P = 0,$
$Params = 0K$

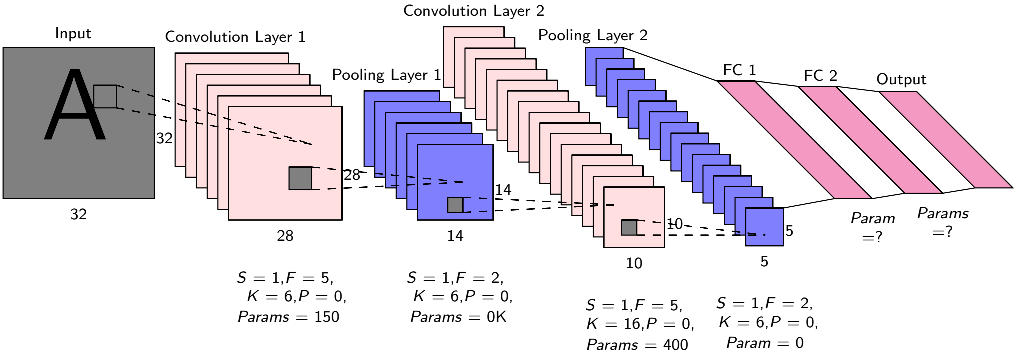
$S = 1, F = 5,$
$K = 16, P = 0,$
$Params = 400$

$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$

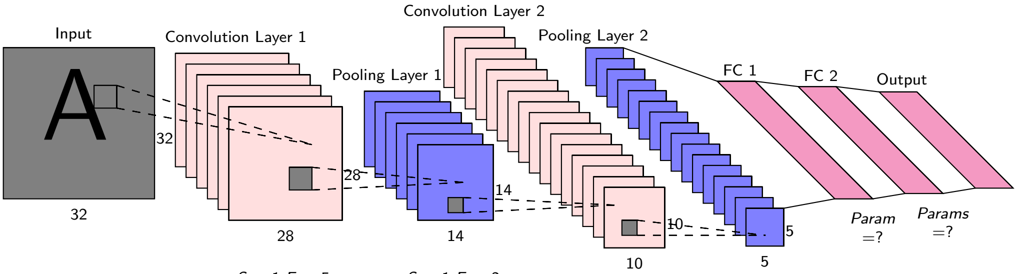


Input
Convolution Layer 1
Convolution Layer 2
Pooling Layer 1
Pooling Layer 2
FC 1
FC 2
Output
32
32
28
28
14
14
10
10
5
5
Param =?
Params =?
S = 1, F = 5, K = 6, P = 0, Params = 150
S = 1, F = 2, K = 6, P = 0, Params = 0K
S = 1, F = 5, K = 16, P = 0, Params = 400
S = 1, F = 2, K = 6, P = 0, Param = 0


Input
A
32
32
Convolution Layer 1
28
28
Pooling Layer 1
14
14
Convolution Layer 2
10
10
Pooling Layer 2
5
5
FC 1
FC 2
Output
$S = 1, F = 5,$
$K = 6, P = 0,$
$Params = 150$
$S = 1, F = 2,$
$K = 6, P = 0,$
$Params = 0$K
$S = 1, F = 5,$
$K = 16, P = 0,$
$Params = 400$
$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$
$Param$
$=?$
$Params$
$=?$


Input
32
32
Convolution Layer 1
28
28
Pooling Layer 1
14
14
Convolution Layer 2
10
10
Pooling Layer 2
5
5
FC 1
FC 2
Output
Param =?
Params =?
$S = 1, F = 5,$
$K = 6, P = 0,$
$Params = 150$
$S = 1, F = 2,$
$K = 6, P = 0,$
$Params = 0K$
$S = 1, F = 5,$
$K = 16, P = 0,$
$Params = 400$
$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$

 99/113

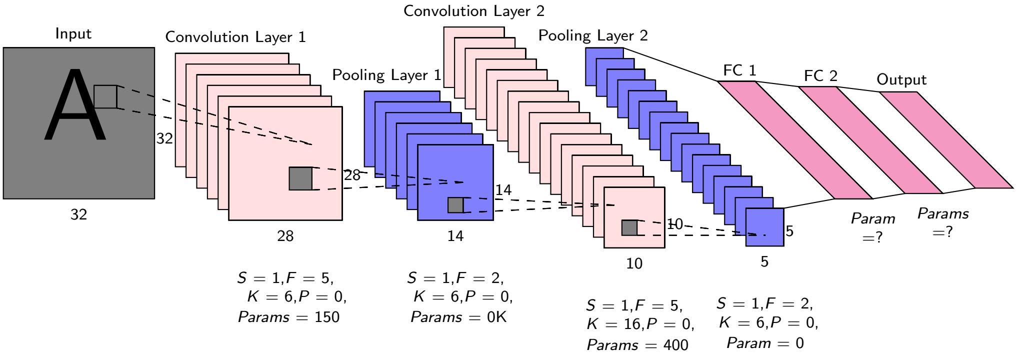


$S = 1, F = 5,$
$K = 6, P = 0,$
$Params = 150$

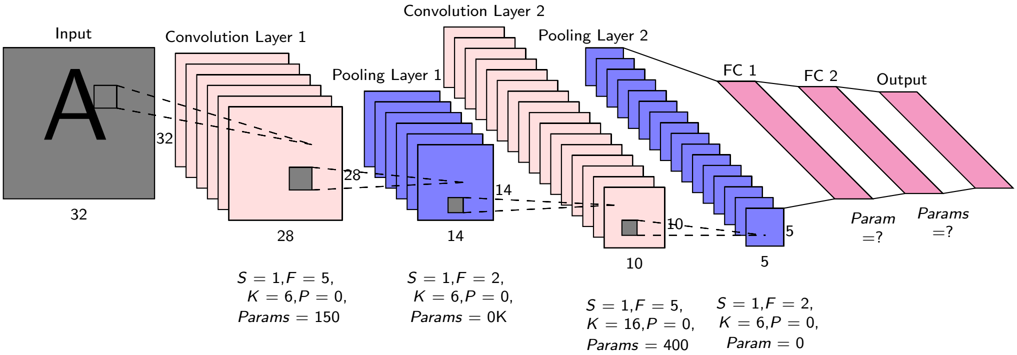
$S = 1, F = 2,$
$K = 6, P = 0,$
$Params = 0K$

$S = 1, F = 5,$
$K = 16, P = 0,$
$Params = 400$

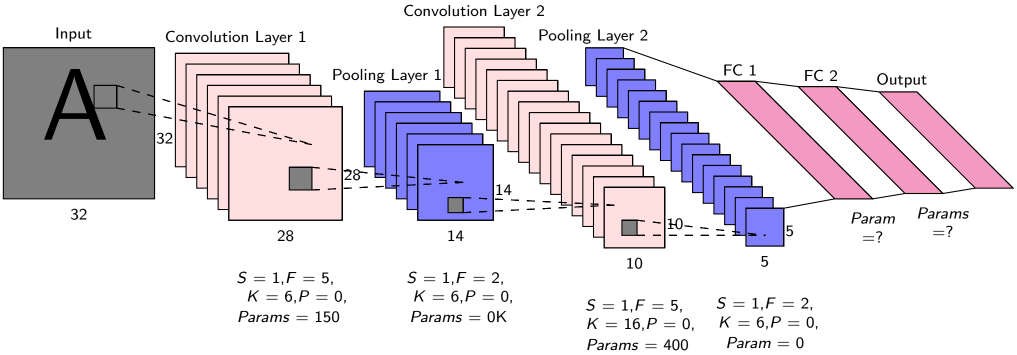
$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$



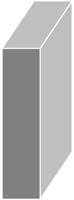
- It has alternate convolution and pooling layers



- It has alternate convolution and pooling layers
- What does a pooling layer do?

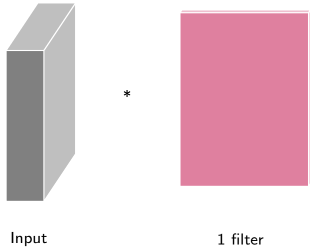


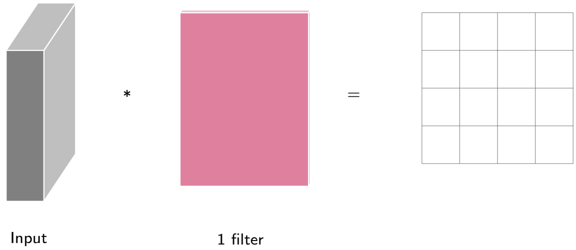
- It has alternate convolution and pooling layers
- What does a pooling layer do?
- Let us see



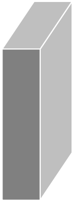
Input

100/113


*
Input
1 filter

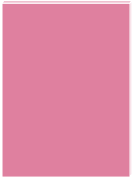


Input
1 filter



Input

*

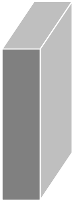


1 filter

=

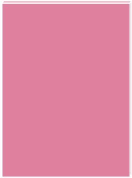
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

 100/113



Input

$*$



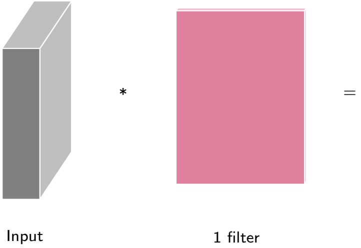
1 filter

$=$

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

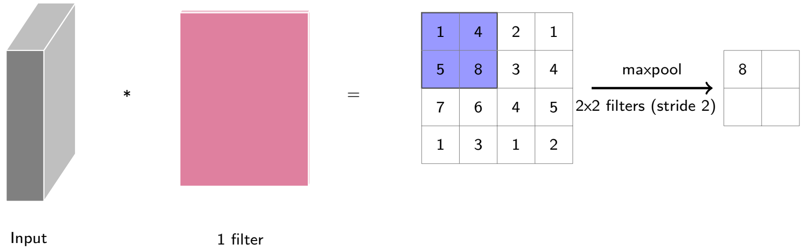
100/113

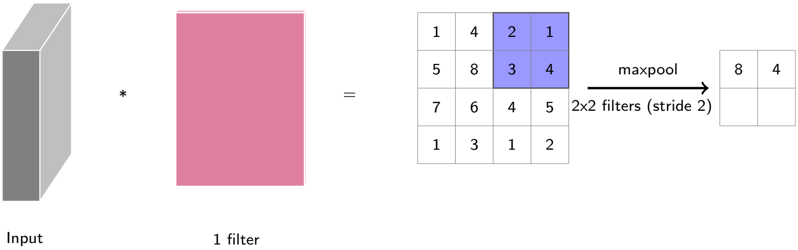


1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

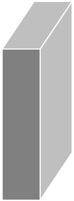
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

100/113


Input
*
1 filter
=
1 4 2 1
5 8 3 4
7 6 4 5
1 3 1 2
maxpool
2x2 filters (stride 2)
8

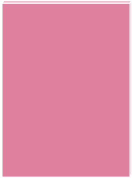

Input
*
1 filter
=
1 4 2 1
5 8 3 4
7 6 4 5
1 3 1 2
maxpool
2x2 filters (stride 2)
8 4

 100/113



Input

*



1 filter

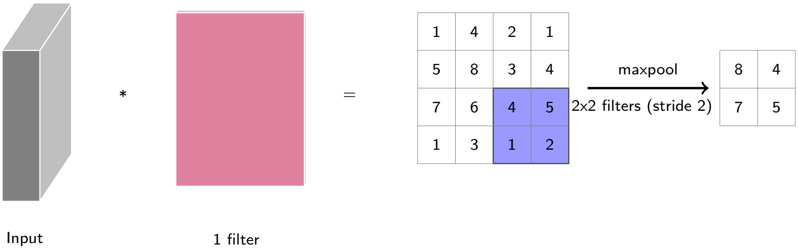
=

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

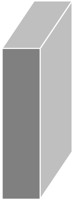
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	

100/113

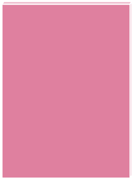

*
=
1 4 2 1
5 8 3 4
7 6 4 5
1 3 1 2
maxpool
2x2 filters (stride 2)
8 4
7 5
Input
1 filter

100/113



Input

*



1 filter

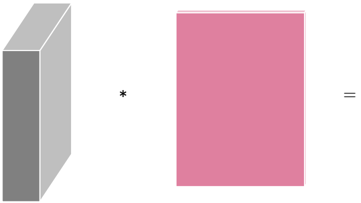
=

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2



Input

1 filter

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

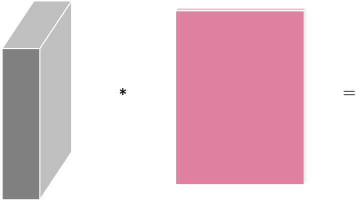
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

100/113



Input

1 filter

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

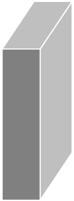
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

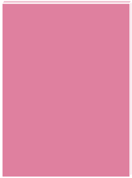
$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

 100/113



Input

*



1 filter

=

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

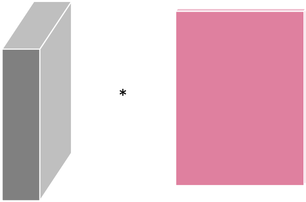
8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8		

 100/113



Input * 1 filter =

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

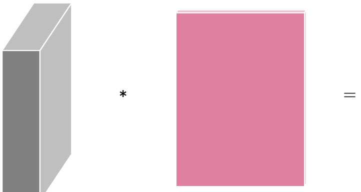
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	



Input

1 filter

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

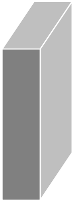
8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

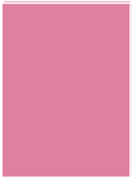
8	8	4

100/113



Input

*



1 filter

=

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

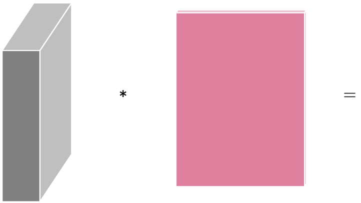
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8		



Input

1 filter

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

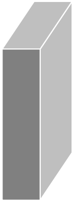
8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

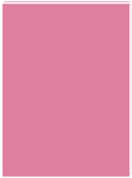
8	8	4
8	8	

100/113



Input

$*$



1 filter

$=$

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

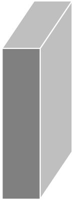
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

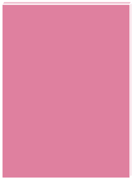
$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	5



Input

*



1 filter

=

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

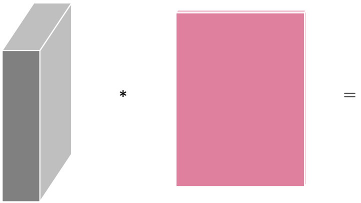
8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	5
7		

100/113



Input

1 filter

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

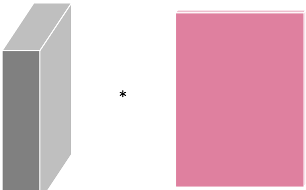
8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	5
7	6	

100/113



Input   *   1 filter   =

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

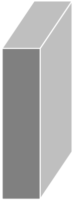
8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

