Building Blocks of Deep Learning

Aditya Prakash

April 9, 2018

Online version link (with animations)

Scream

Shipwreck of Minataur

Udnie

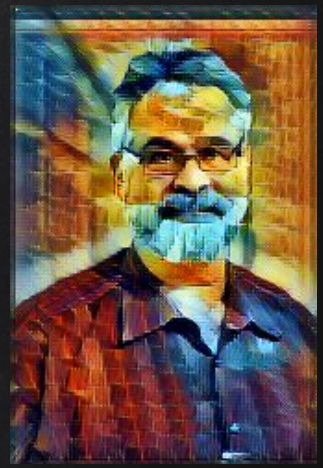
Wave

Rain Princess

Art Lives Forever







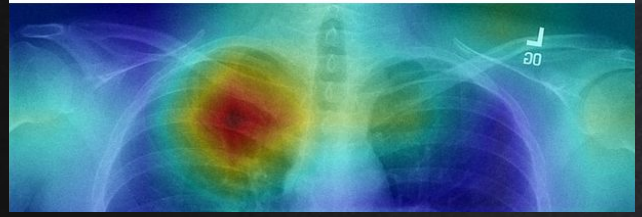


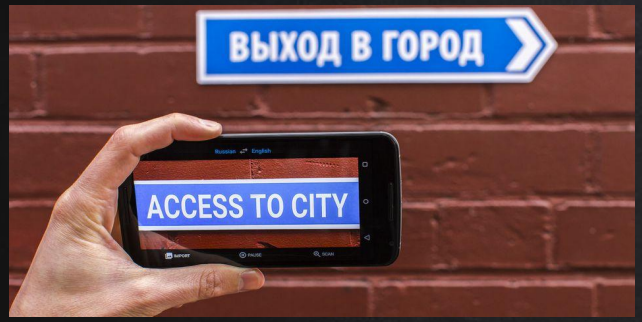

Munch Turner Picabia Hokusai Afremov

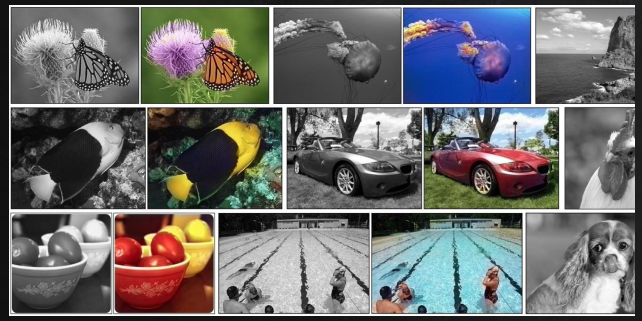
Art Lives Forever
ARTIST

New AI Can Diagnose Pneumonia Better Than Doctors

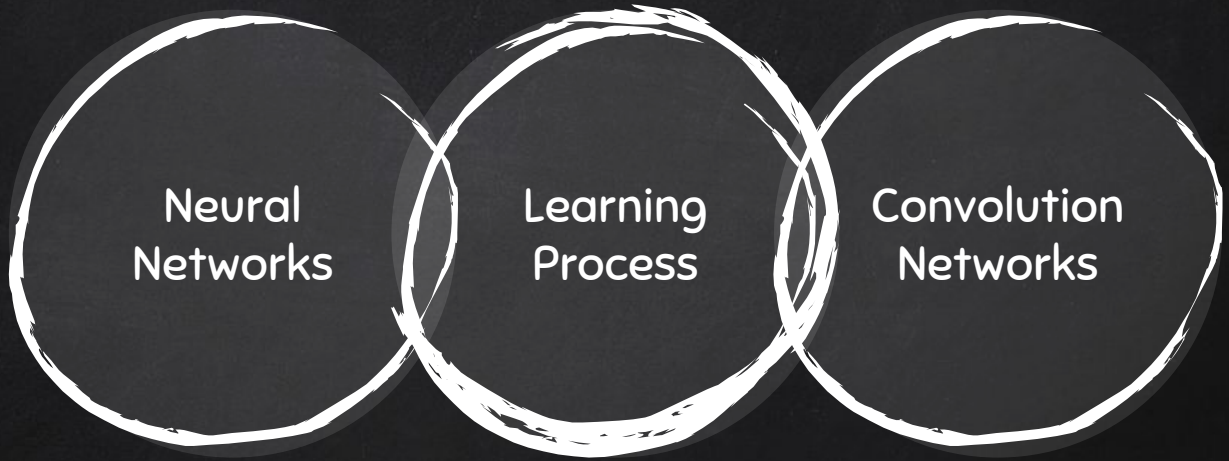
The software can greatly help in avoiding the misdiagnosis of pneumonia.







What we will Learn



Textbook

http://www.deeplearningbook.org

Course

http://cs231n.stanford.edu

Quick Guide

http://neuralnetworksanddeeplearning.com

Youtube: 3Blue1Brown

Machine Learning recap

Apply a prediction function to a feature representation of given input (x) and get the desired output.

f() = “apple”

f() = “bird”

f() = “bike”

Machine Learning recap

Training Data $\longrightarrow$ model $\longrightarrow$ output

x f $\tilde{y}$

$f(x) \longrightarrow \tilde{y}$

$y - \tilde{y} \longrightarrow$ error

$L(y, \tilde{y}) \longrightarrow$ Loss/Objective

Loss function L quantifies how unhappy you would be if you used 'f' to make predictions on x.
It is the objective we want to minimize

Machine Learning recap

Training Data $\longrightarrow$ model $\longrightarrow$ output

x f $\tilde{y}$

$$\text{TrainLoss} = \sum_{i} \text{Loss}(f(x_i), y_i)$$

$$\text{Model} = \underset{f}{\text{argmin}} \ \text{TrainLoss}$$

Find 'f' such that it minimizes the 'training loss', and hope that this is also true for 'test' loss.

Machine Learning recap

Training Data $\longrightarrow$ model $\longrightarrow$ output

x f $\tilde{y}$

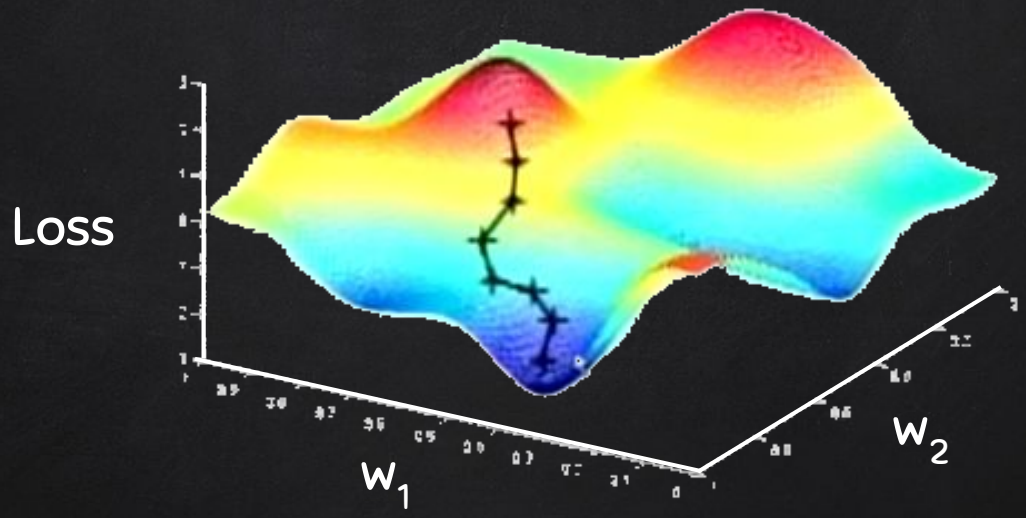
$$\text{Model} = \underset{f}{\operatorname{argmin}}\ \text{TrainLoss}$$

