Convolution Neural Network

Presentation Covers

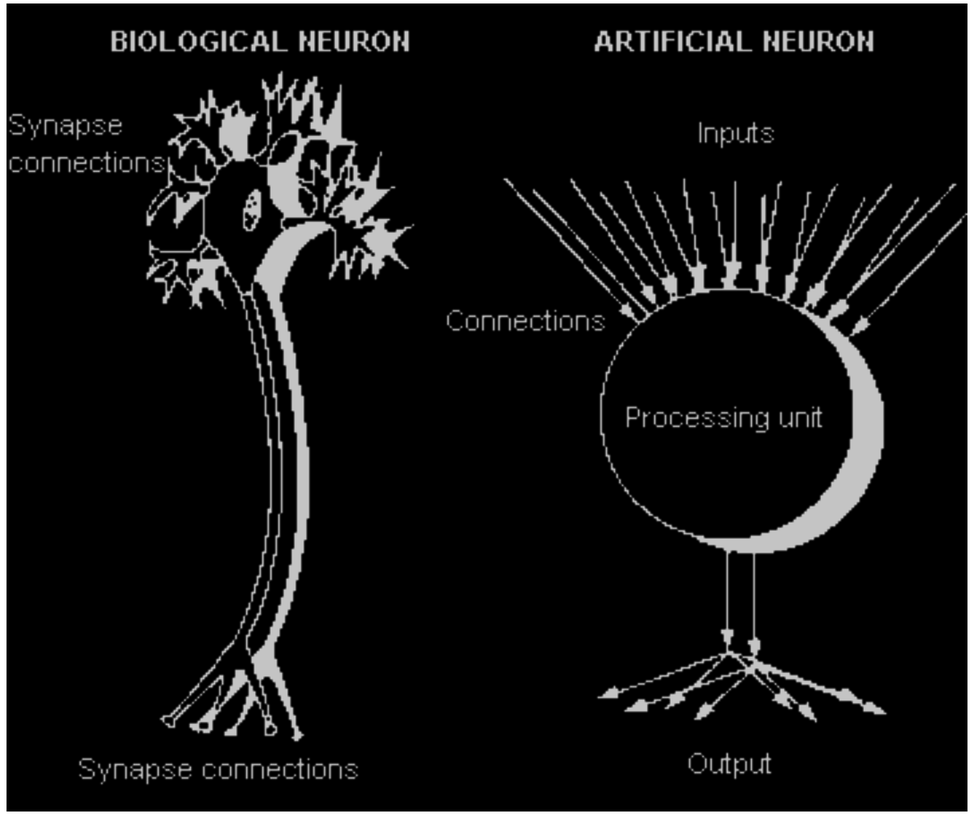
- The basic learning entity in the network - Perceptron
- Important Aspects of Deep Learning

By-Amit Kushwaha

Who am I

Machine Learning Engineer
@

Artificial Neural Net (ANN)



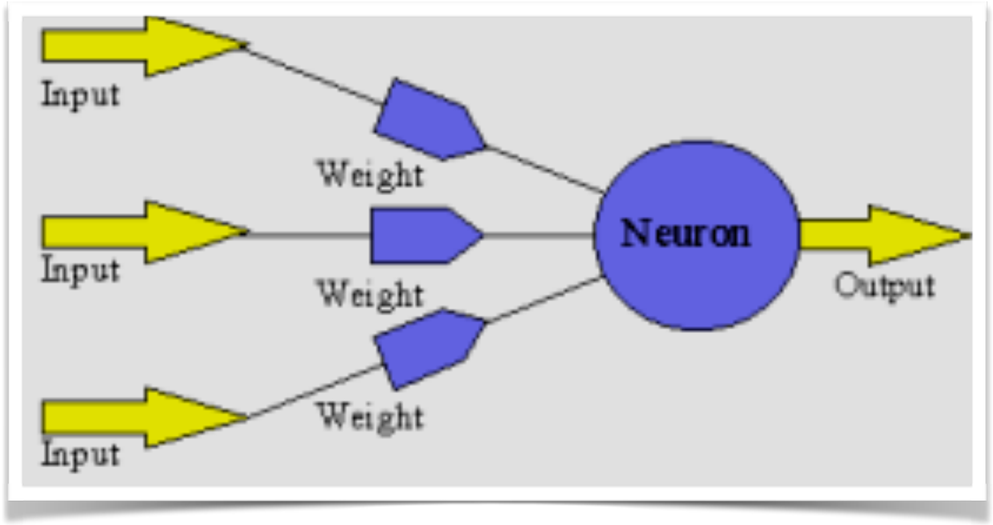
a Biological Inspiration

Perceptron

The elementary entity of a neural network.