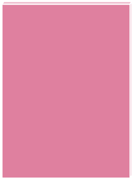
8	8	4
8	8	5
7	6	5

100/113



Input

*



1 filter

=

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

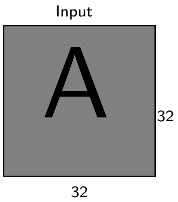
8	8	4
8	8	5
7	6	5

- Instead of max pooling we can also do average pooling

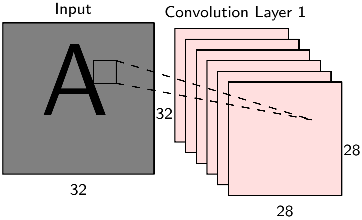
 100/113

We will now see some case studies where convolution neural networks have been successful

LeNet-5 for handwritten character recognition



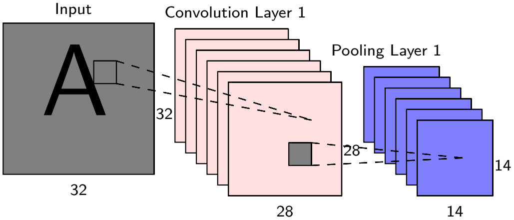
LeNet-5 for handwritten character recognition



$S = 1, F = 5,$
$K = 6, P = 0,$
$Params = 150$

102/113

LeNet-5 for handwritten character recognition

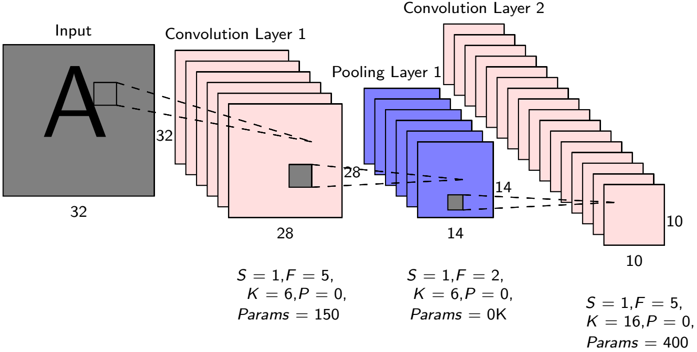


$S = 1, F = 5,$
$K = 6, P = 0,$
$Params = 150$

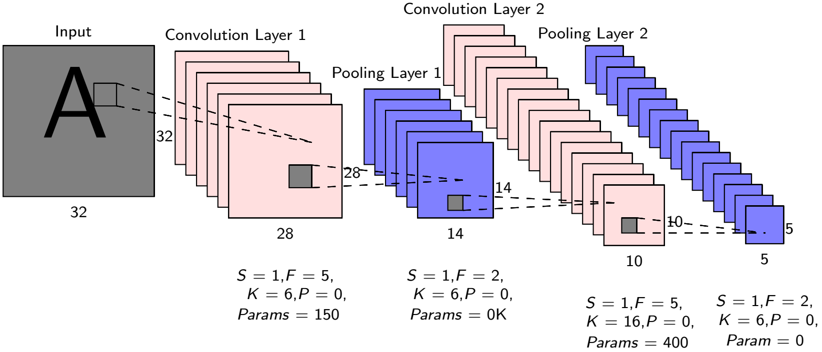
$S = 1, F = 2,$
$K = 6, P = 0,$
$Params = 0K$

102/113

LeNet-5 for handwritten character recognition

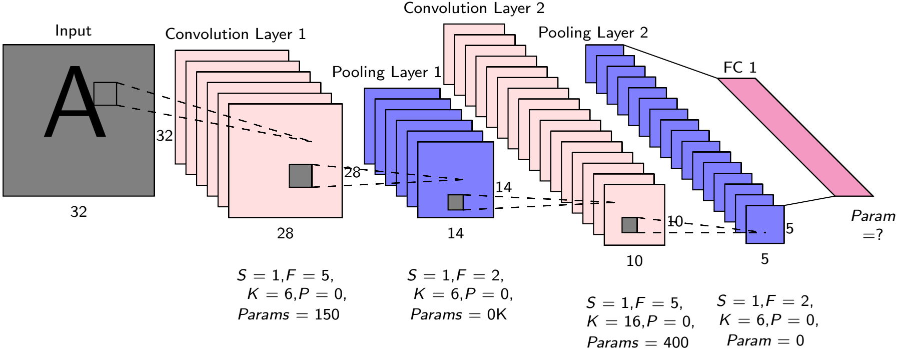


LeNet-5 for handwritten character recognition

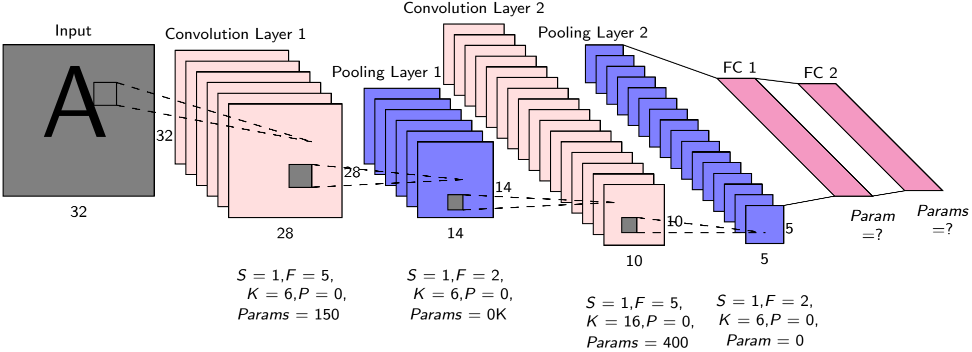


 102/113

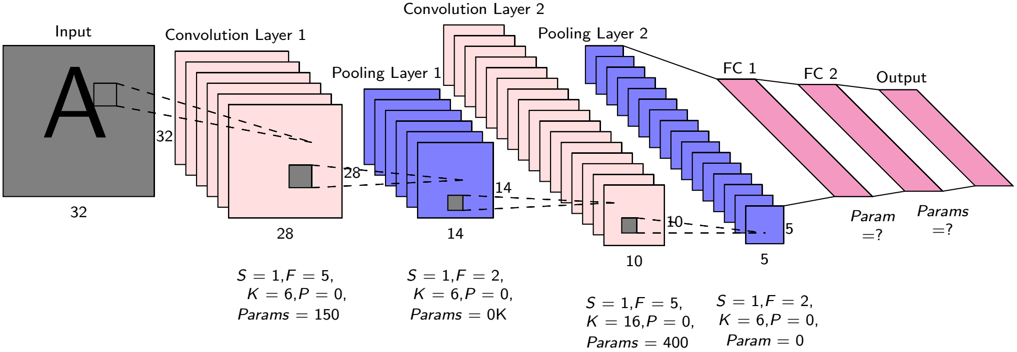
LeNet-5 for handwritten character recognition



LeNet-5 for handwritten character recognition



LeNet-5 for handwritten character recognition



102/113

ImageNet Success Stories(roadmap for rest of the talk)

- AlexNet

103/113

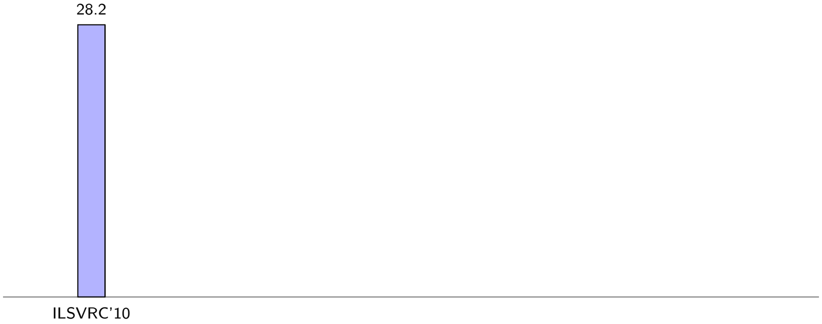
ImageNet Success Stories(roadmap for rest of the talk)

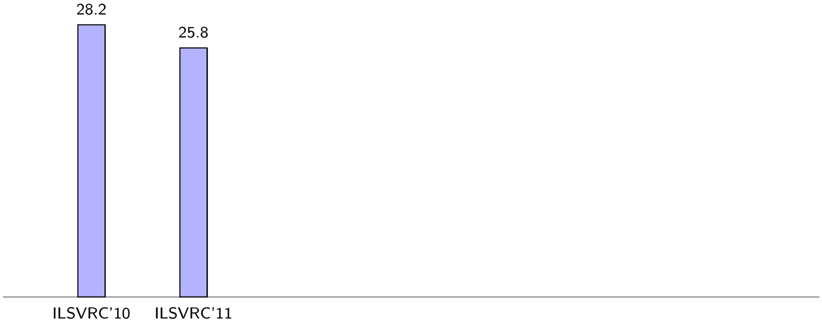
- AlexNet
- ZFNet

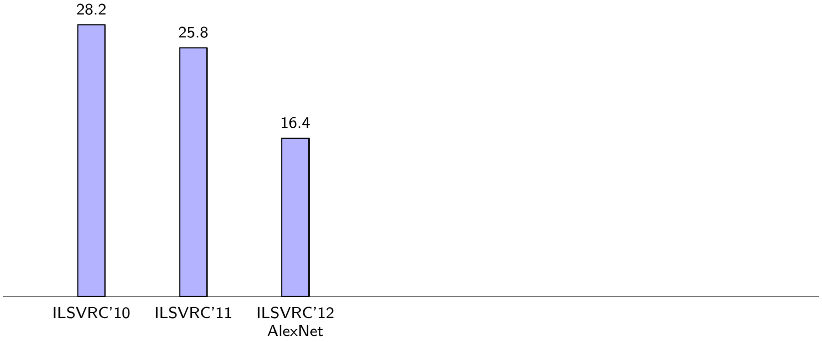
103/113

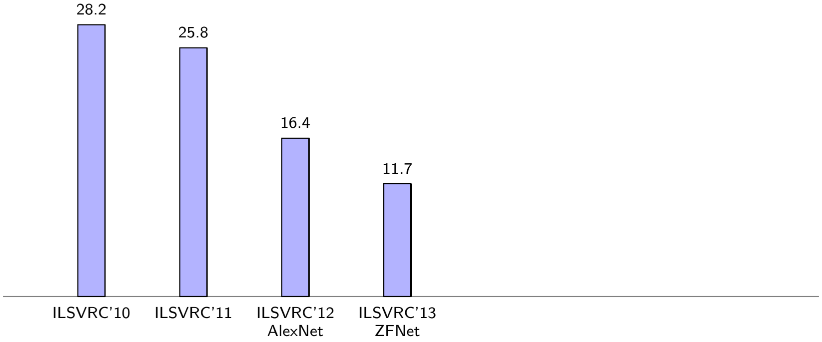
ImageNet Success Stories(roadmap for rest of the talk)

- AlexNet
- ZFNet
- VGGNet

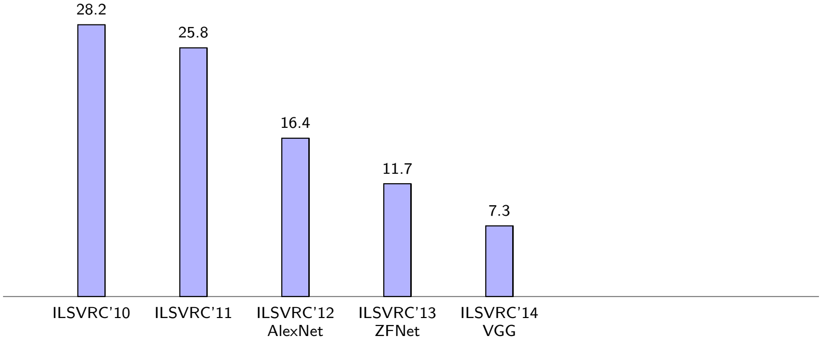

28.2
ILSVRC'10


28.2
25.8
ILSVRC'10
ILSVRC'11

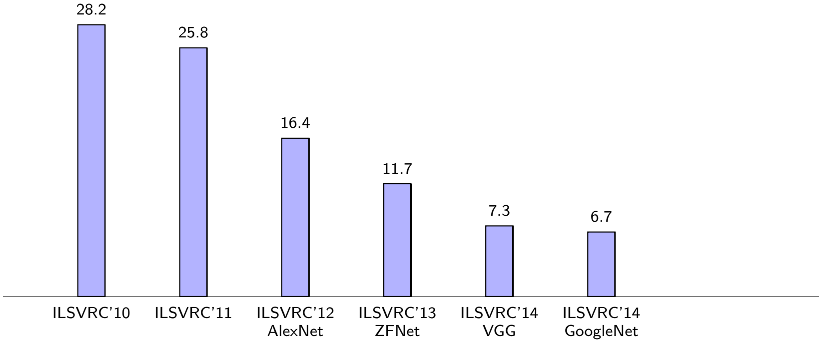

28.2
25.8
16.4
ILSVRC'10
ILSVRC'11
ILSVRC'12
AlexNet

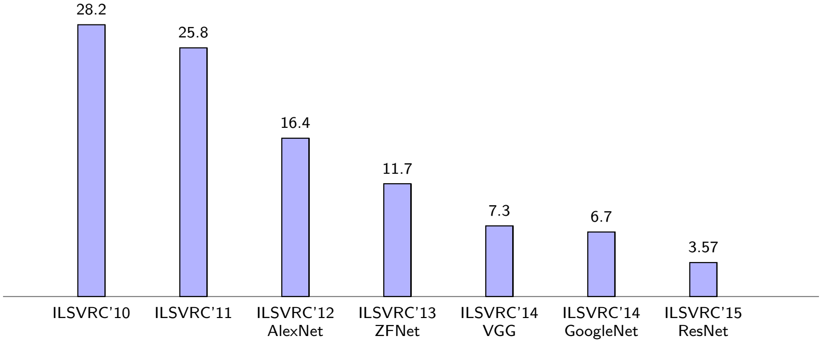

28.2
25.8
16.4
11.7
ILSVRC'10
ILSVRC'11
ILSVRC'12
AlexNet
ILSVRC'13
ZFNet

104/113

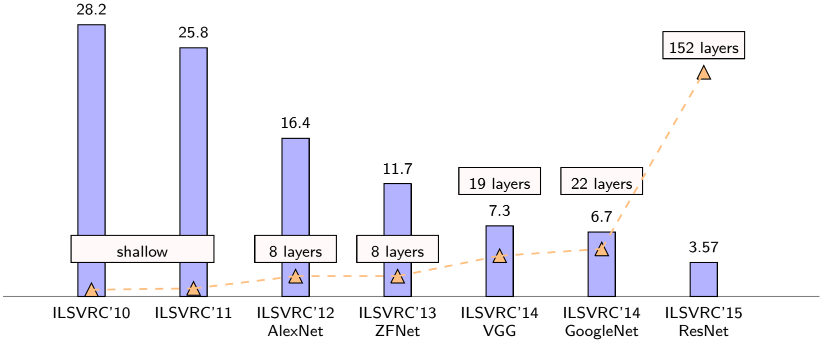

28.2
25.8
16.4
11.7
7.3
ILSVRC'10
ILSVRC'11
ILSVRC'12
AlexNet
ILSVRC'13
ZFNet
ILSVRC'14
VGG

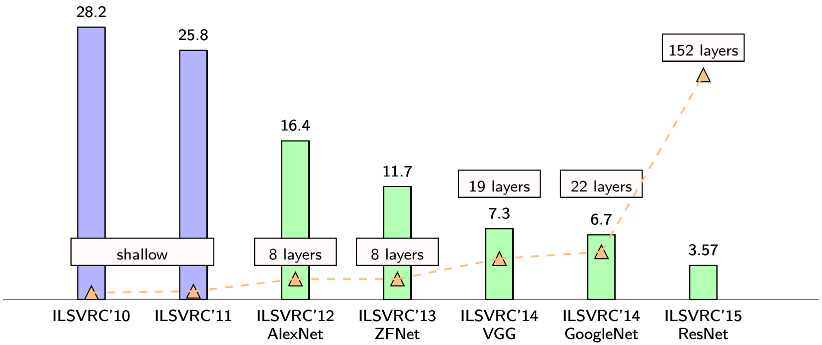
 104/113


28.2
25.8
16.4
11.7
7.3
6.7
ILSVRC'10
ILSVRC'11
ILSVRC'12 AlexNet
ILSVRC'13 ZFNet
ILSVRC'14 VGG
ILSVRC'14 GoogleNet