$$W \leftarrow W - \eta \nabla_w \text{TrainLoss}(f)$$

$$w = w - \eta \frac{\partial L}{\partial w}$$

Process of updating 'weights' like this is called gradient descent

Gradient Descent



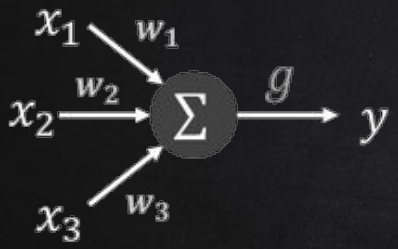
Artificial Neural Network

Artificial Neural Network

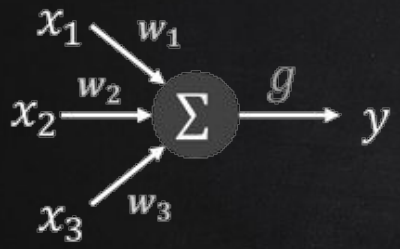
Organic is overrated

Neuron

Artificial Neuron



Artificial Neuron



$$y = g\left(\sum_j W_j x_j + b\right)$$

Input from unit j

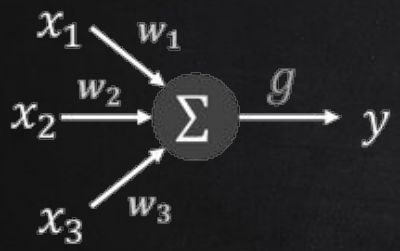
Output of unit

Activation Function

Linear weights

Bias unit

Artificial Neuron



$$y = g\left(\sum_j W_j x_j + b\right)$$

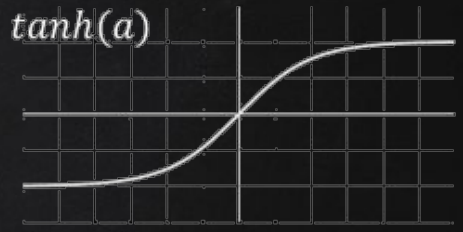
Input from unit j

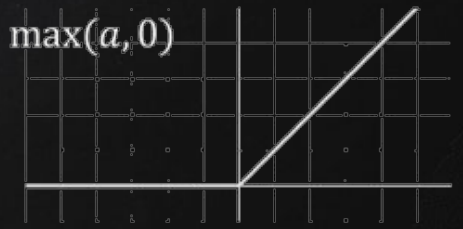
Output of unit

Activation Function

Linear weights

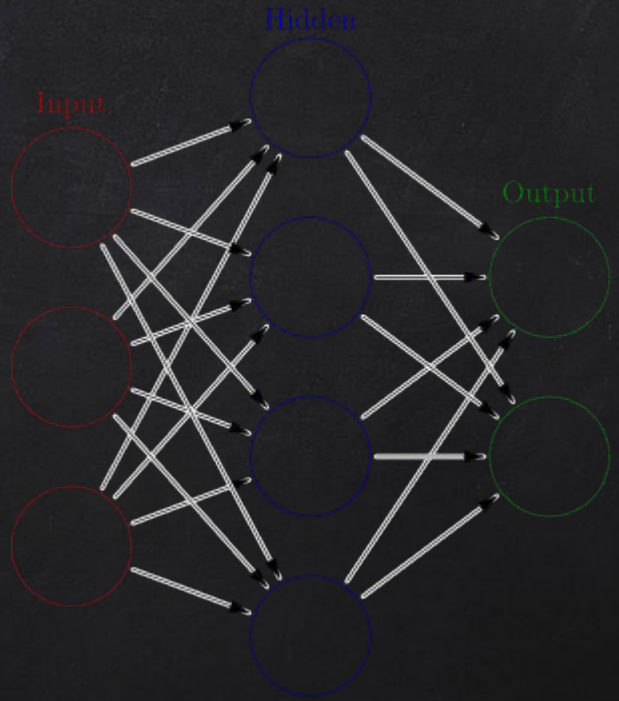
Bias unit





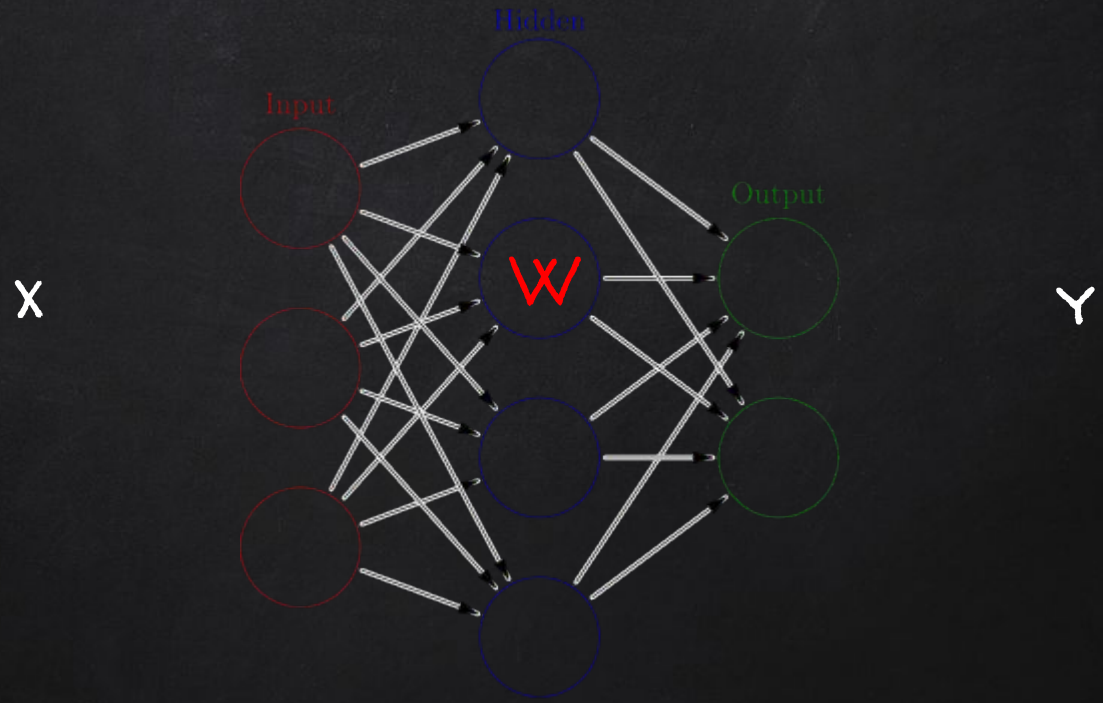
Artificial Neural Networks

X



Y

Artificial Neural Networks



Artificial Neural Networks

Artificial Neural Networks

Image Classification

Choose among the following ---

Cairn terrier (b) Norwich terrier (c) Australian terrier

Image Classification

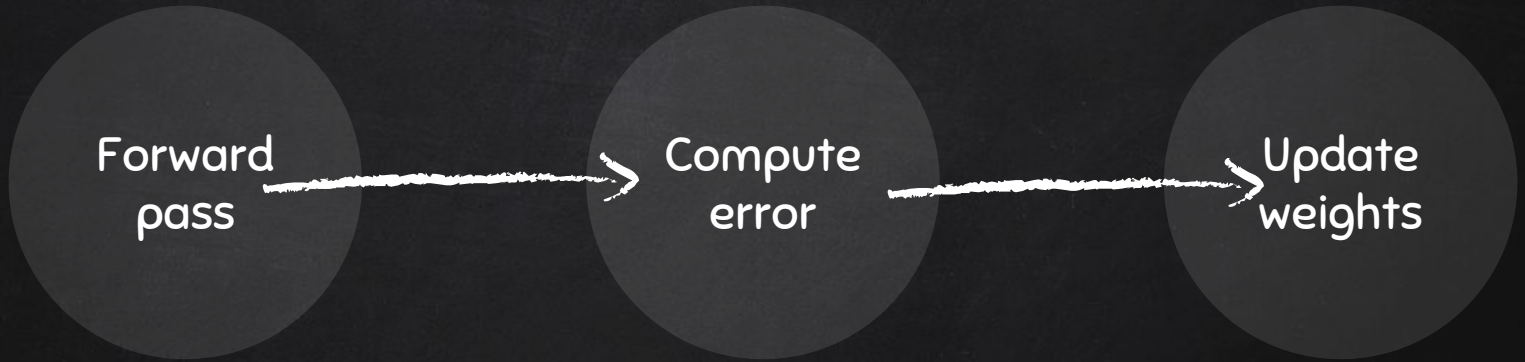
Choose among the following ---

Cairn terrier (b) Norwich terrier (c) Australian terrier

Using Magic

Neural networks and backpropagation

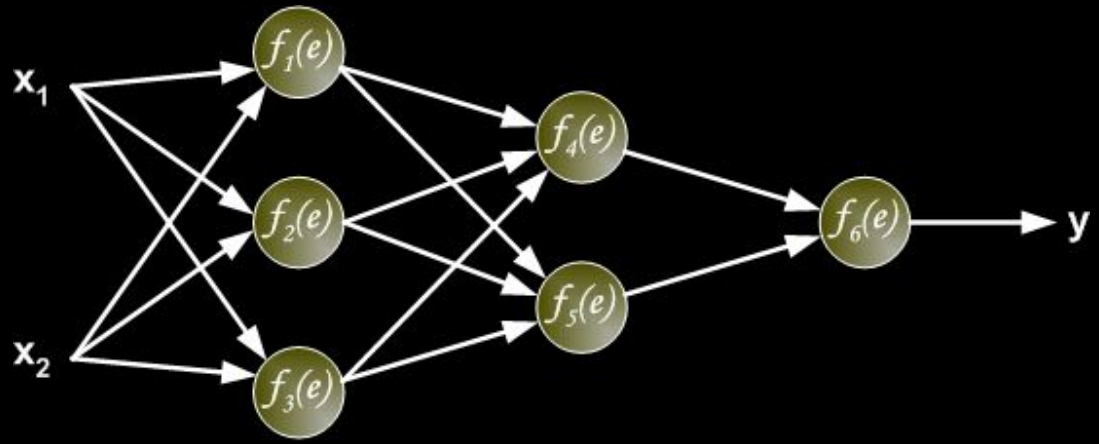
Back Propagation



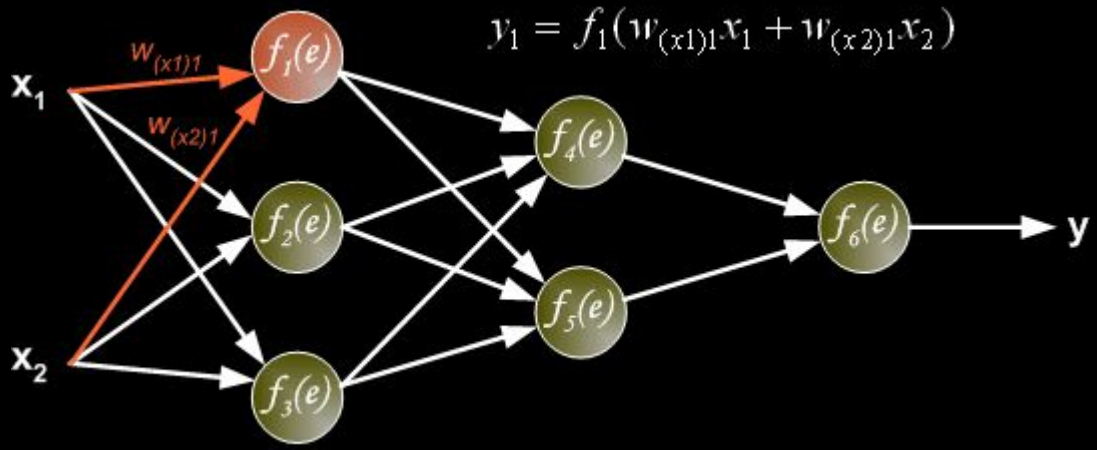
Back Propagation

Forward Pass

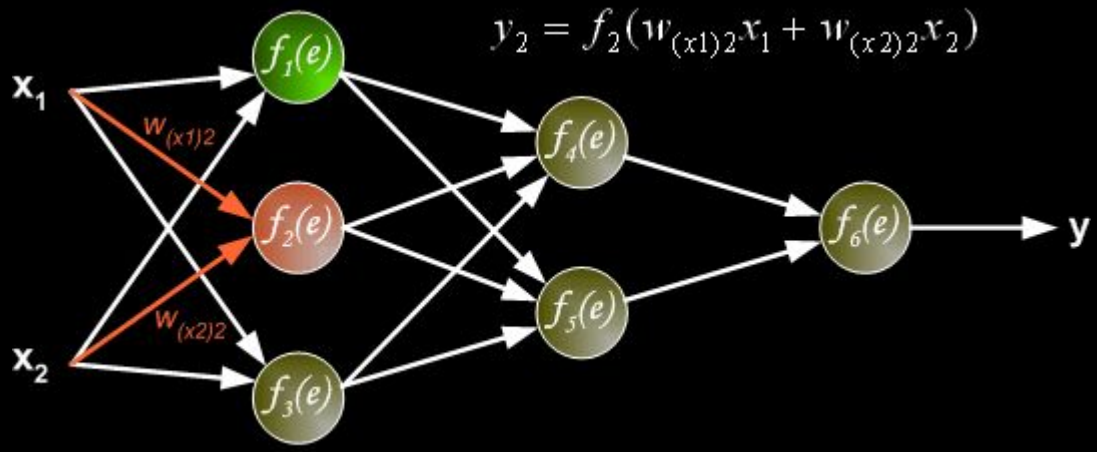
Forward Pass



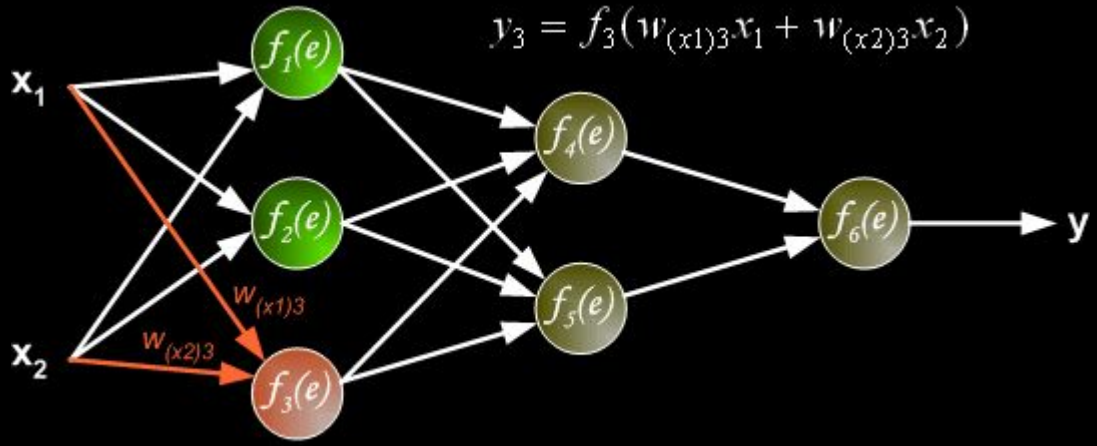
Forward Pass



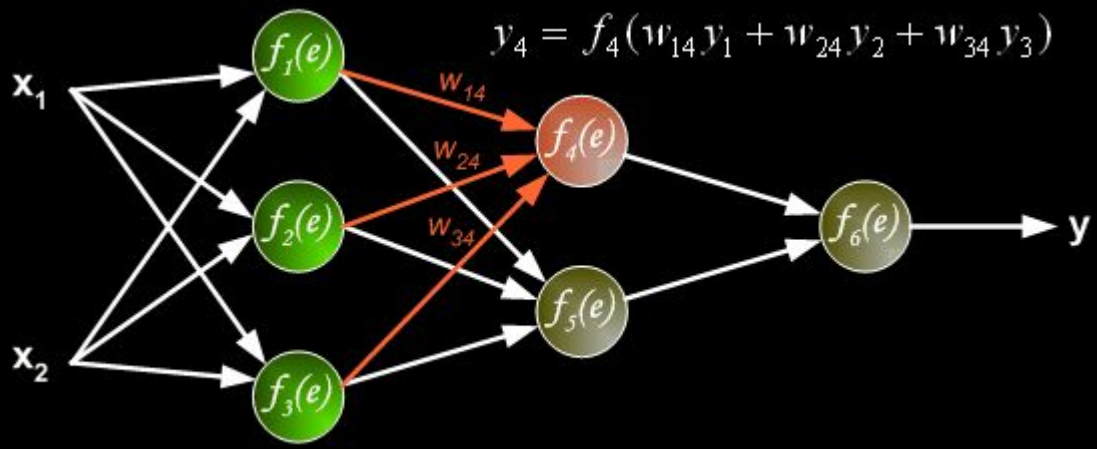
Forward Pass



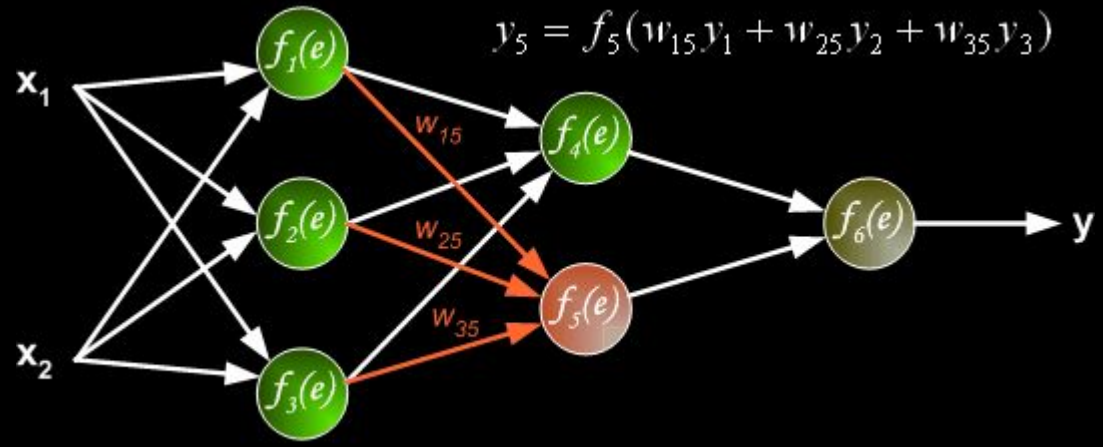
Forward Pass



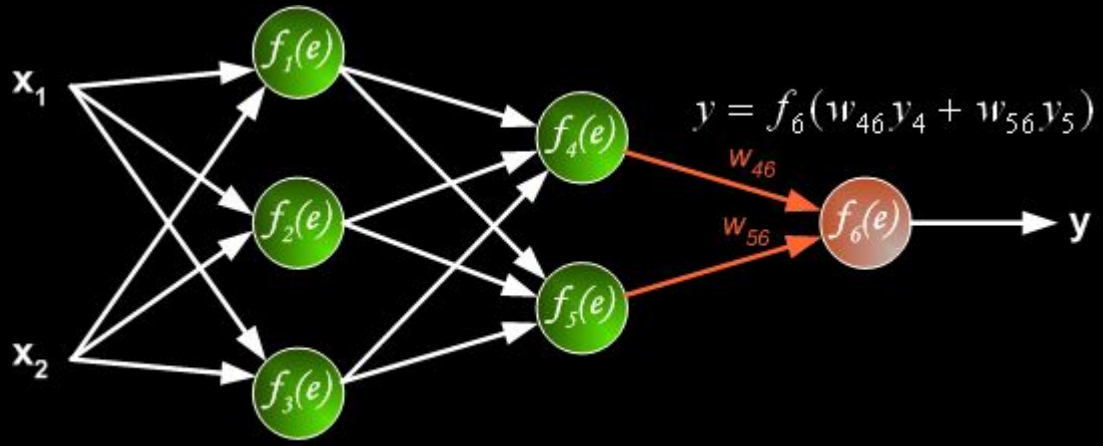
Forward Pass



Forward Pass



Forward Pass



Back Propagation

Compute error

Back Propagation

Derivative Chain Rule

$$(f \circ g(x))' = g'(x) f'(g(x))$$

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

Back Propagation

Derivative Chain Rule

$$(f \circ g(x))' = g'(x) f'(g(x))$$

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

$$f(x) = sin(5x) \qquad f'(x) = 5[cos(5x)]$$

Back Propagation

Derivative Chain Rule

$$L = z - y$$

Back Propagation

Derivative Chain Rule

$$L = z - y$$

$$\frac{\partial L}{\partial w}$$

Back Propagation

Derivative Chain Rule

$$x \xrightarrow{w} f(xw) \longrightarrow y \qquad L = z - y$$

$$\frac{\partial L}{\partial w} = \frac{\partial f(xw)}{\partial w} \frac{\partial L}{\partial f(xw)}$$

Back Propagation

Derivative Chain Rule

$$L = z - y$$

$$\frac{\partial L}{\partial w} = \frac{\partial f(xw)}{\partial w} \frac{\partial L}{\partial f(xw)}$$

$$\frac{\partial L}{\partial w} = \frac{\partial (xw)}{\partial w} \frac{\partial f(xw)}{\partial (xw)} \frac{\partial L}{\partial f(xw)}$$