28.2
25.8
16.4
11.7
7.3
6.7
3.57
ILSVRC'10
ILSVRC'11
ILSVRC'12 AlexNet
ILSVRC'13 ZFNet
ILSVRC'14 VGG
ILSVRC'14 GoogleNet
ILSVRC'15 ResNet

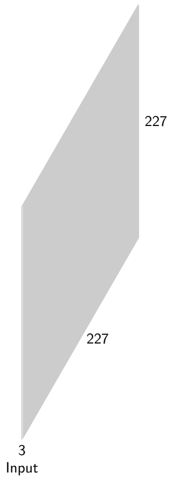
 104/113

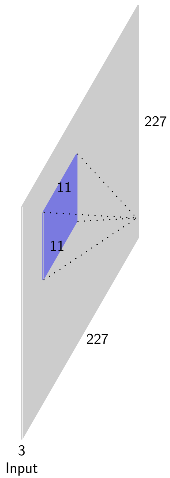

28.2
25.8
16.4
11.7
7.3
6.7
3.57
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
ILSVRC'10
ILSVRC'11
ILSVRC'12 AlexNet
ILSVRC'13 ZFNet
ILSVRC'14 VGG
ILSVRC'14 GoogleNet
ILSVRC'15 ResNet


28.2
25.8
16.4
11.7
7.3
6.7
3.57
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
ILSVRC'10
ILSVRC'11
ILSVRC'12 AlexNet
ILSVRC'13 ZFNet
ILSVRC'14 VGG
ILSVRC'14 GoogleNet
ILSVRC'15 ResNet

ImageNet Success Stories(roadmap for rest of the talk)

- AlexNet
- ZFNet
- VGGNet


227
227
3
Input



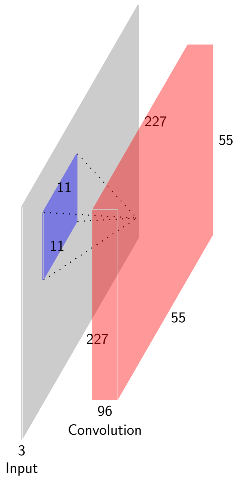
Input: $227 \times 227 \times 3$

Conv1: $K = 96, F = 11$

$S = 4, P = 0$

Output: $W_2 = ?, \ H_2 = ?$

Parameters: ?



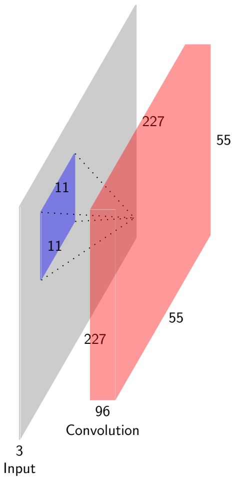
Input: $227 \times 227 \times 3$

Conv1: $K = 96, F = 11$

$S = 4, P = 0$

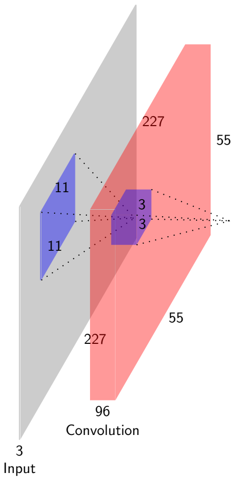
Output: $W_2 = 55,\ H_2 = 55$

Parameters: ?



Input: $227 \times 227 \times 3$
Conv1: $K = 96, F = 11$
$S = 4, P = 0$
Output: $W_2 = 55,\ H_2 = 55$
Parameters: $(11 \times 11 \times 3) \times 96 = 34K$

 106/113

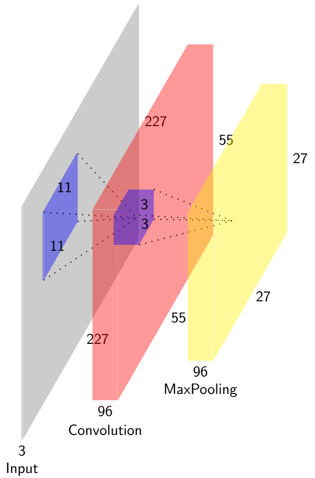


Max Pool Input: $55 \times 55 \times 96$

$F = 3, S = 2$

Output: $W_2 = ?,\ H_2 = ?$

Parameters: ?



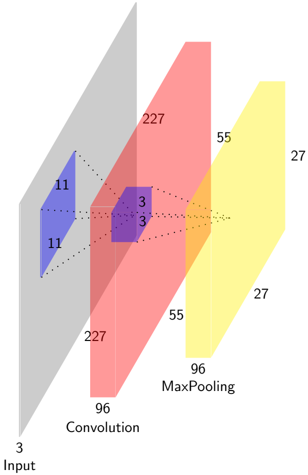
Max Pool Input: $55 \times 55 \times 96$

$F = 3, S = 2$

Output: $W_2 = 27,\ H_2 = 27$

Parameters: ?

106/113

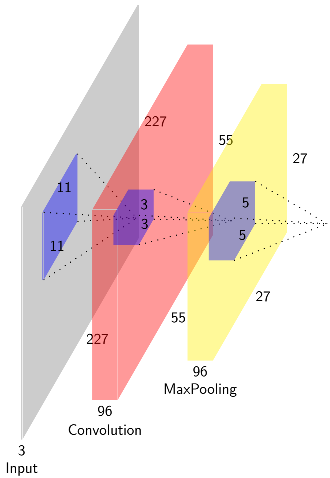


Max Pool Input: $55 \times 55 \times 96$

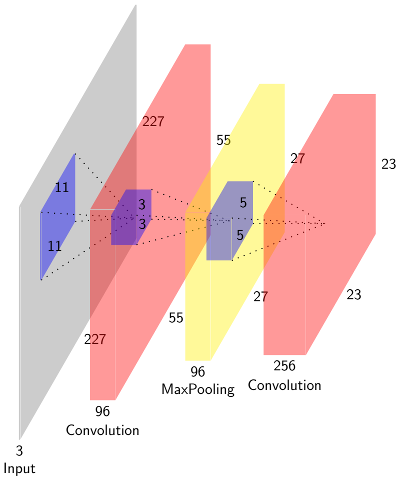
$F = 3, S = 2$

Output: $W_2 = 27,\ H_2 = 27$

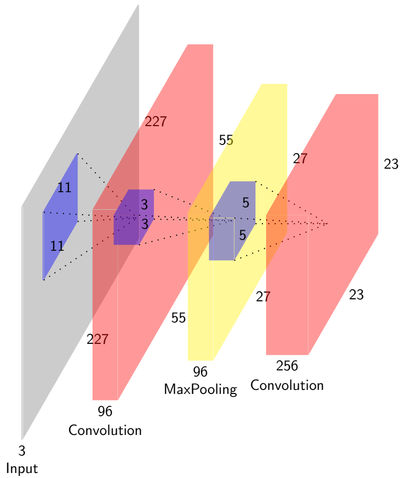
Parameters: 0



Input: $27 \times 27 \times 96$
Conv1: $K = 256, F = 5$
$S = 1, P = 0$
Output: $W_2 = ?,\ H_2 = ?$
Parameters: ?

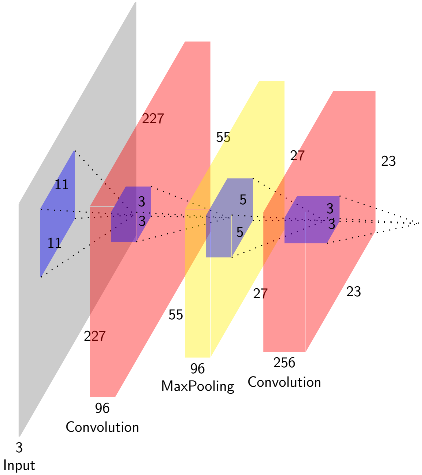


Input: $27 \times 27 \times 96$
Conv1: $K = 256, F = 5$
$S = 1, P = 0$
Output: $W_2 = 23,\ H_2 = 23$
Parameters: ?



Input: $27 \times 27 \times 96$
Conv1: $K = 256, F = 5$
$S = 1, P = 0$
Output: $W_2 = 23,\ H_2 = 23$
Parameters: $(5 \times 5 \times 96) \times 256 = 0.6M$

106/113

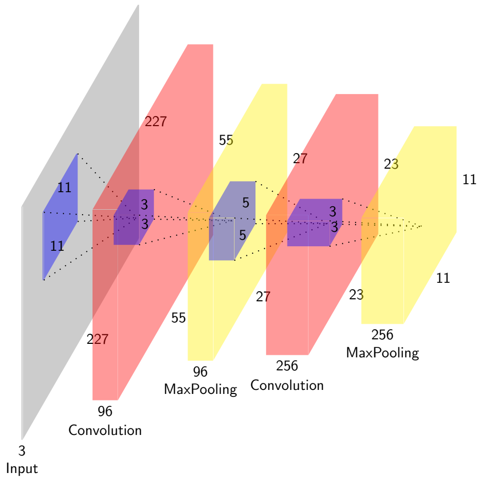


Max Pool Input: $23 \times 23 \times 256$

$F = 3, S = 2$

Output: $W_2 = ?,\ H_2 = ?$

Parameters: ?

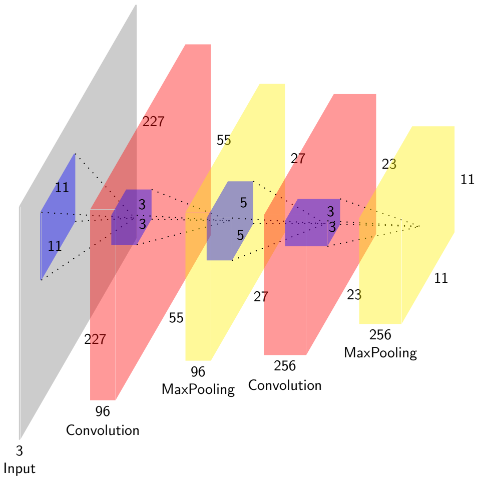


Max Pool Input: $23 \times 23 \times 256$

$F = 3, S = 2$

Output: $W_2 = 11,\ H_2 = 11$

Parameters: ?

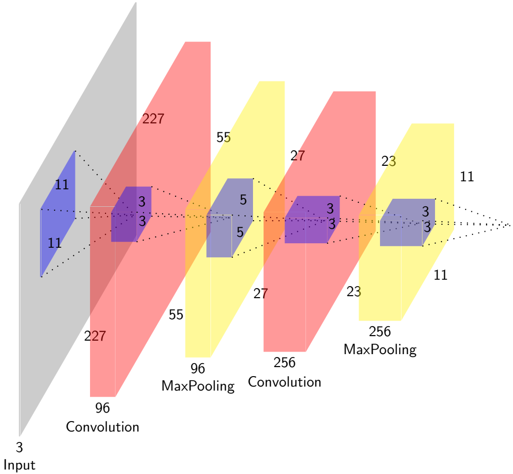


Max Pool Input: $23 \times 23 \times 256$

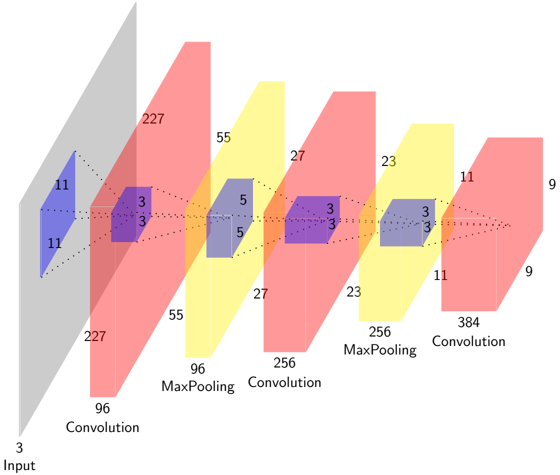
$F = 3, S = 2$

Output: $W_2 = 11,\ H_2 = 11$

Parameters: 0



Input: $11 \times 11 \times 256$
Conv1: $K = 384, F = 3$
$S = 1, P = 0$
Output: $W_2 = ?,\ H_2 = ?$
Parameters: ?



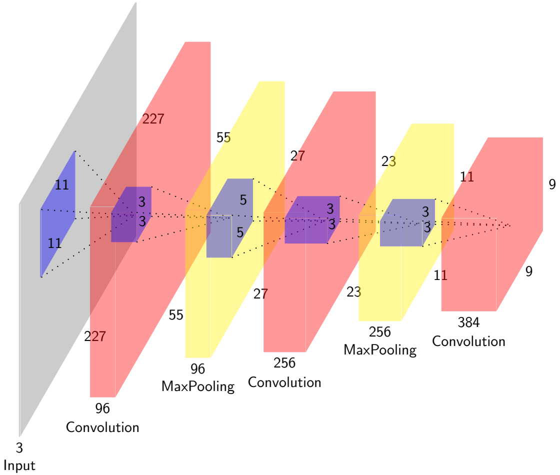
Input: $11 \times 11 \times 256$

Conv1: $K = 384, F = 3$

$S = 1, P = 0$

Output: $W_2 = 9,\ H_2 = 9$

Parameters: ?



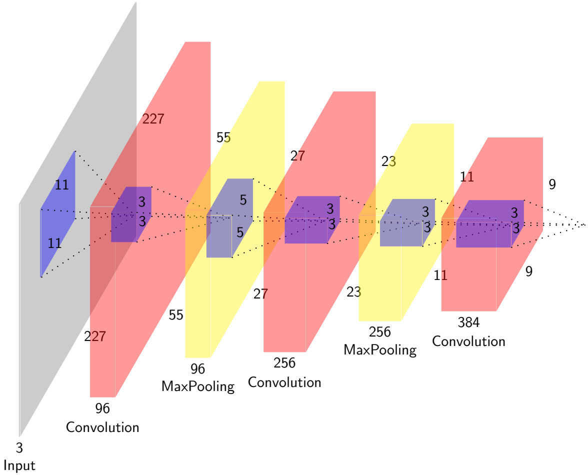
Input: $11 \times 11 \times 256$

Conv1: $K = 384, F = 3$

$S = 1, P = 0$

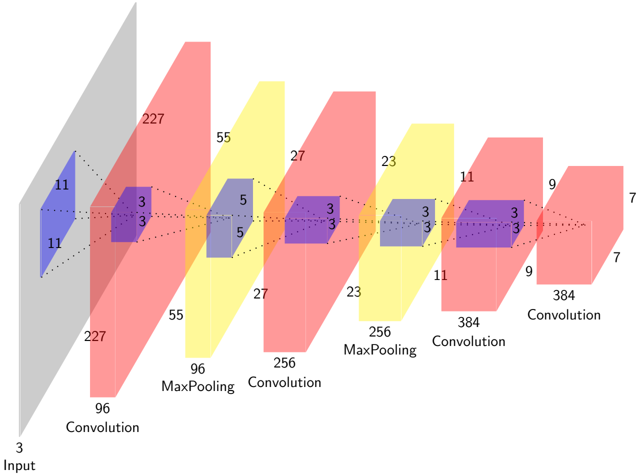
Output: $W_2 = 9, \ H_2 = 9$

Parameters: $(3 \times 3 \times 256) \times 384 = 0.8M$



Input: $9 \times 9 \times 384$
Conv1: $K = 384, F = 3$
$S = 1, P = 0$
Output: $W_2 = ?,\ H_2 = ?$
Parameters: ?