Back Propagation

Derivative Chain Rule

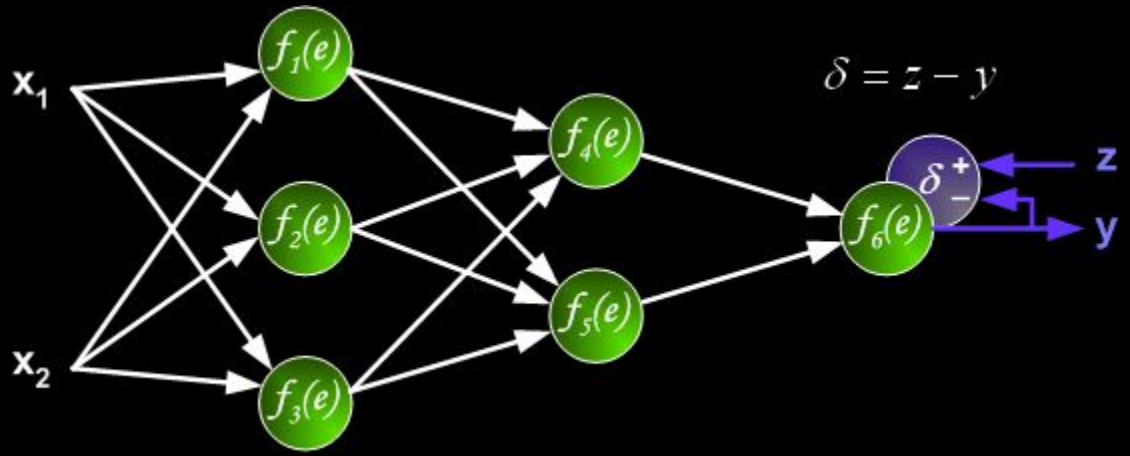
$$L = z - y$$

$$\frac{\partial L}{\partial w} = \frac{\partial f(xw)}{\partial w} \frac{\partial L}{\partial f(xw)}$$

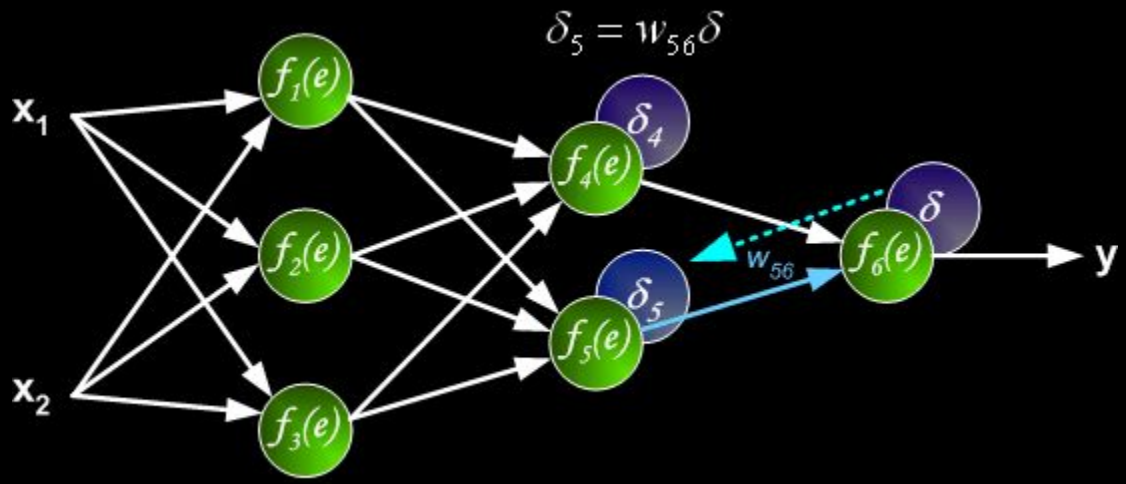
$$\frac{\partial L}{\partial w} = \frac{\partial (xw)}{\partial w} \frac{\partial f(xw)}{\partial (xw)} \frac{\partial L}{\partial f(xw)}$$

$$\frac{\partial L}{\partial w} = x \frac{\partial f(e)}{\partial (e)} \delta$$

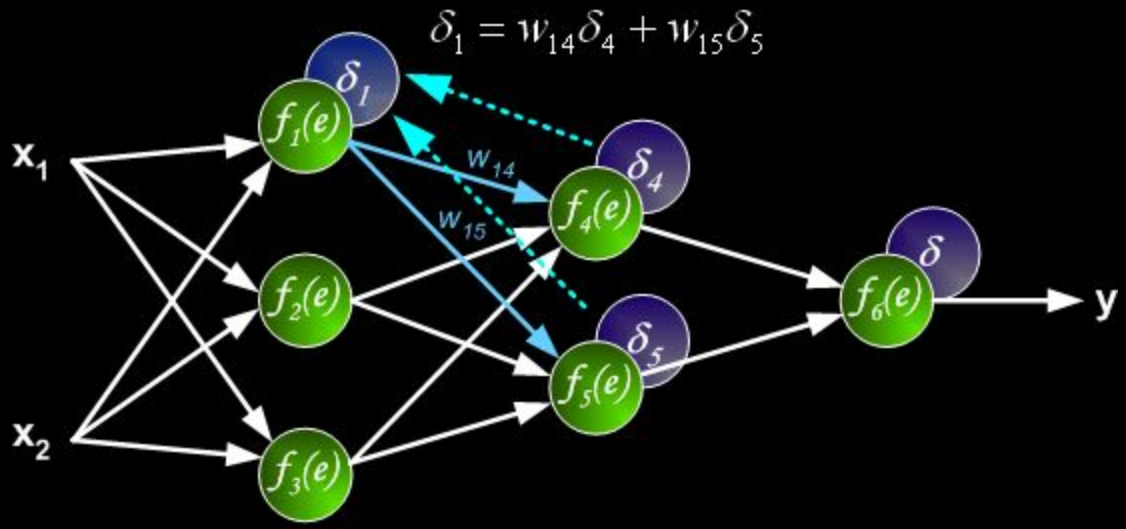
Compute Error



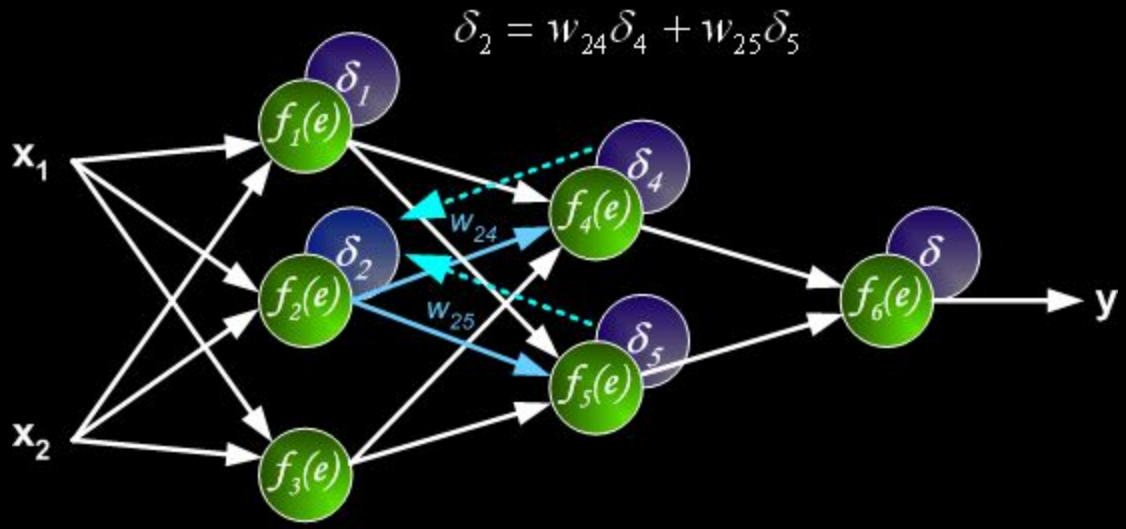
Compute Error



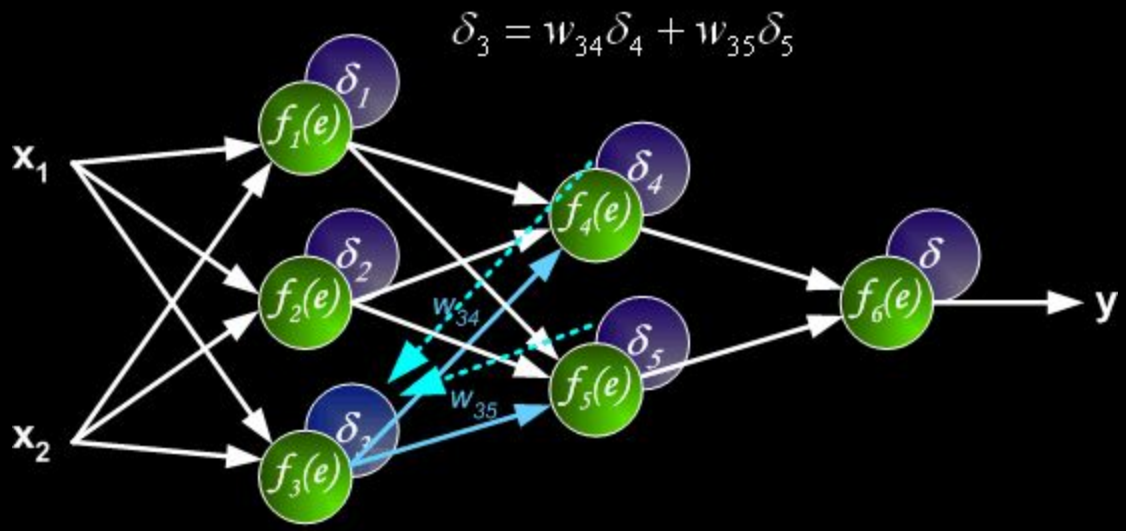
Compute Error



Compute Error



Compute Error



Back Propagation

Update weights

Back Propagation

Update weights

$$w = w - \eta \frac{\partial L}{\partial w}$$

$$\frac{\partial L}{\partial w} = x \quad \frac{\partial f(e)}{\partial (e)} \quad \delta$$