106/113



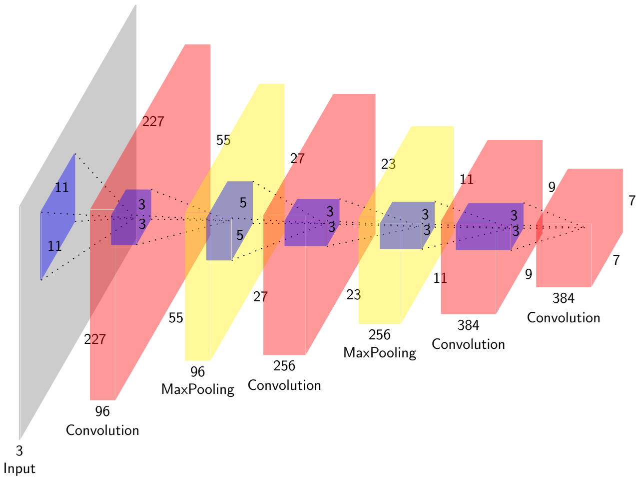
Input: $9 \times 9 \times 384$

Conv1: $K = 384, F = 3$

$S = 1, P = 0$

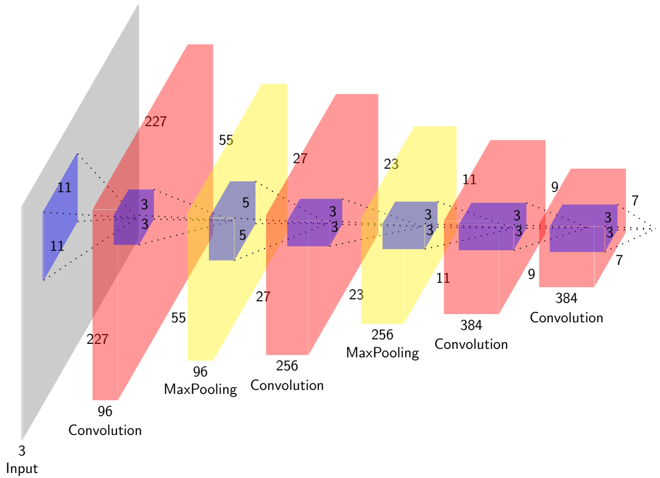
Output: $W_2 = 7, \ H_2 = 7$

Parameters: ?



Input: $9 \times 9 \times 384$
Conv1: $K = 384, F = 3$
$S = 1, P = 0$
Output: $W_2 = 7,\ H_2 = 7$
Parameters: $(3 \times 3 \times 384) \times 384 = 1.327M$

106/113



Input: $7 \times 7 \times 384$

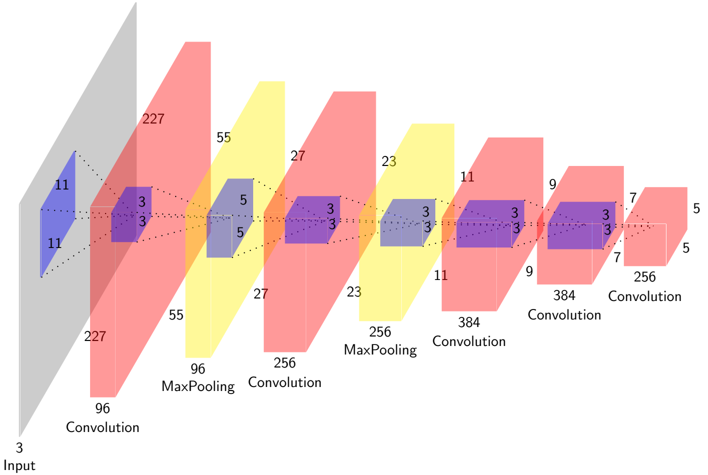
Conv1: $K = 256, F = 3$

$S = 1, P = 0$

Output: $W_2 = ?,\ H_2 = ?$

Parameters: ?

 106/113



Input: $7 \times 7 \times 384$

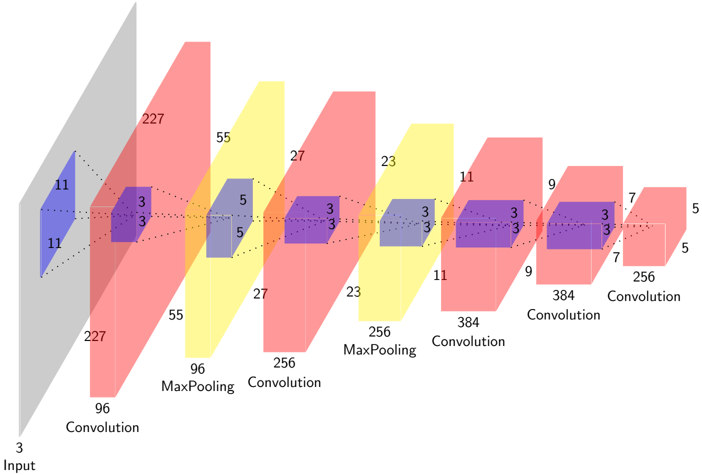
Conv1: $K = 256, F = 3$

$S = 1, P = 0$

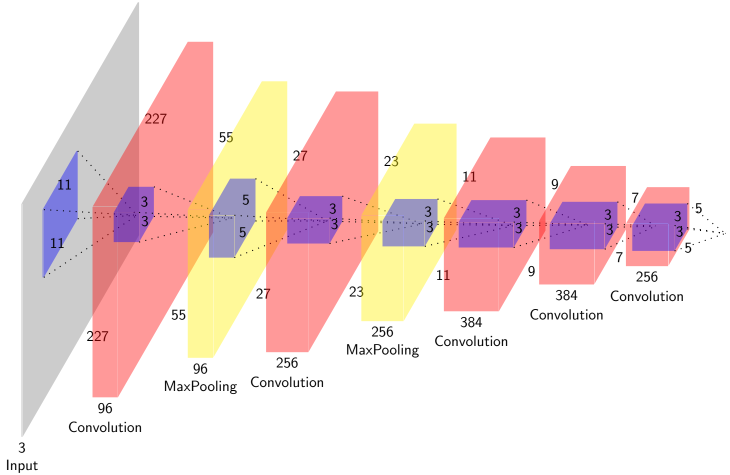
Output: $W_2 = 5,\ H_2 = 5$

Parameters: ?

106/113



Input: $7 \times 7 \times 384$
Conv1: $K = 256, F = 3$
$S = 1, P = 0$
Output: $W_2 = 5, \ H_2 = 5$
Parameters: $(3 \times 3 \times 384) \times 256 = 0.8M$



Max Pool Input: $5 \times 5 \times 256$

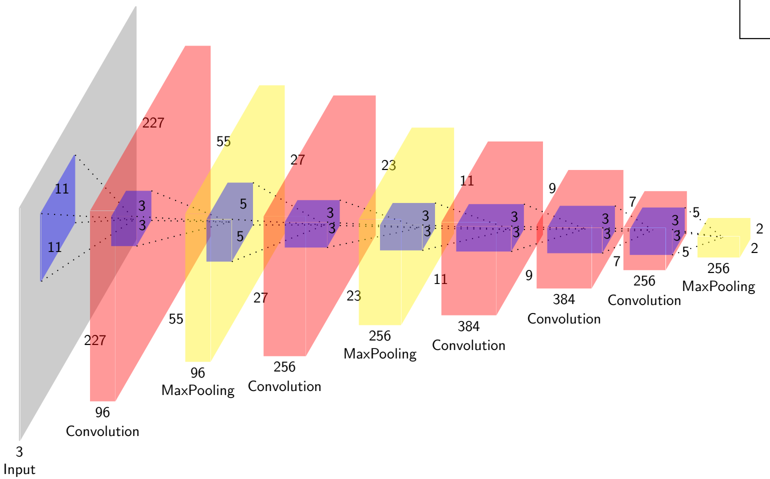
$F = 3, S = 2$

Output: $W_2 = ?$, $H_2 = ?$

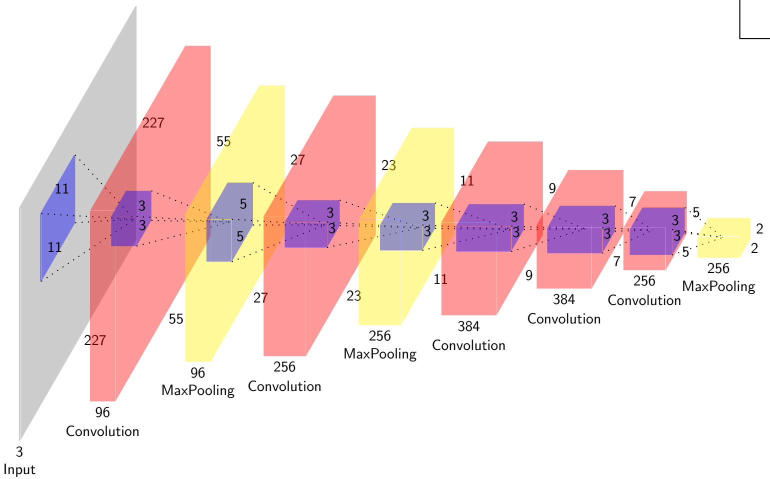
Parameters: ?

106/113

Max Pool Input: $5 \times 5 \times 256$
$F = 3, S = 2$
Output: $W_2 = 2,\ H_2 = 2$
Parameters: ?



106/113



Max Pool Input: $5 \times 5 \times 256$

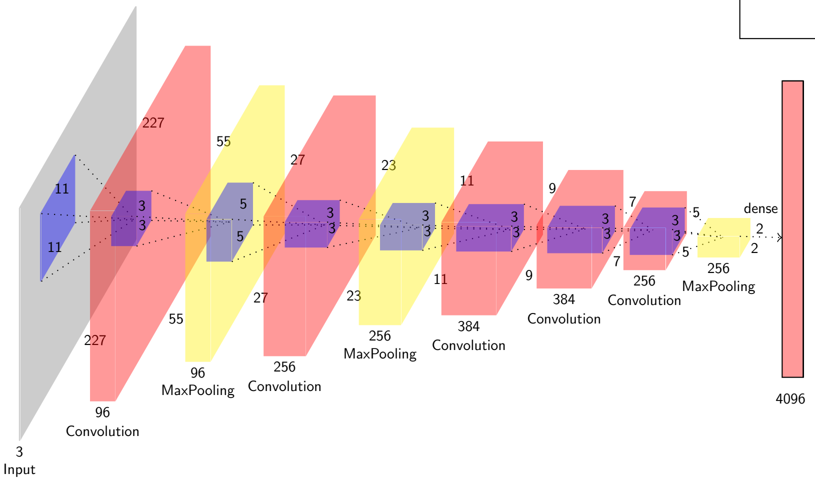
$F = 3, S = 2$

Output: $W_2 = 2,\ H_2 = 2$

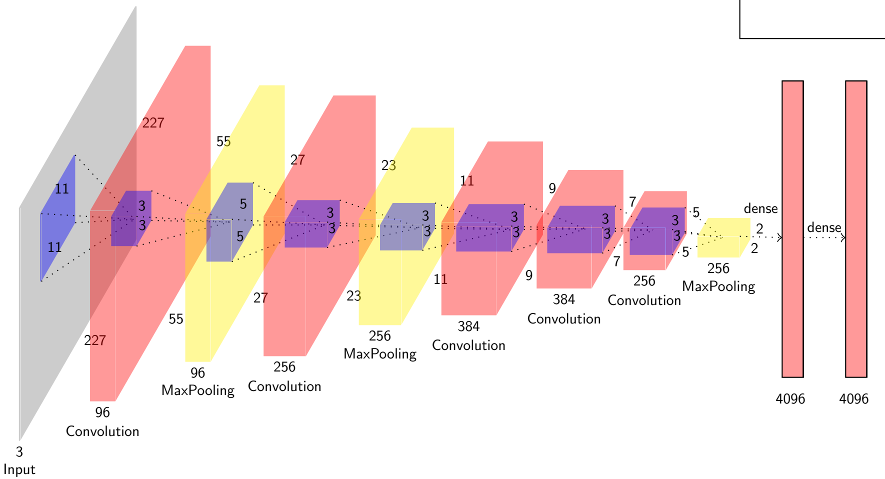
Parameters: 0

FC1

Parameters: $(2 \times 2 \times 256) \times 4096 = 4M$



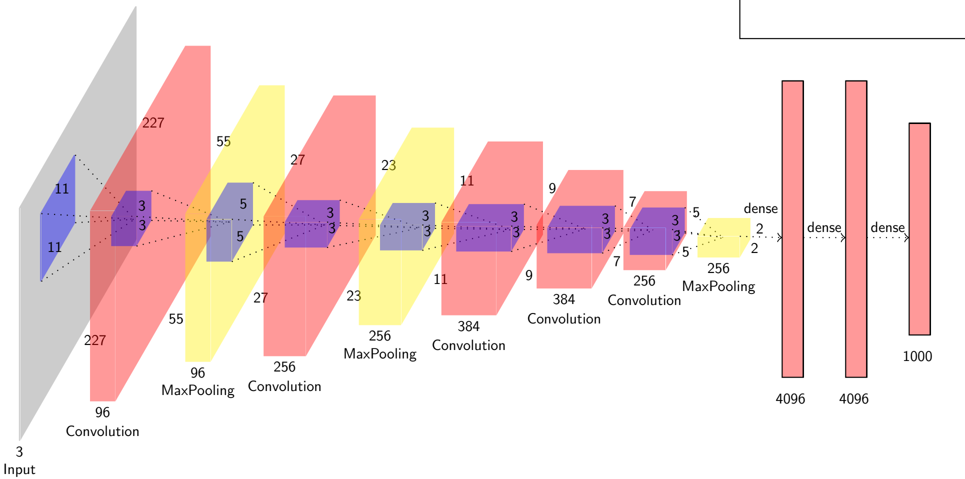
FC1
Parameters: $4096 \times 4096 = 16M$