Back Propagation

Update weights

$$w = w - \eta \frac{\partial L}{\partial w}$$

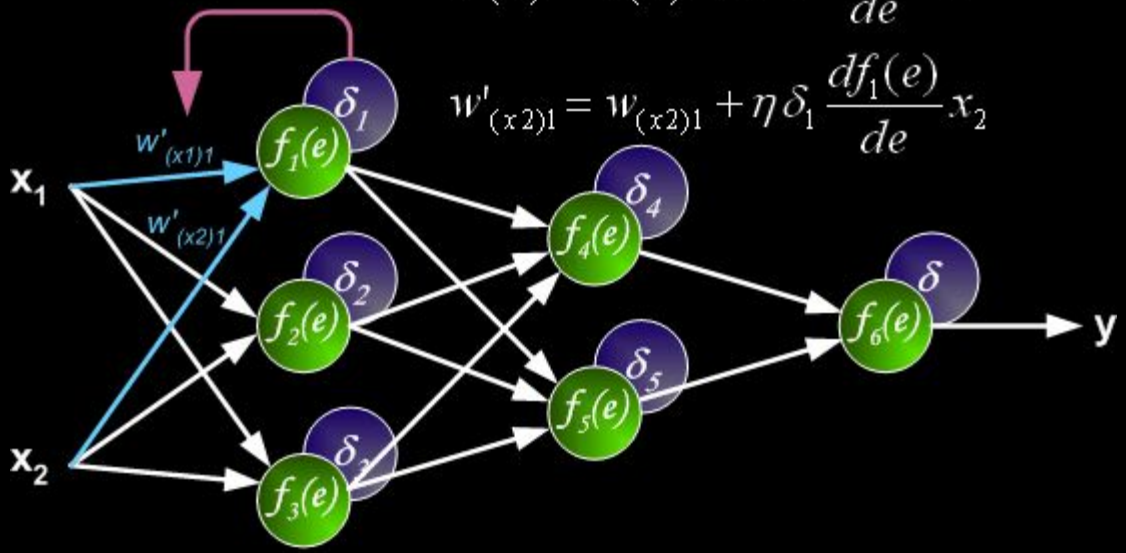
$$\frac{\partial L}{\partial w} = x \ \frac{\partial f(e)}{\partial (e)} \ \delta$$

$$w = w - \eta x \ \frac{\partial f(e)}{\partial (e)} \delta$$

Update Weights

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$

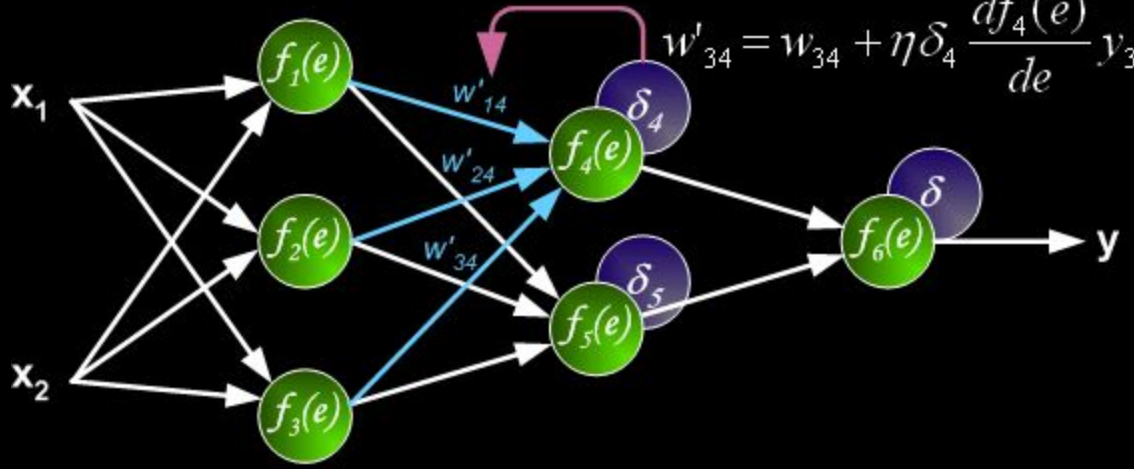


Update Weights

$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$

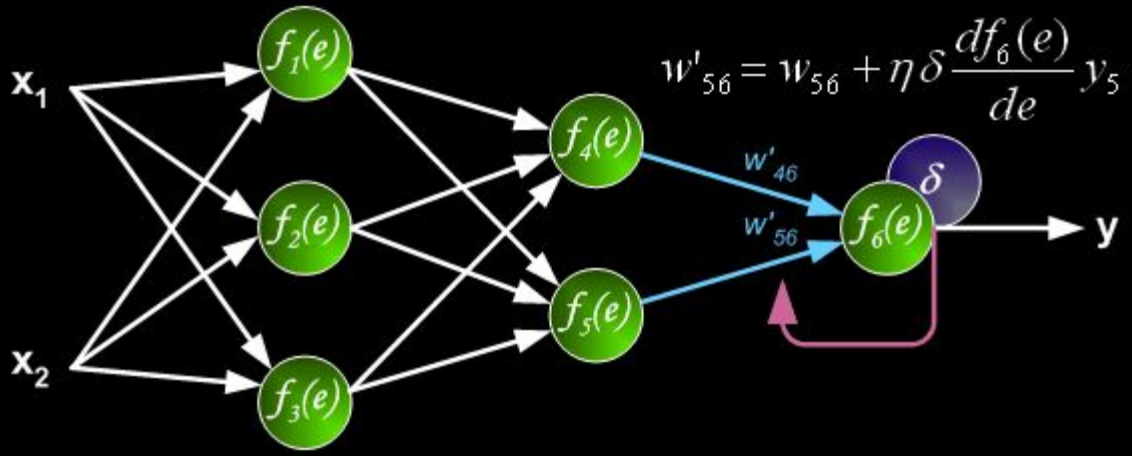
$$w'_{34} = w_{34} + \eta \delta_4 \frac{df_4(e)}{de} y_3$$



Update Weights

$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



Summary

1. Random weights
2. Forward Pass
3. Compute Cost
4. Backward Pass
5. Update weights

...

N. Profit???

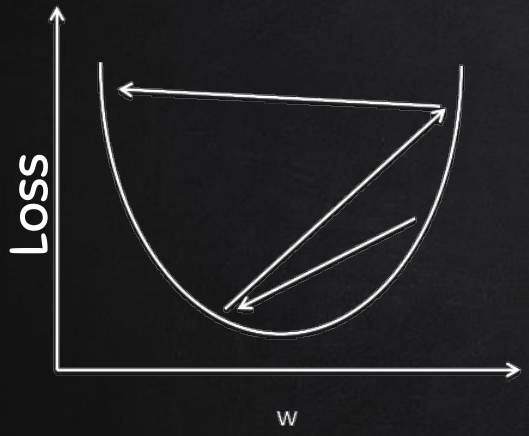
Summary

1. Random weights
2. Forward Pass
3. Compute Cost
4. Backward Pass
5. Update weights

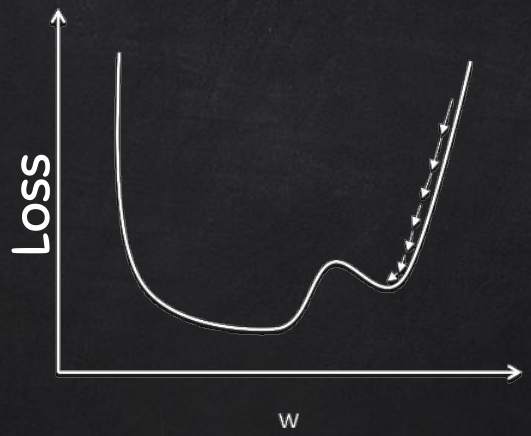
...

N. Profit???

Learning Rate

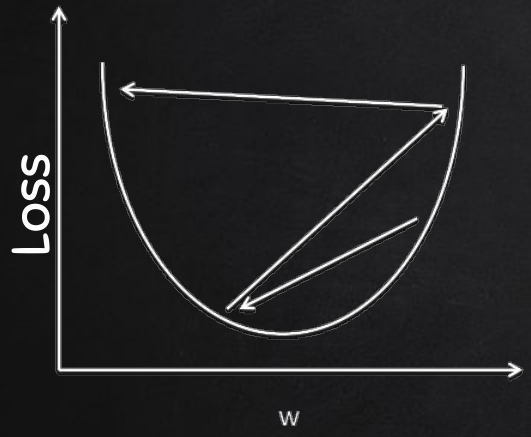


Large learning rate: Overshooting.

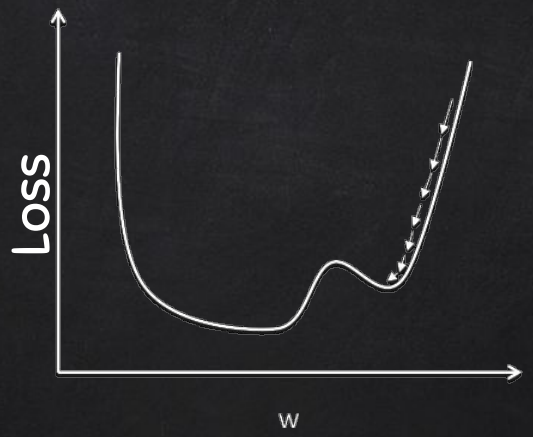


Small learning rate: Many iterations until convergence and trapping in local minima.