106/113

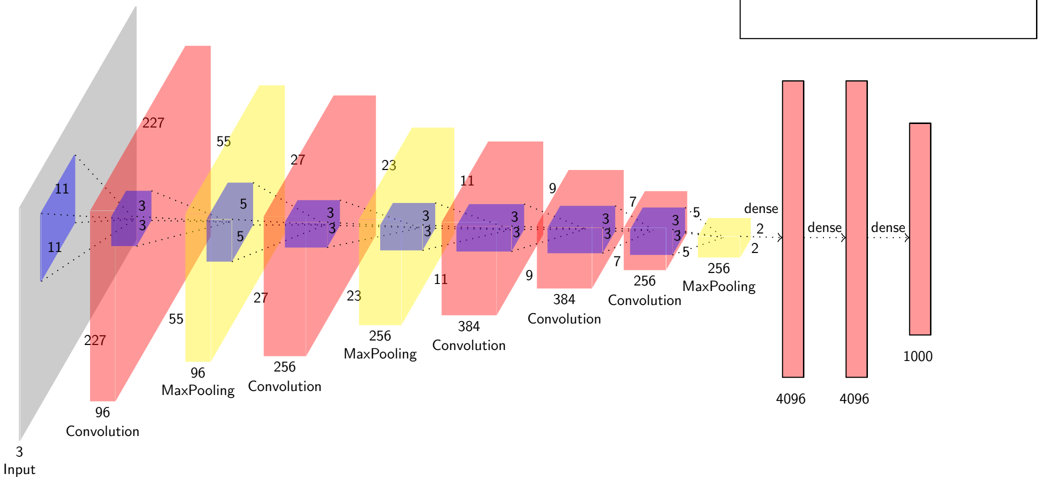
FC1

Parameters: $4096 \times 1000 = 4M$

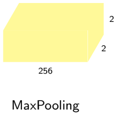


106/113

Total Parameters: $27.55M$

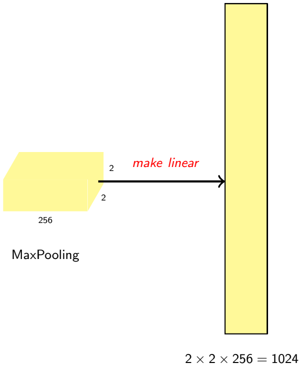


- Let us look at the connections in the fully connected layers in more detail



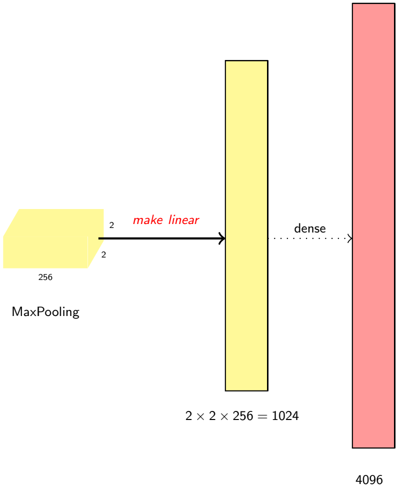
107/113

- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector

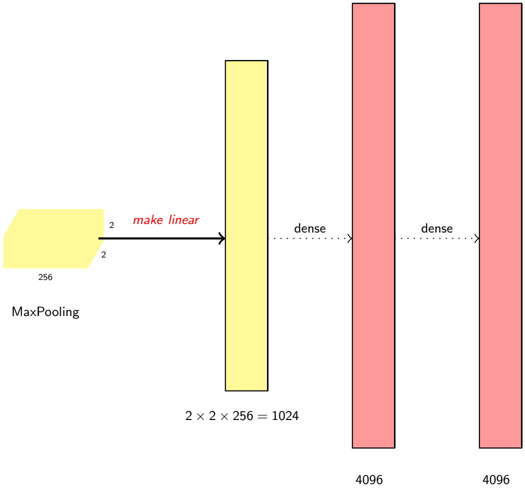


107/113

- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector
- This 1d vector is then densely connected to other layers just as in a regular feedforward neural network

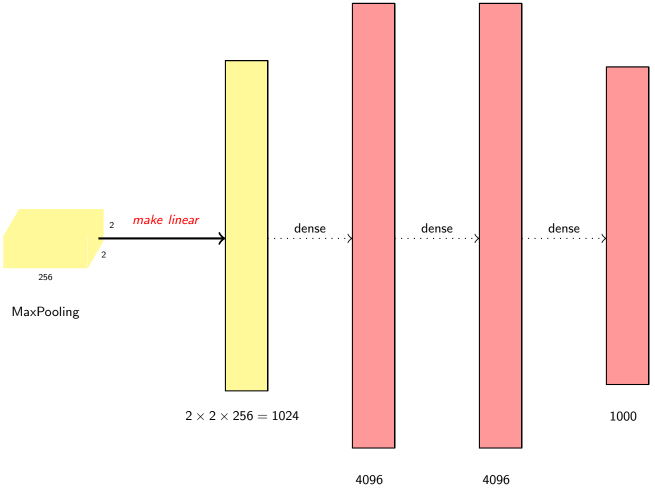


- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector
- This 1d vector is then densely connected to other layers just as in a regular feedforward neural network



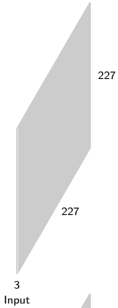
107/113

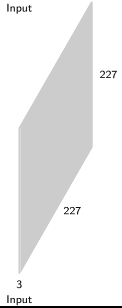
- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector
- This 1d vector is then densely connected to other layers just as in a regular feedforward neural network

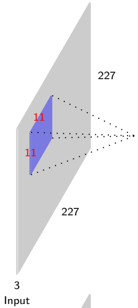


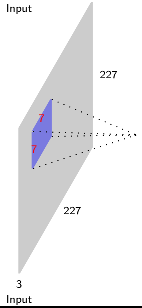
ImageNet Success Stories(roadmap for rest of the talk)

- AlexNet
- ZFNet
- VGGNet

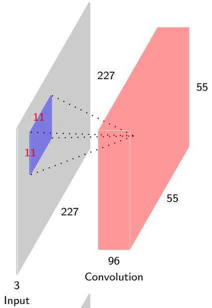

227
227
3
Input

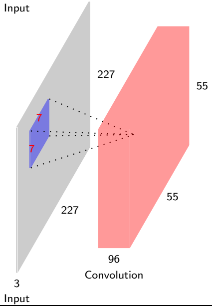

227
227
3
Input



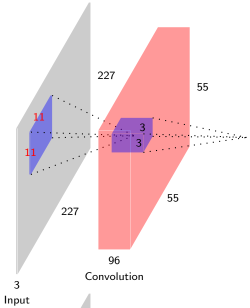


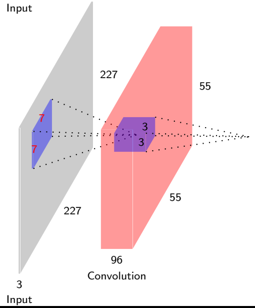
Layer1: $F = 11 \rightarrow 7$
Difference in Parameters
$((11 - 7) \times (11 - 7) \times 3) \times 96 = 4.6K$



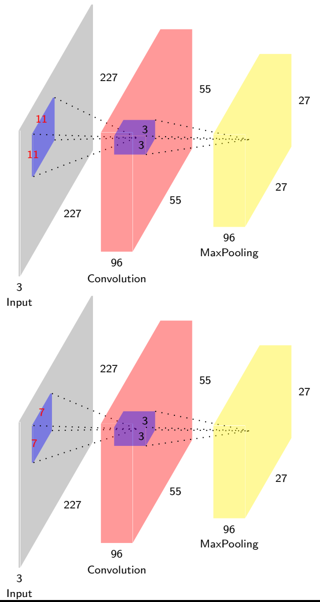


Layer1: $F = 11 \rightarrow 7$
Difference in Parameters
$((11-7) \times (11-7) \times 3) \times 96 = 4.6K$

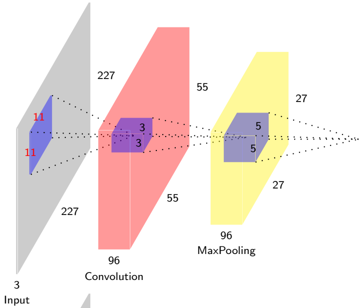


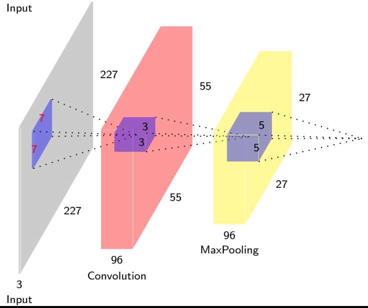


Layer2: No difference

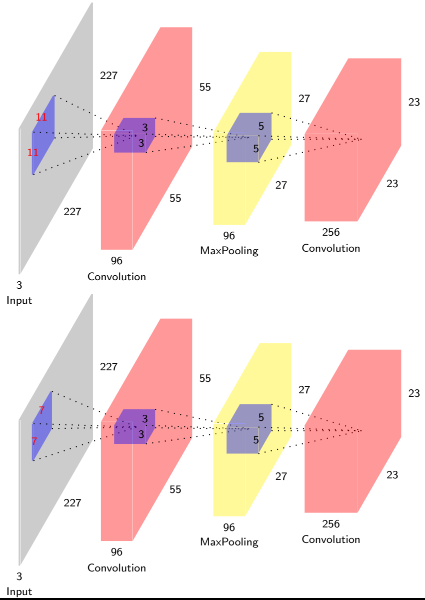


Layer2: No difference

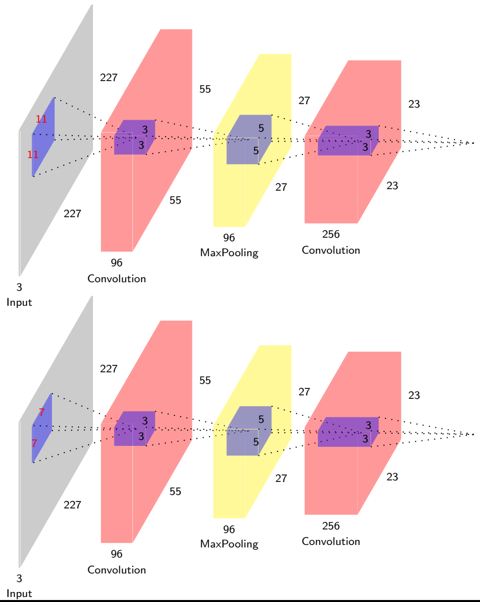




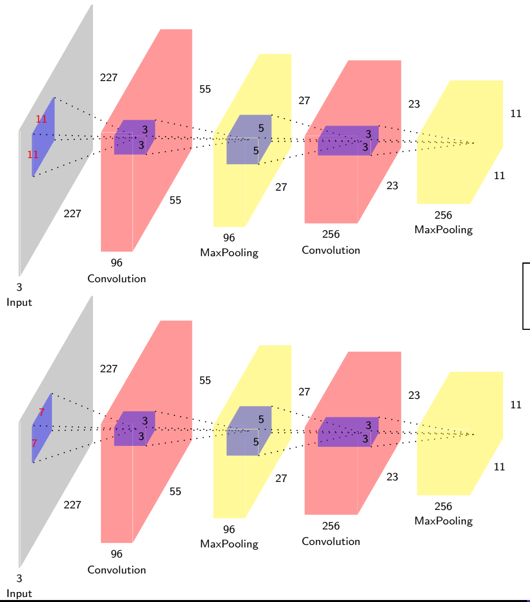
Layer3: No difference



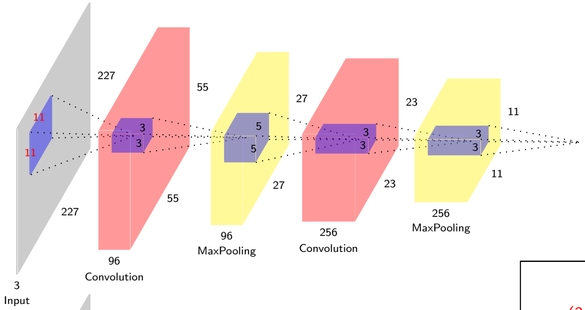
Layer3: No difference

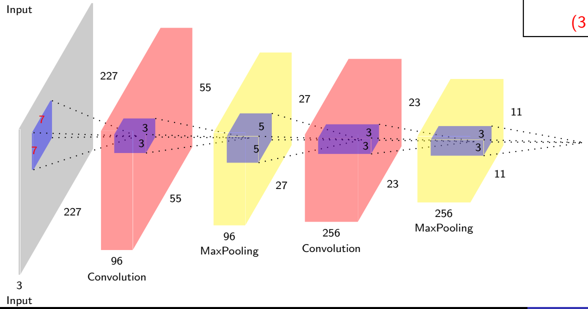


Layer4: No difference

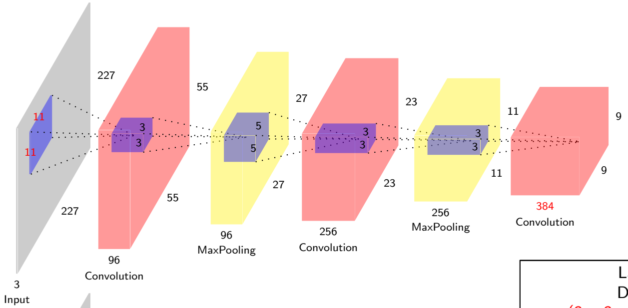


Layer4: No difference

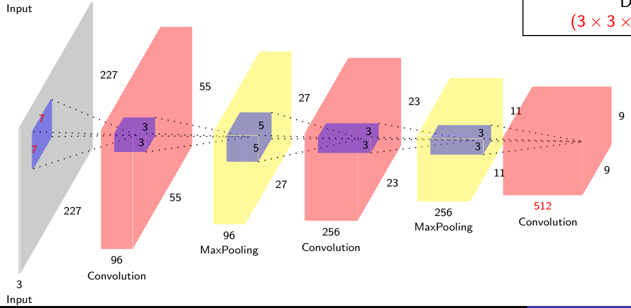


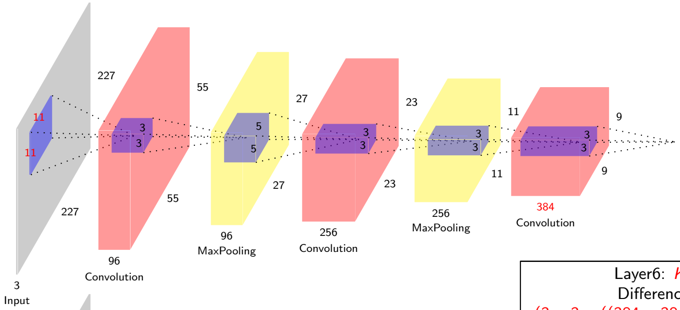


Layer5: $K = 384 \rightarrow 512$
Difference in Parameters
$(3 \times 3 \times 256) \times (512 - 384) = 0.29M$



Layer5: $K = 384 \rightarrow 512$
Difference in Parameters
$(3 \times 3 \times 256) \times (512 - 384) = 0.29M$

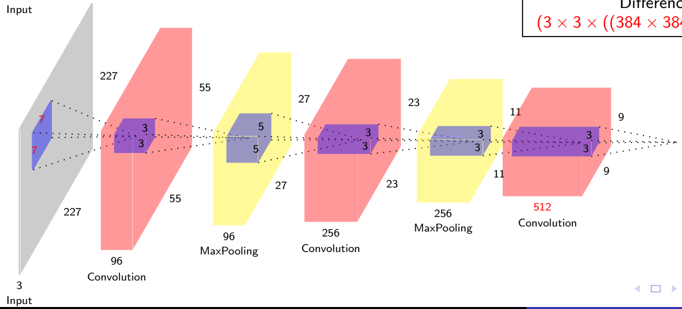


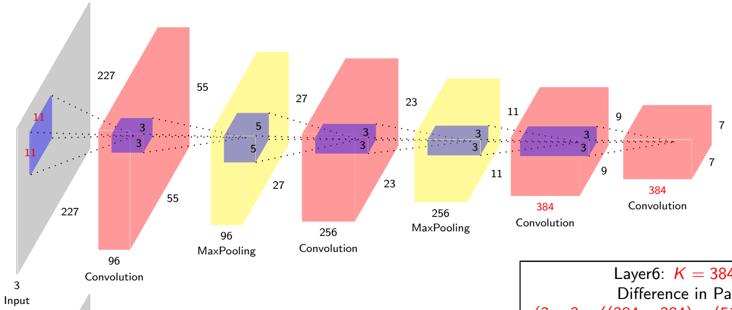


Layer6: $K = 384 \rightarrow 1024$

Difference in Parameters

$(3 \times 3 \times ((384 \times 384) - (512 \times 1024)) = 0.8M$

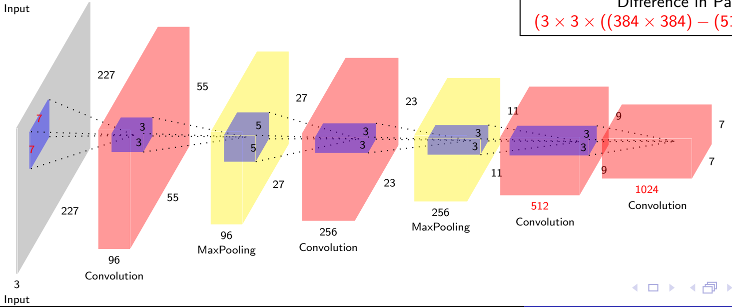


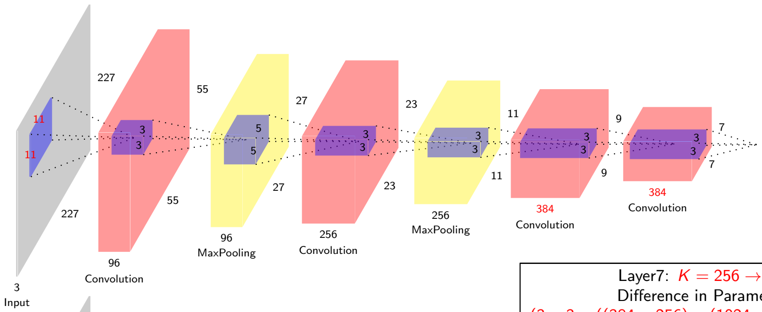


Layer6: $K = 384 \rightarrow 1024$

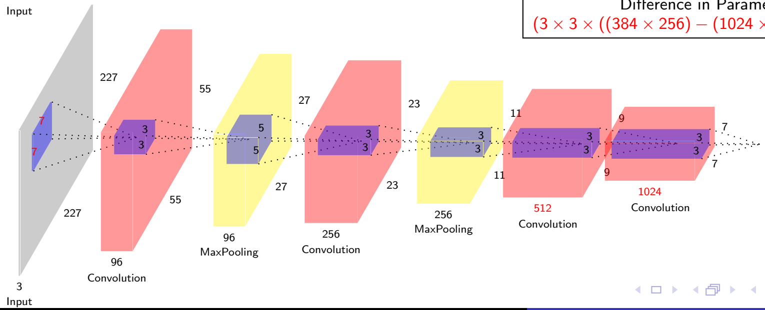
Difference in Parameters

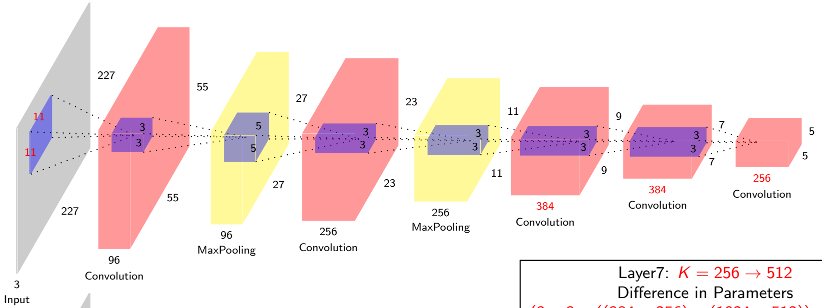
$(3 \times 3 \times ((384 \times 384) - (512 \times 1024)) = 0.8M$





Layer7: $K = 256 \rightarrow 512$
Difference in Parameters
$(3 \times 3 \times ((384 \times 256) - (1024 \times 512)) = 0.36M$

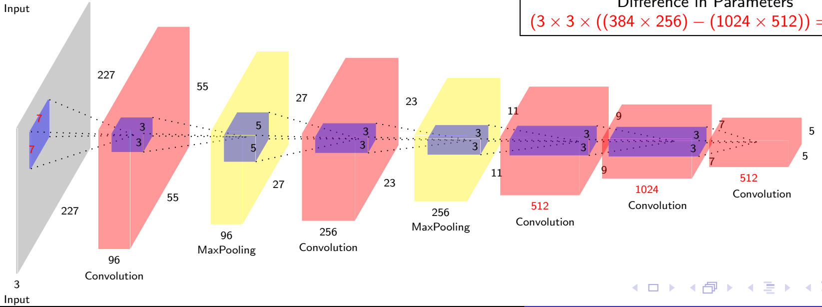


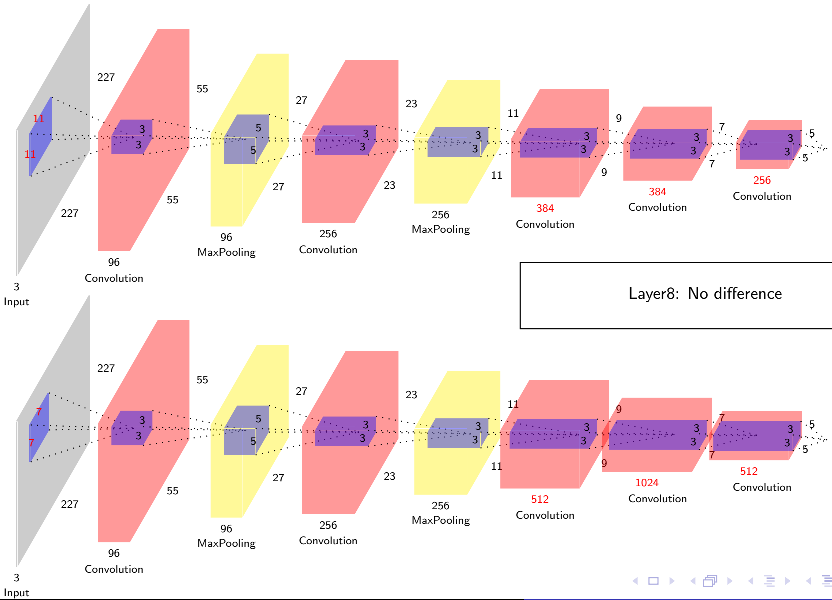


Layer7: $K = 256 \rightarrow 512$

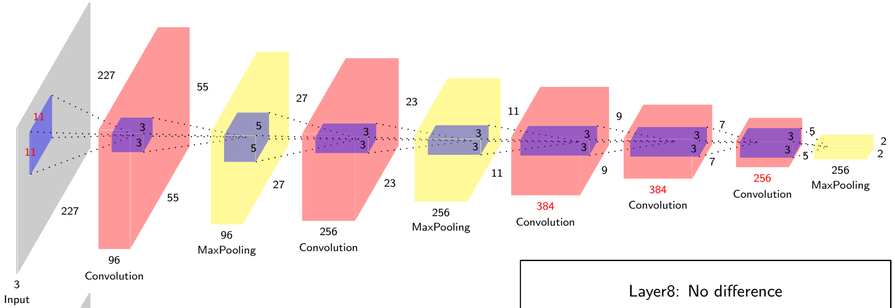
Difference in Parameters

$(3 \times 3 \times ((384 \times 256) - (1024 \times 512)) = 0.36M$

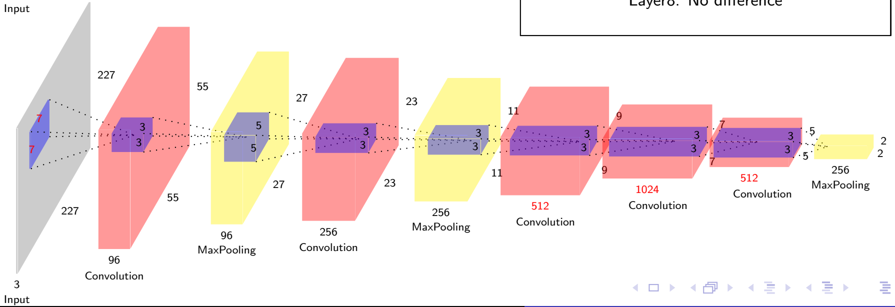


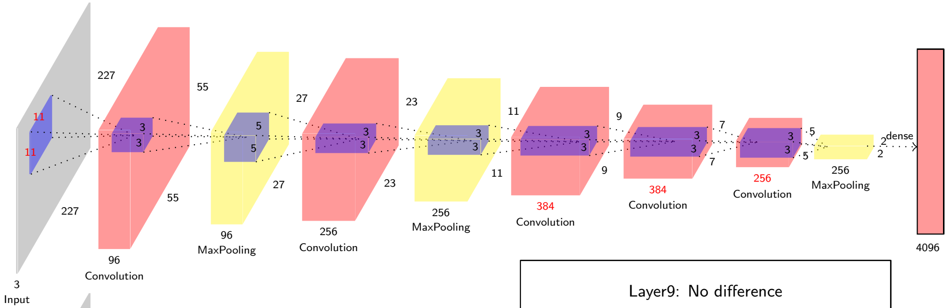


Layer8: No difference

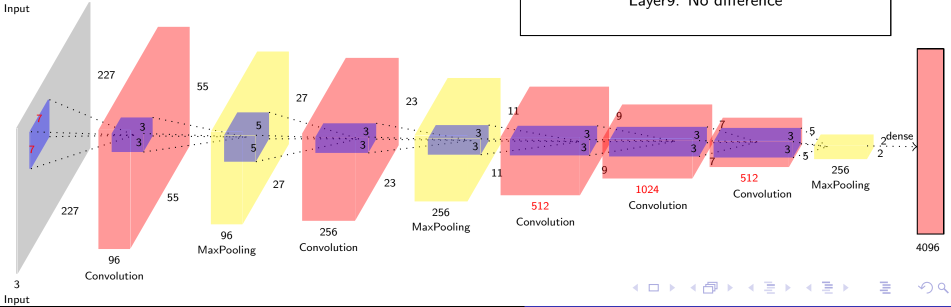


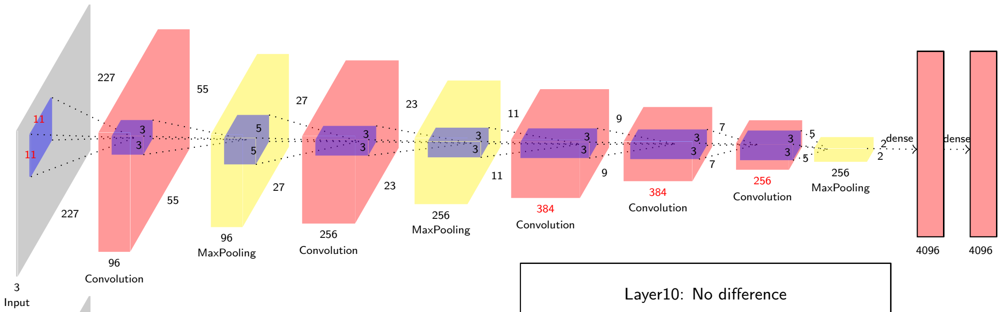
Layer8: No difference



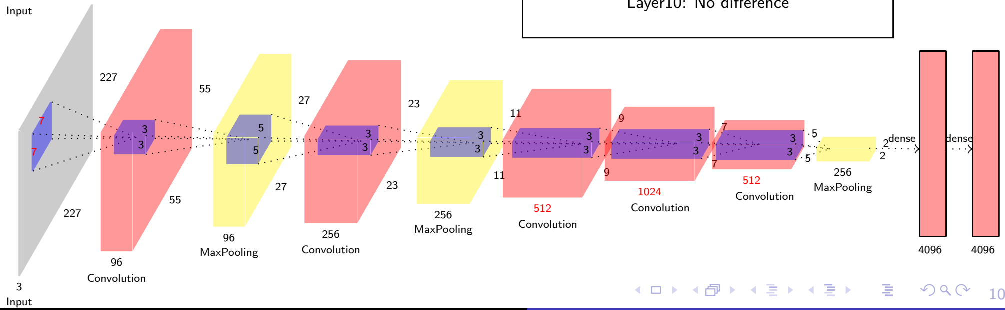


Layer9: No difference

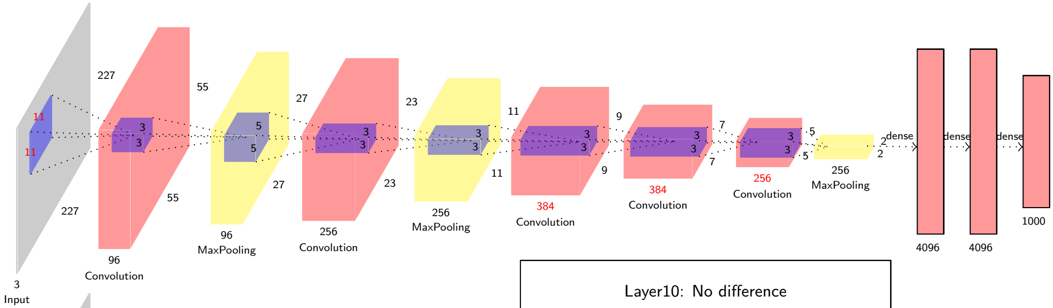




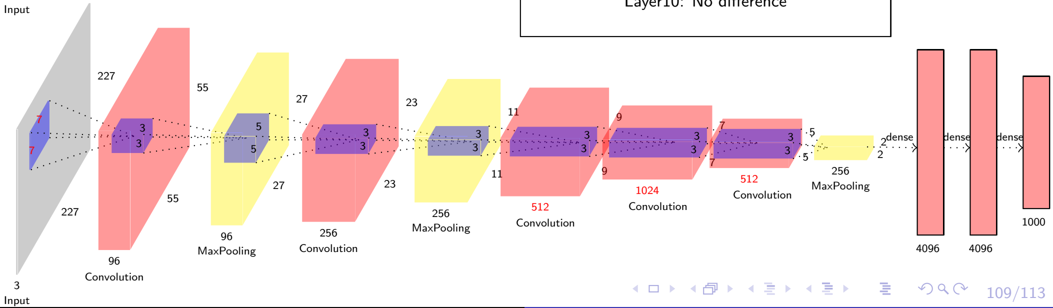
Layer10: No difference

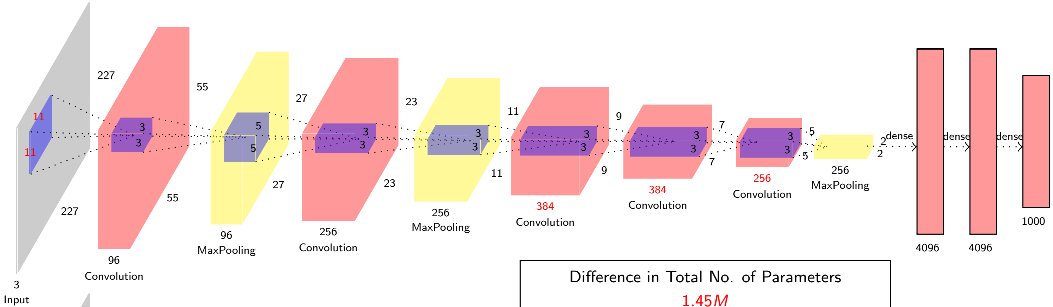


109/113



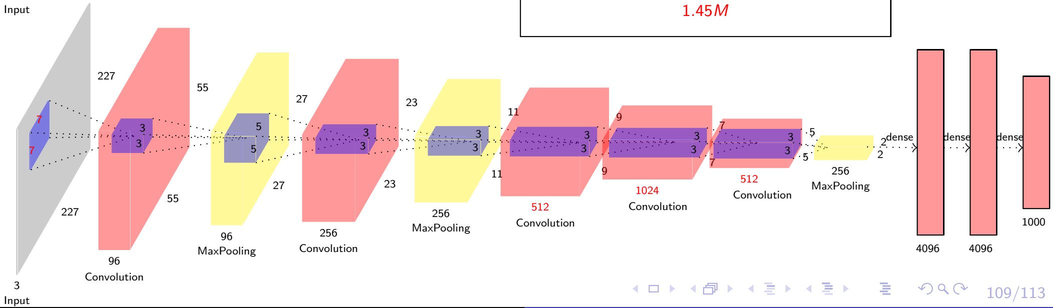
Layer10: No difference





Difference in Total No. of Parameters

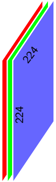
$1.45M$



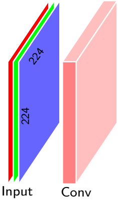
ImageNet Success Stories(roadmap for rest of the talk)