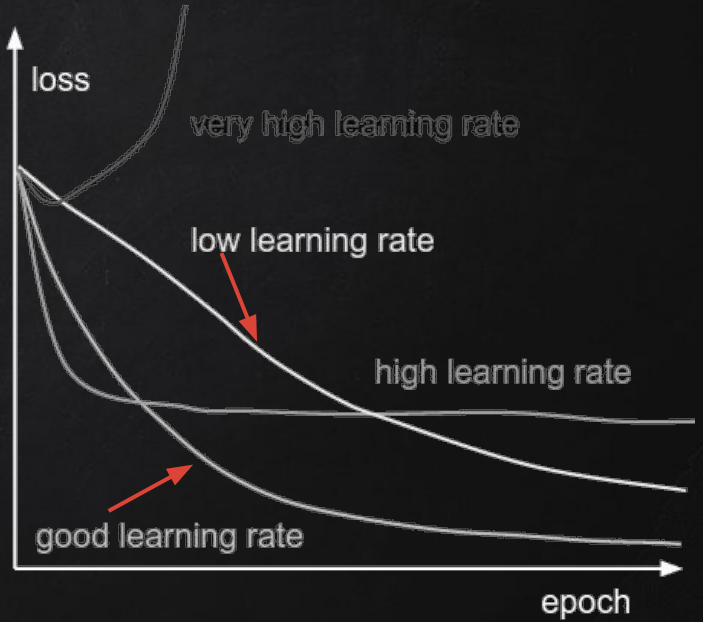
Learning Rate



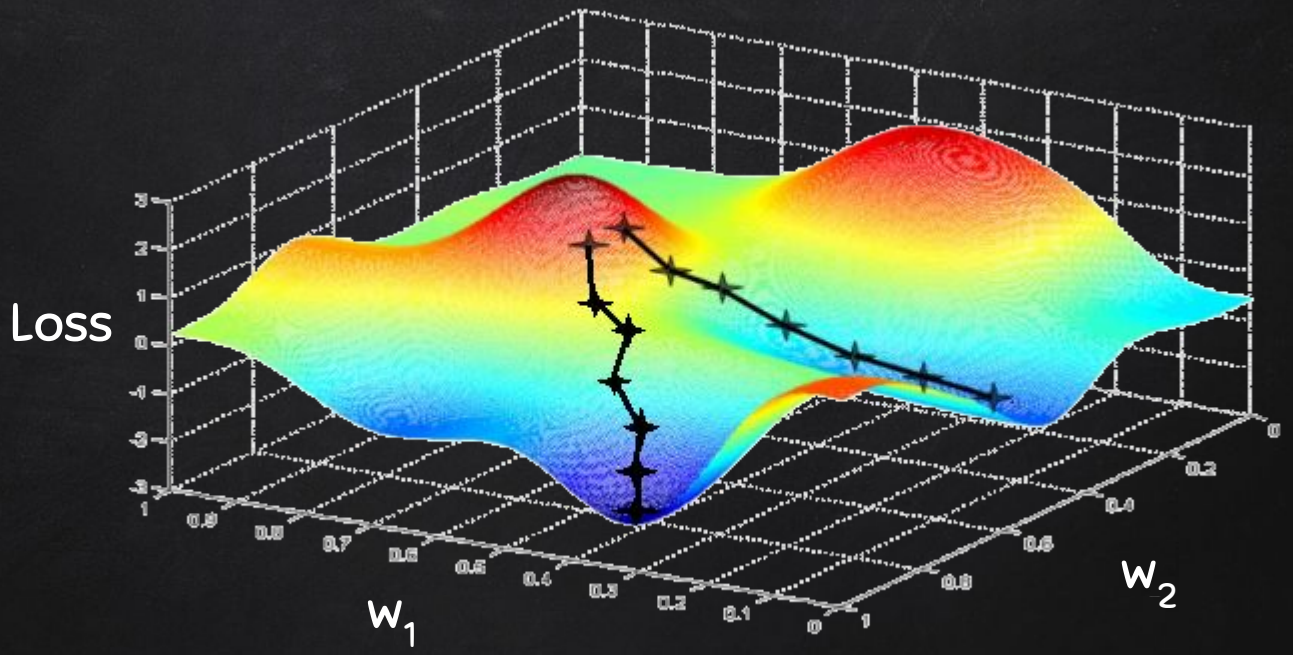
Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.



Gradient Descent



Gradient Descent

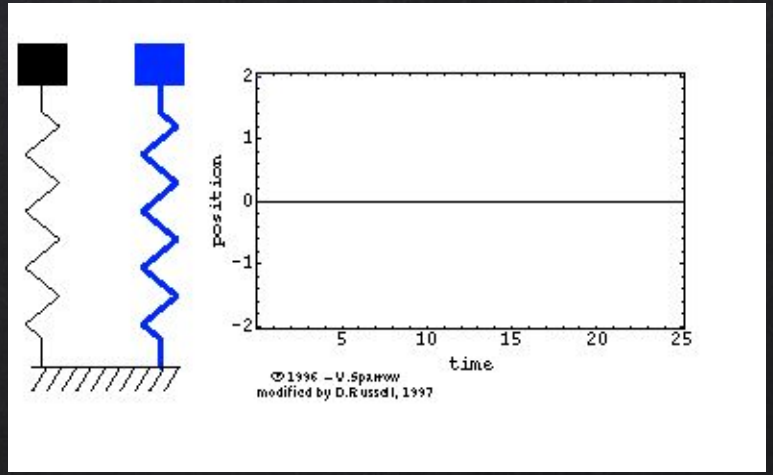
$$\Delta W = \eta \frac{1}{N} \sum \delta W$$

Stochastic Gradient Descent

Mini-batch weight update #1

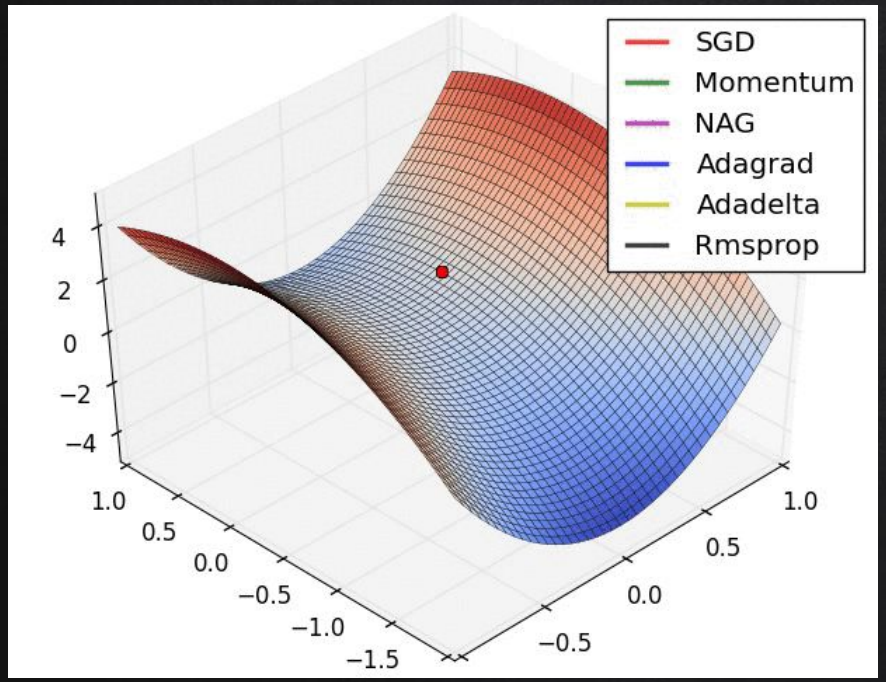
Mini-batch weight update #2

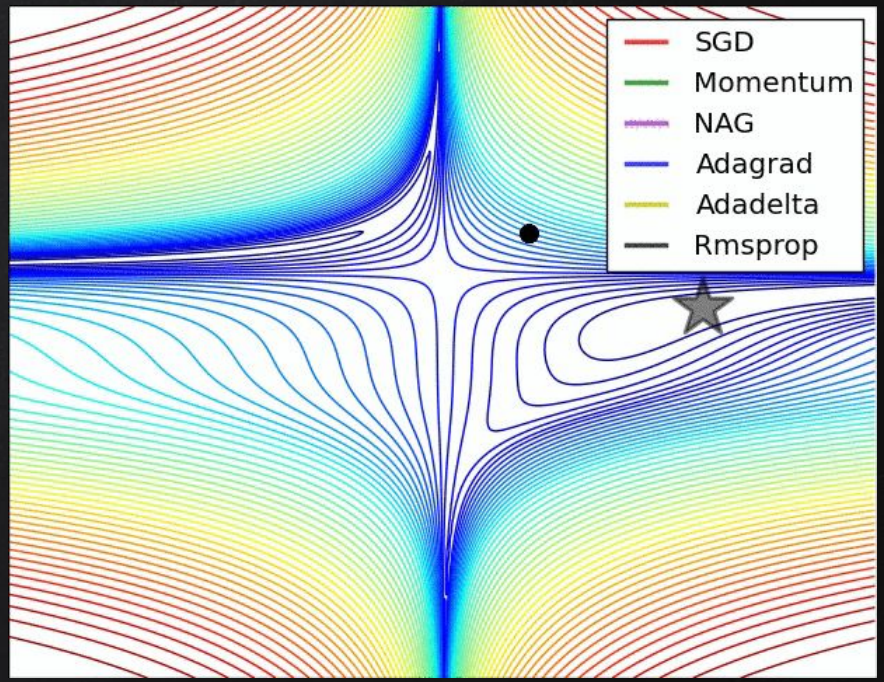
Stochastic Gradient Descent with Momentum



Demo: Why Momentum really works?