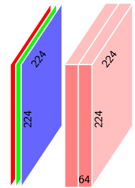
- AlexNet
- ZFNet
- VGGNet

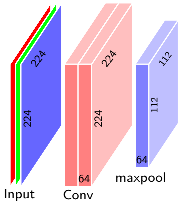
110/113

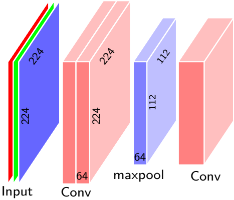


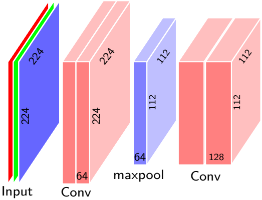
Input

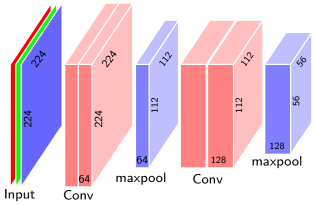

224
224
Input
Conv

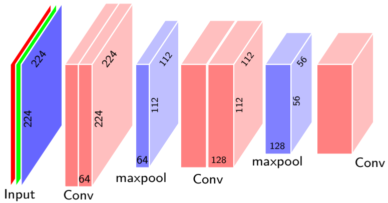

224
224
Input
224
224
64
Conv

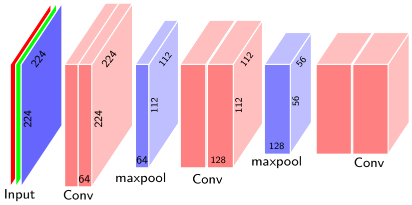

224
224
Input
224
224
64
Conv
112
112
64
maxpool

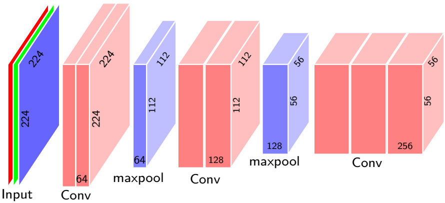

224
224
Input
224
224
64
Conv
112
112
64
maxpool
Conv

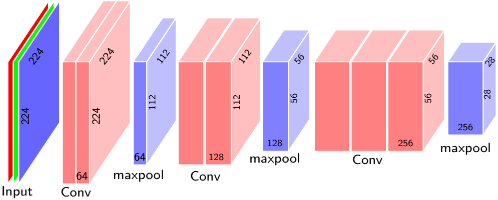

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv

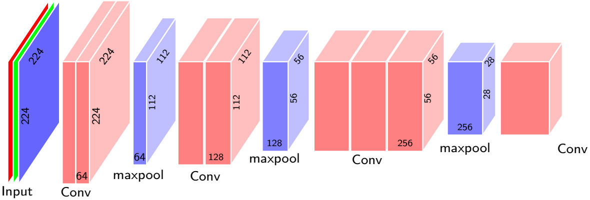

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool


224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
Conv

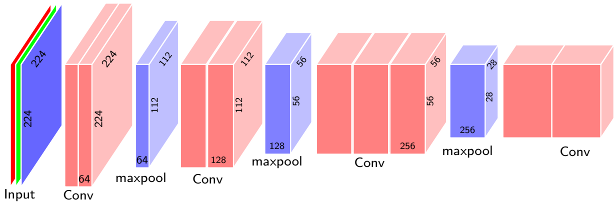

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
Conv

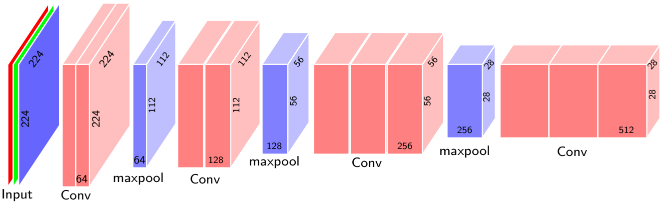

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv

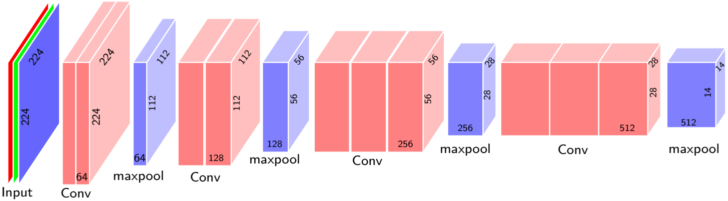

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool

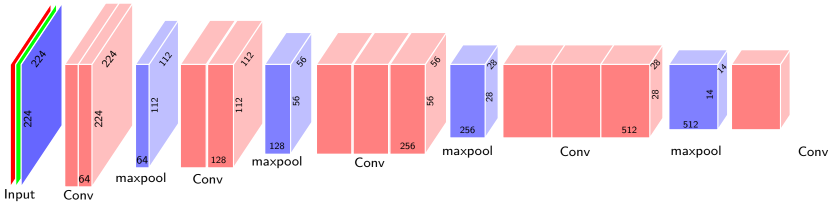

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
Conv

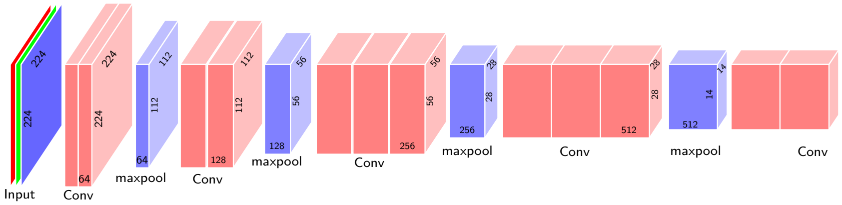
111/113

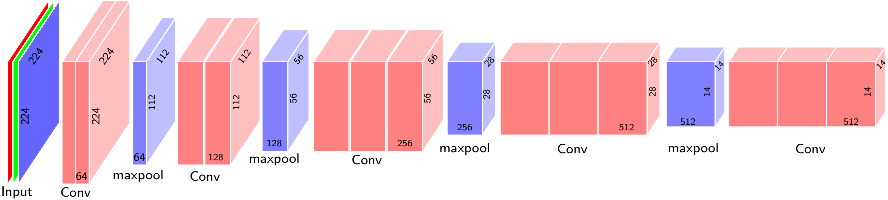

224
224
224
224
64
112
112
64
112
112
128
56
56
128
56
56
256
28
28
256
Input
Conv
maxpool
Conv
maxpool
Conv
maxpool
Conv

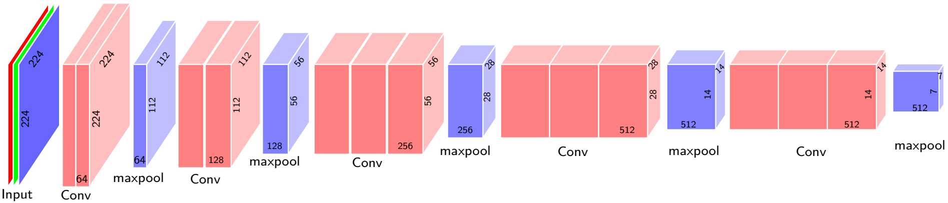

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv

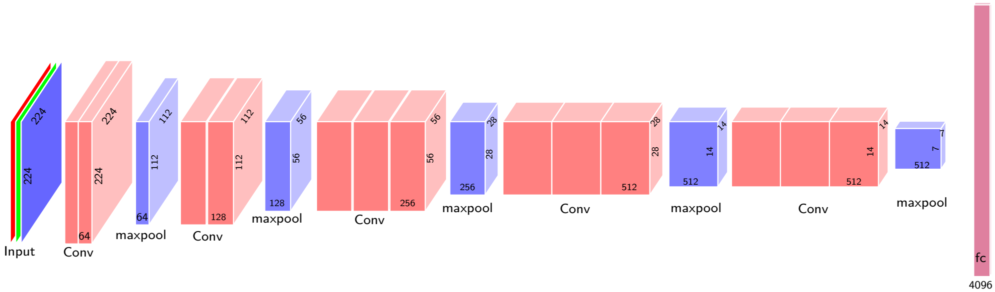

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool


224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
Conv

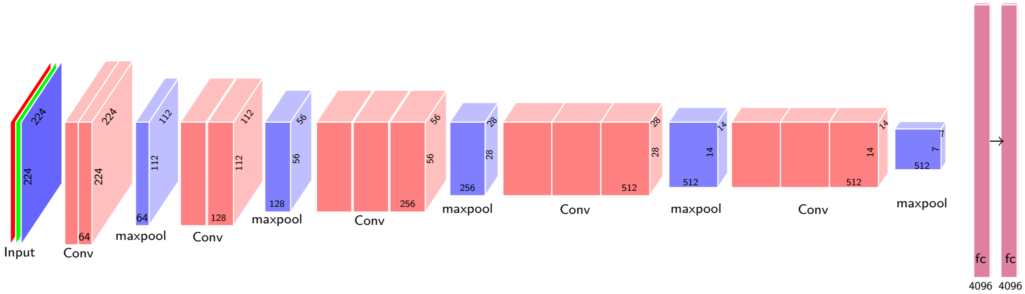

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
Conv


224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
14
14
512
Conv

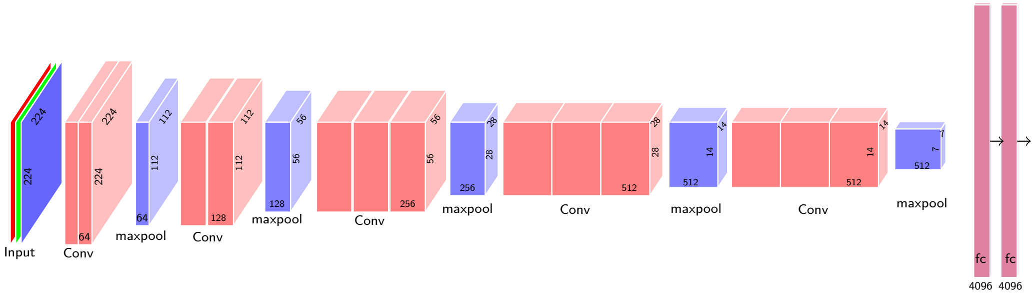

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
14
14
512
Conv
7
7
512
maxpool

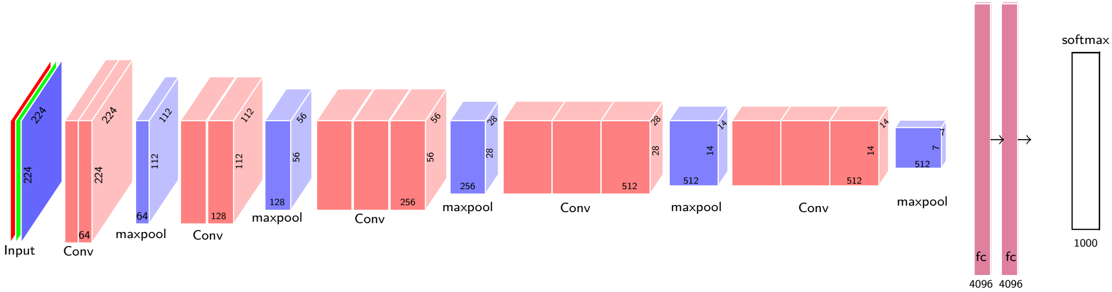


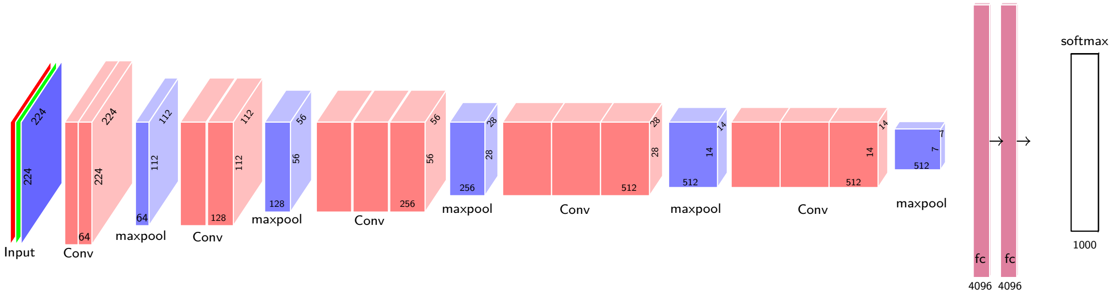
Input
224
224
Conv
64
224
224
maxpool
64
112
112
Conv
128
112
112
maxpool
128
56
56
Conv
256
56
56
maxpool
256
28
28
Conv
512
28
28
maxpool
512
14
14
Conv
512
14
14
maxpool
512
7
7
fc
4096



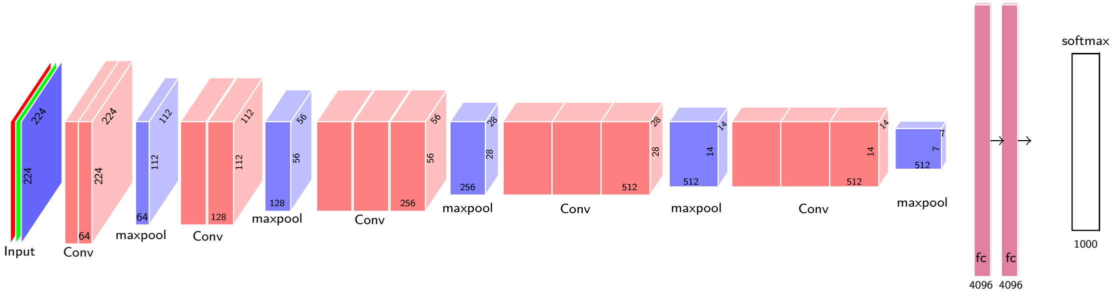
Input
224
224
Conv
64
224
224
maxpool
64
112
112
Conv
128
112
112
maxpool
128
56
56
Conv
256
56
56
maxpool
256
28
28
Conv
512
28
28
maxpool
512
14
14
Conv
512
14
14
maxpool
512
7
7
fc
fc
4096
4096


224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
14
14
512
Conv
7
7
512
maxpool
fc
4096
fc
4096

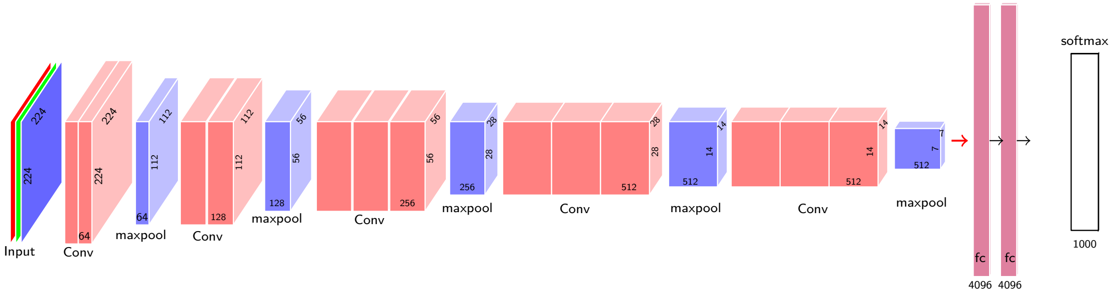

Input
224
224
Conv
64
224
224
maxpool
64
112
112
Conv
128
112
112
maxpool
128
56
56
Conv
256
56
56
maxpool
256
28
28
Conv
512
28
28
maxpool
512
14
14
Conv
512
14
14
maxpool
512
7
7
fc
4096
fc
4096
softmax
1000



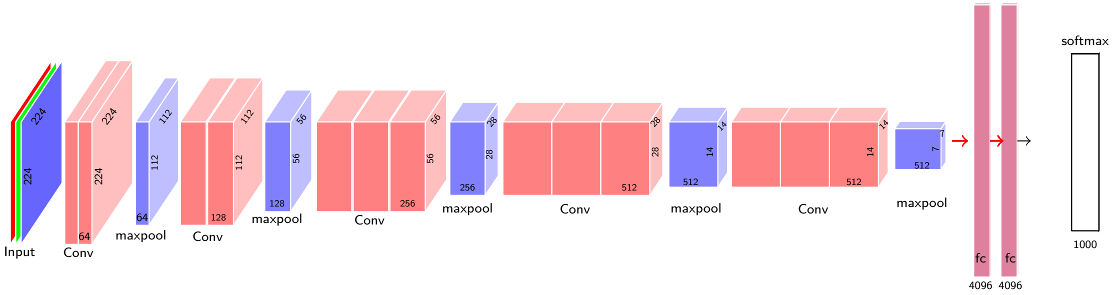
- Kernel size is 3x3 throughout



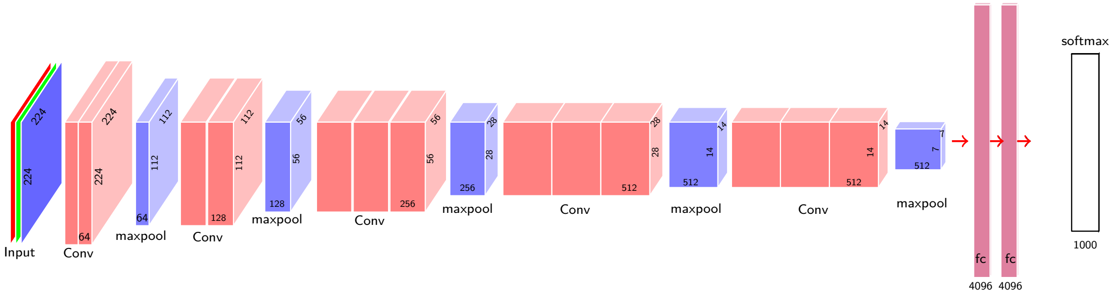
- Kernel size is 3x3 throughout
- Total parameters in non FC layers $= \sim 16M$



- Kernel size is 3x3 throughout
- Total parameters in non FC layers $= \sim 16M$
- $Total\ Parameters\ in\ FClayers =$

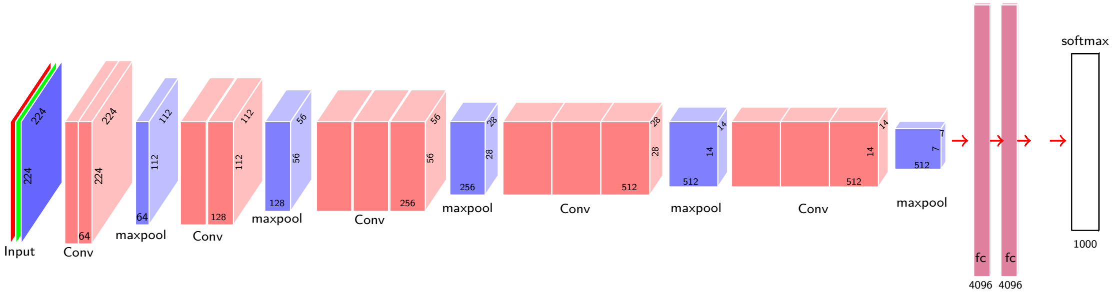


- Kernel size is 3x3 throughout
- Total parameters in non FC layers $= \sim 16M$
- $Total\ Parameters\ in\ FClayers\ =\ (512x7x7x4096)$

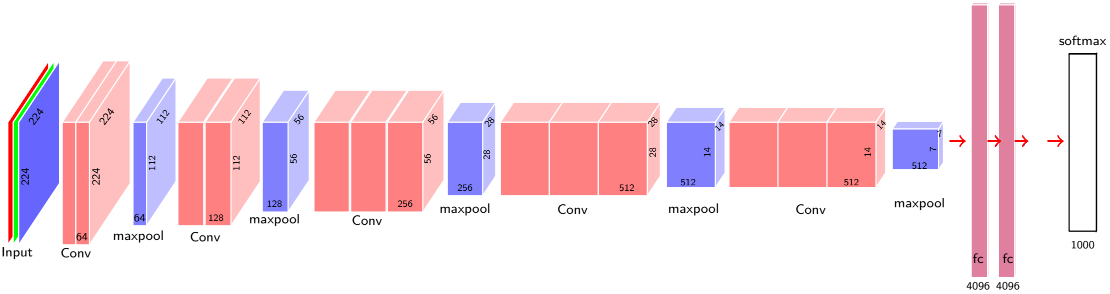


- Kernel size is 3x3 throughout
- Total parameters in non FC layers $= \sim 16M$
- $Total\ Parameters\ in\ FClayers =$ $(512x7x7x4096) + (4096x4096)$

111/113

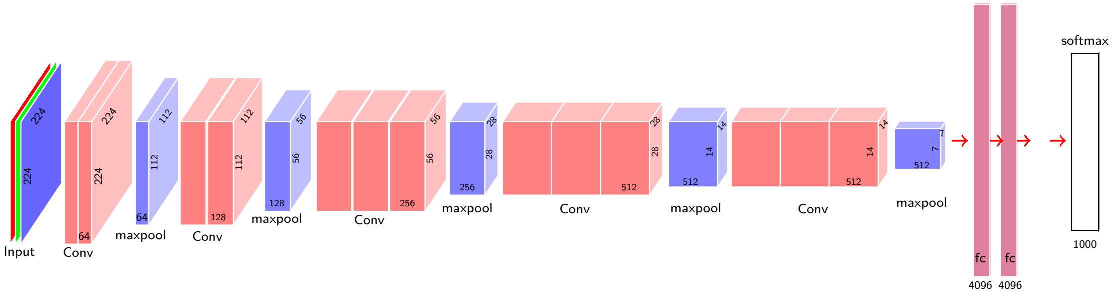


- Kernel size is 3x3 throughout
- Total parameters in non FC layers $= \sim 16M$
- *Total Parameters in FClayers* $=$ (512x7x7x4096) + (4096x4096) + (4096x1024)



- Kernel size is 3x3 throughout
- Total parameters in non FC layers $= \sim 16M$
- $Total\ \ Parameters\ \ in\ \ FClayers = \ (512x7x7x4096) + (4096x4096)\ \ + (4096x1024) = \sim 122M$

111/113



- Kernel size is 3x3 throughout
- Total parameters in non FC layers $= \sim 16M$
- $Total\ Parameters\ in\ FClayers =\ (512x7x7x4096) + (4096x4096)\ + (4096x1024) = \sim 122M$
- Most parameters are in the first FC layer ($\sim$ 102M)

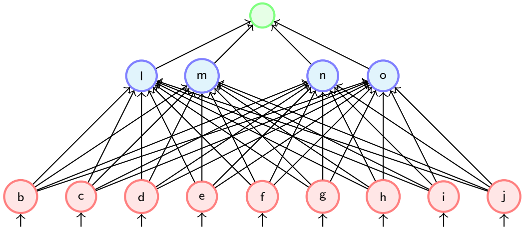
- How do we train a convolutional neural network ?

Input

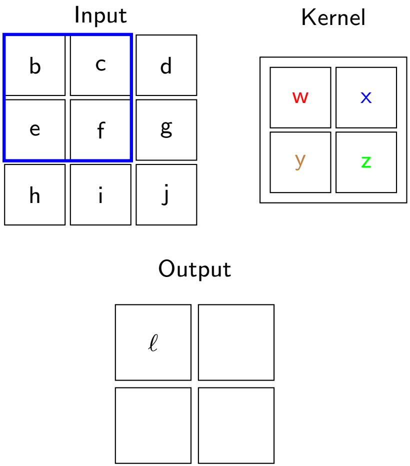
b	c	d
e	f	g
h	i	j

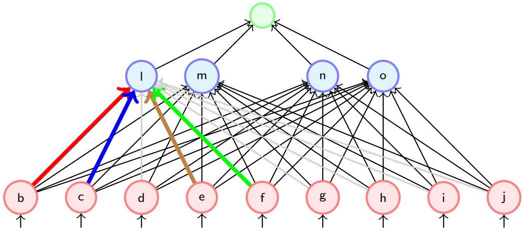
Kernel

w	x
y	z

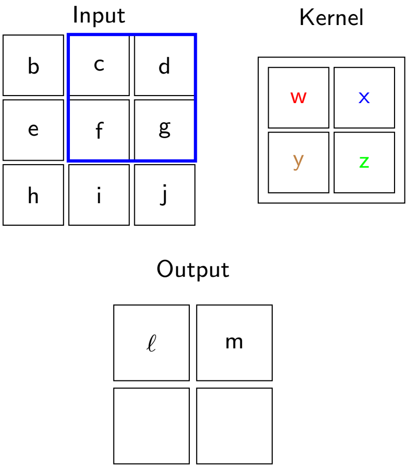


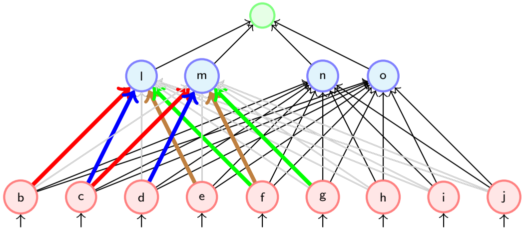
- A CNN can be implemented as a feedforward neural network



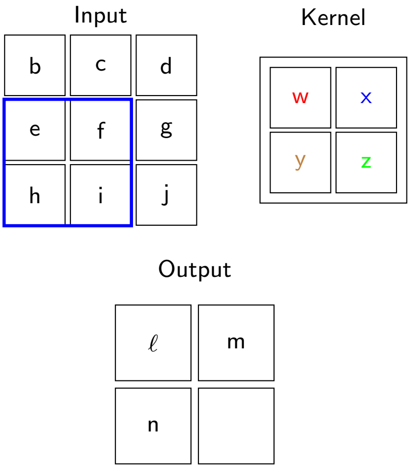


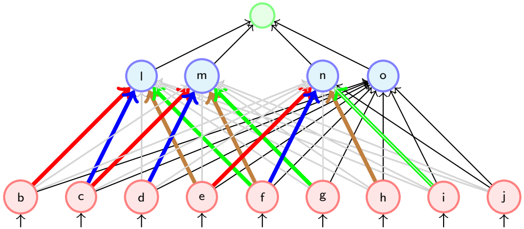
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active



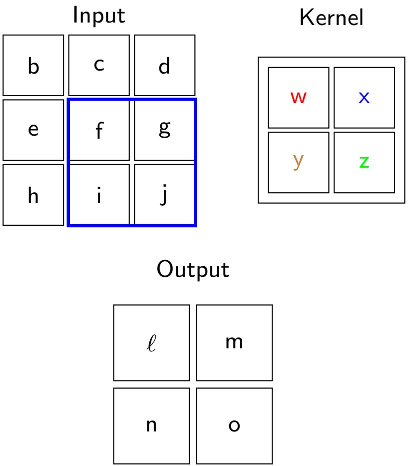


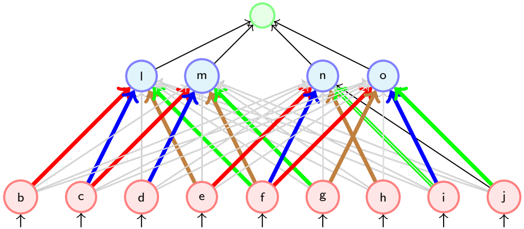
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero





- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero





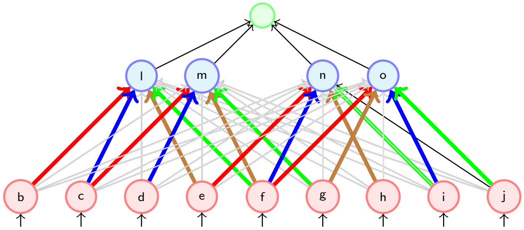
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

Input

b	c	d
e	f	g
h	i	j

Kernel

w	x
y	z



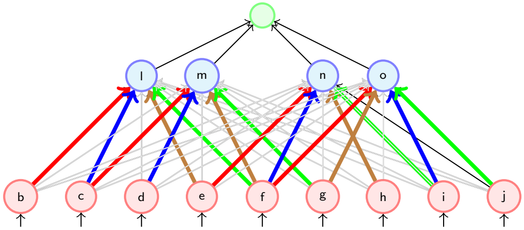
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

Input

b	c	d
e	f	g
h	i	j

Kernel

w	x
y	z



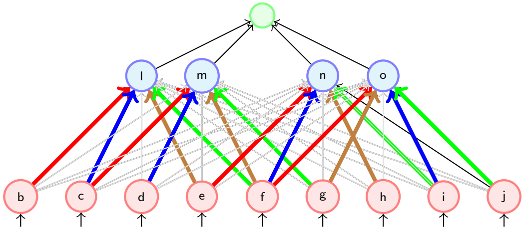
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

Input

b	c	d
e	f	g
h	i	j

Kernel

w	x
y	z



- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

- We can thus train a convolution neural network using backpropagation by thinking of it as a feedforward neural