Stochastic Gradient Descent





Credit: Sebastian Rudder

Choices

Initializations

- Uniform
- Gaussian
- TrunNorm
- GlorotUni
- Xavier
- Kaiming

Activation

- TanH
- Sigmoid
- ReLU
- PreLU
- LReLU
- Swish

Cost

- Cross entropy
- MSE
- L1 loss
- KL Div
- JS Div

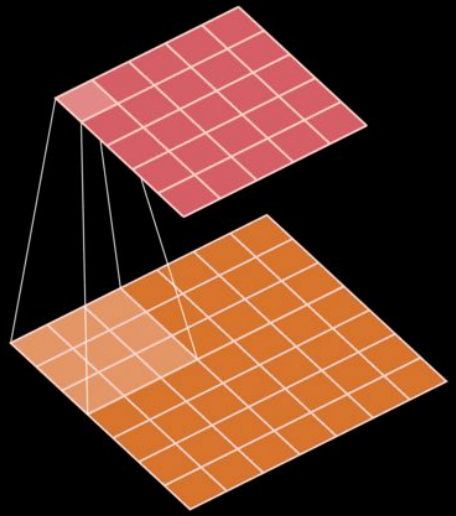
Optimizers

- SGD
- Momentum
- RMSProp
- ADAGRAD
- ADADELTA
- Adam

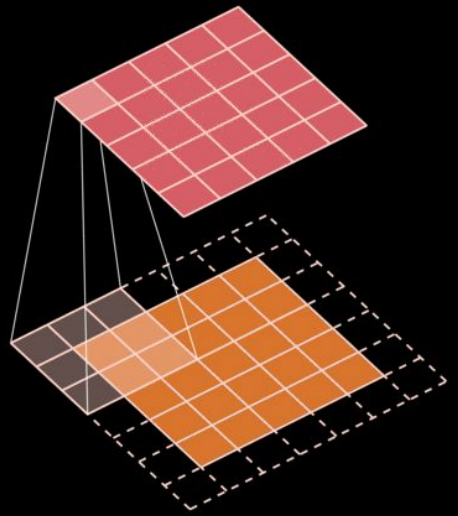
Convolution Networks

Demo Convolution

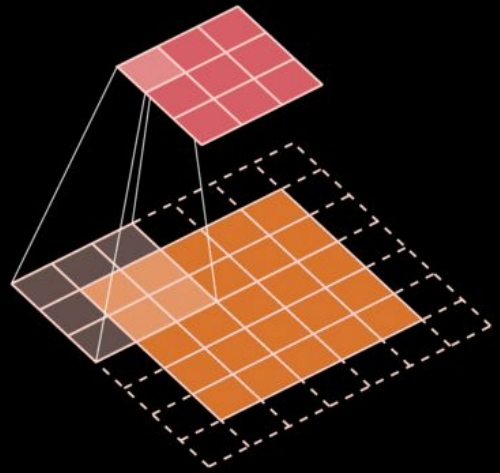
Convolution



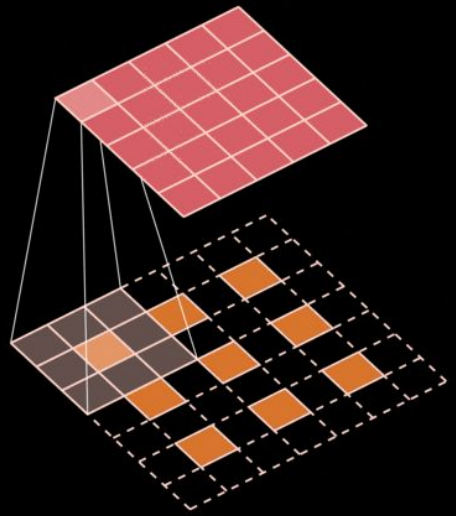
Convolution with Padding



Convolution with Padding + Strides



Deconvolution / Convolution Transposed



Max Pool

0	1	4	9
3	2	5	8
1	2	3	1
3	1	7	4

Size: 2x2, Stride: 2

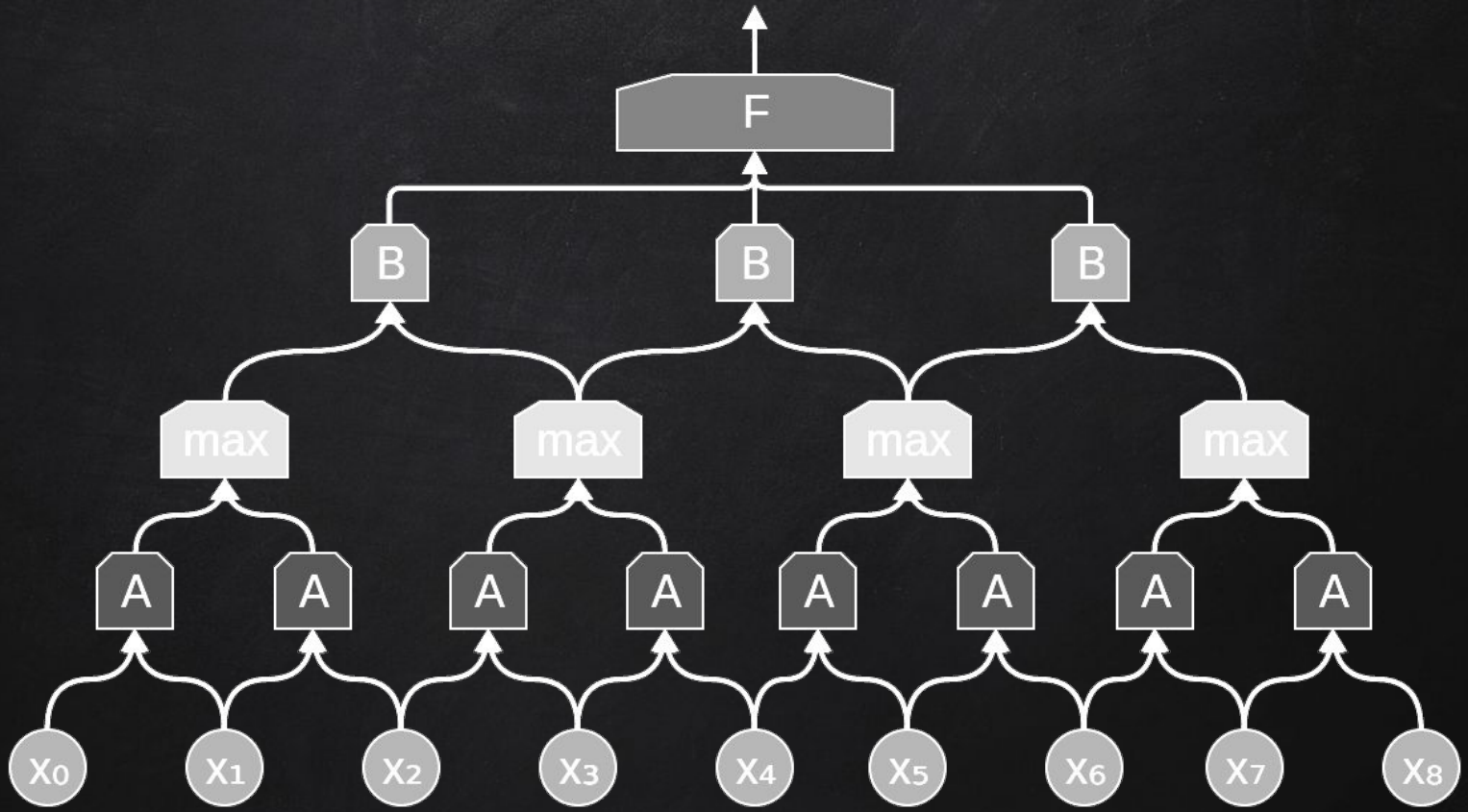
3	9
3	7

Geoff Hinton on Pooling----

The pooling operation used in convolutional neural networks is a **big mistake** and the fact that it works so well is a disaster.

If the pools do not overlap, pooling loses valuable information about where things are. We need this information to detect precise relationships between the parts of an object.

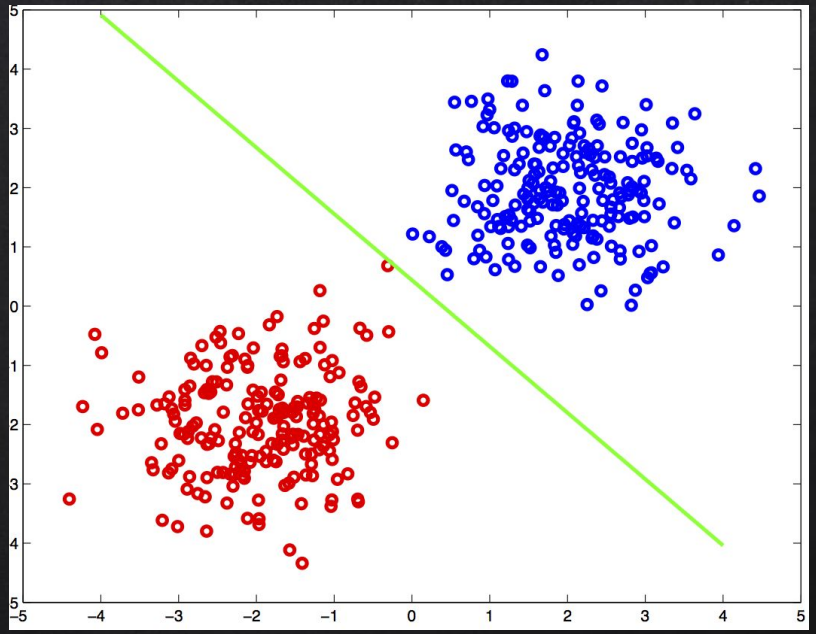
Convolutional Neural Network

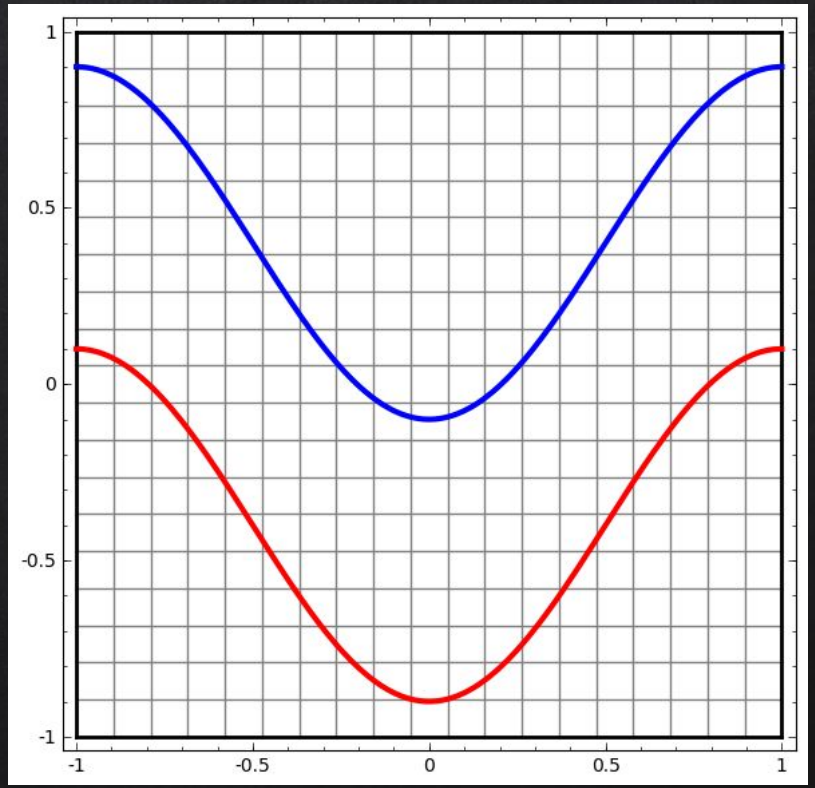


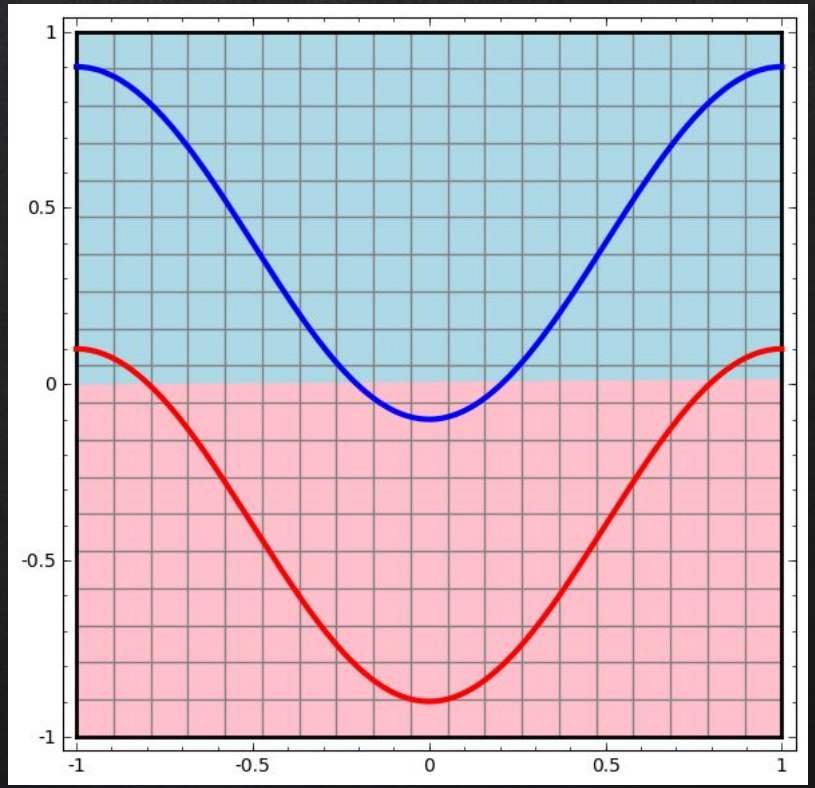
But what do the layers do?

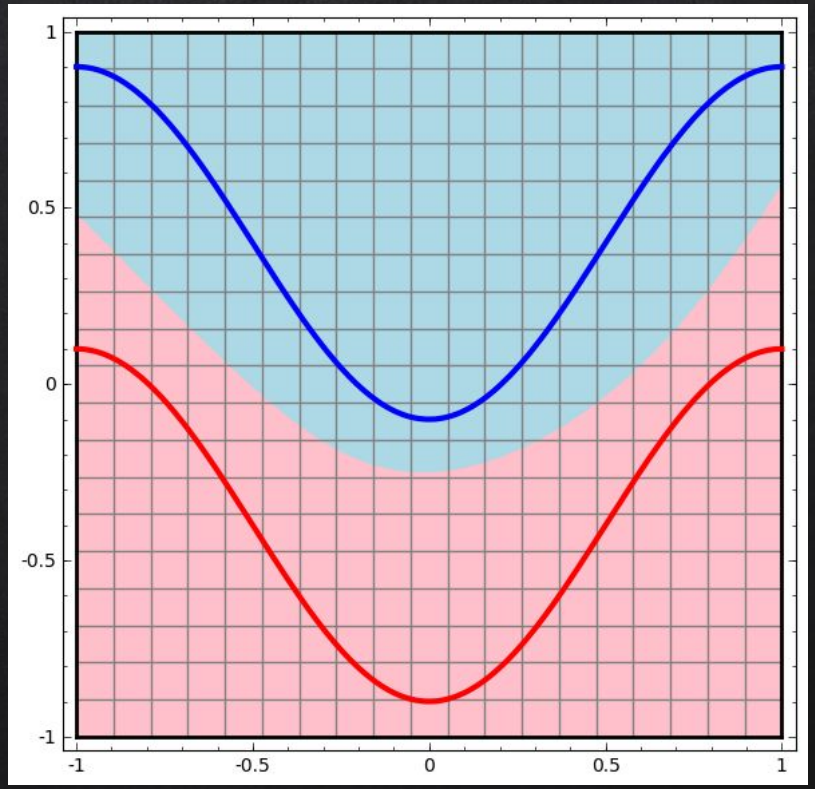
How many layers do I need?

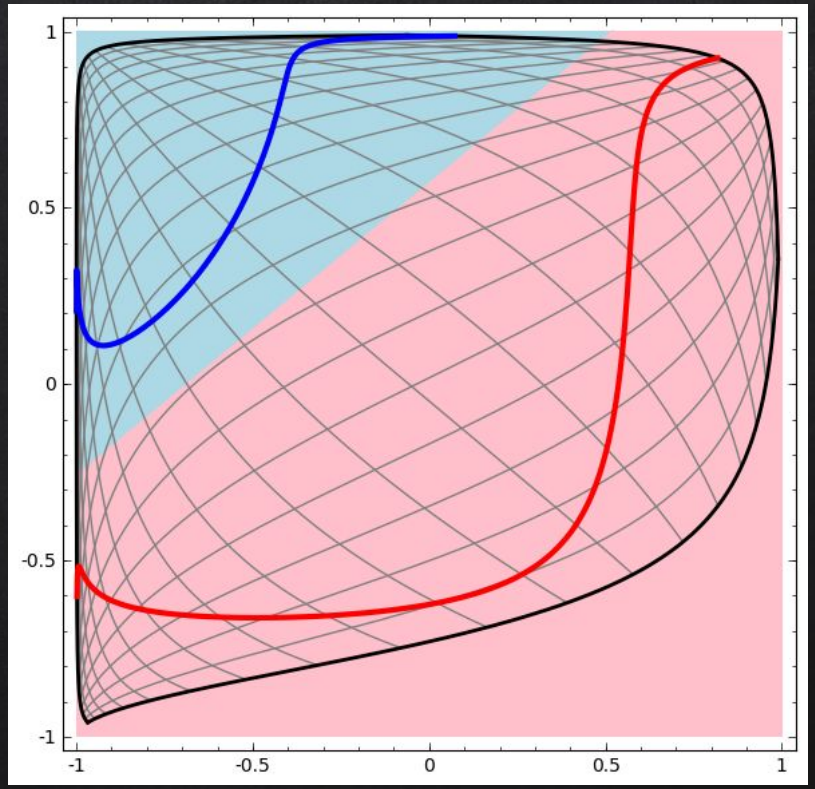
Separating Data

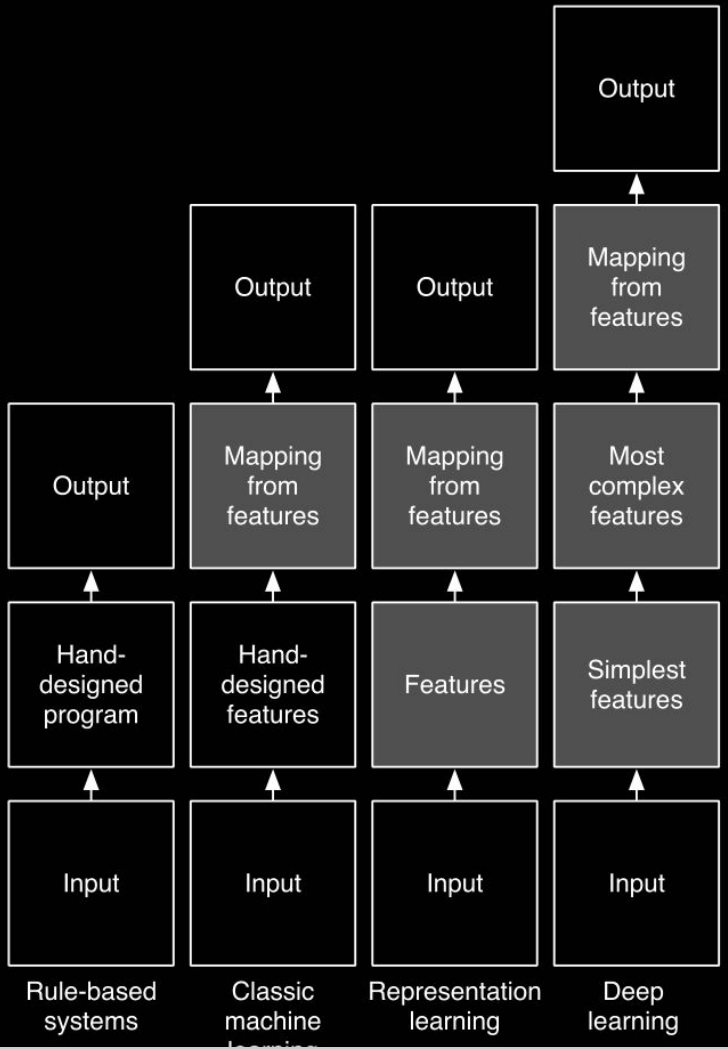



1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


Output
Output
Output
Mapping from features
Output
Mapping from features
Mapping from features
Most complex features
Hand-designed program
Hand-designed features
Features
Simplest features
Input
Input
Input
Input
Rule-based systems
Classic machine
Representation learning
Deep learning

Selfie !

<= OR =>

Any questions?

You can find me at
iamaaditya.github.io
aprakash@brandeis.edu

THANKS!

Credits

- Presentation template by SlidesCarnival
- Awesome diagrams on CNN by http://colah.github.io/
- Formulas from, you know where, wikipedia
- Selfie http://karpathy.github.io/2015/10/25/selfie/
- Neural Networks and Deep Learning
- Going Deeper - GoogLeNet
- Back propagation images with formulas, Mariusz Bernacki
- Geoff Hinton, guy who started all this -- NIPS 2015 tutorial
- Yann LeCun, guy who made it popular, Deep Representations
- CS231n Lecture Slides
- Nervana presentation on DL