The most basic form of neural network which can also learn

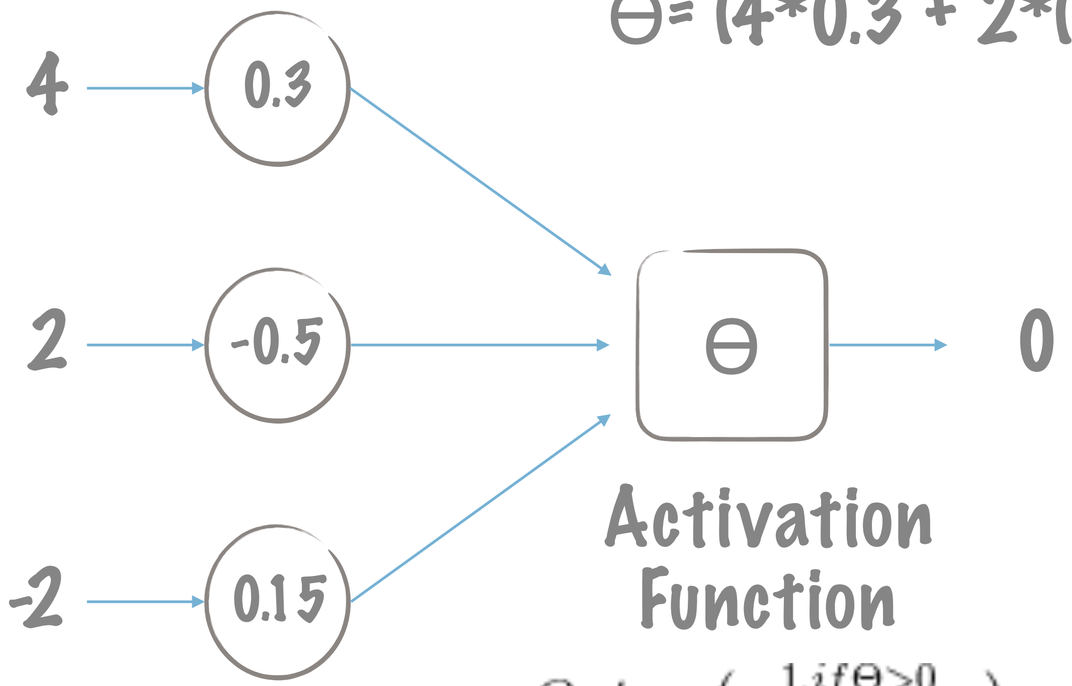
Perceptron



A Basic Linear discriminant Classifier

How Perceptron Work

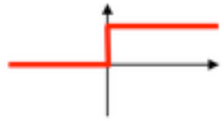
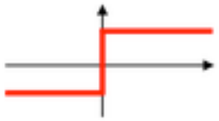
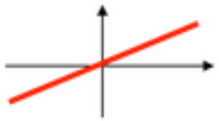
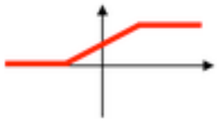
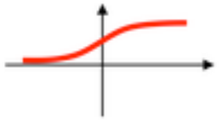
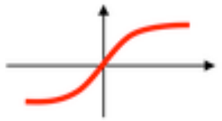
$\Theta = (4*0.3 + 2*(-0.5) + (-2*0.15)$



Activation Function

$$Out = \begin{pmatrix} 1, if \Theta > 0 \\ 0, Otherwise \end{pmatrix}$$

Activation Functions

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

Training a Perceptron

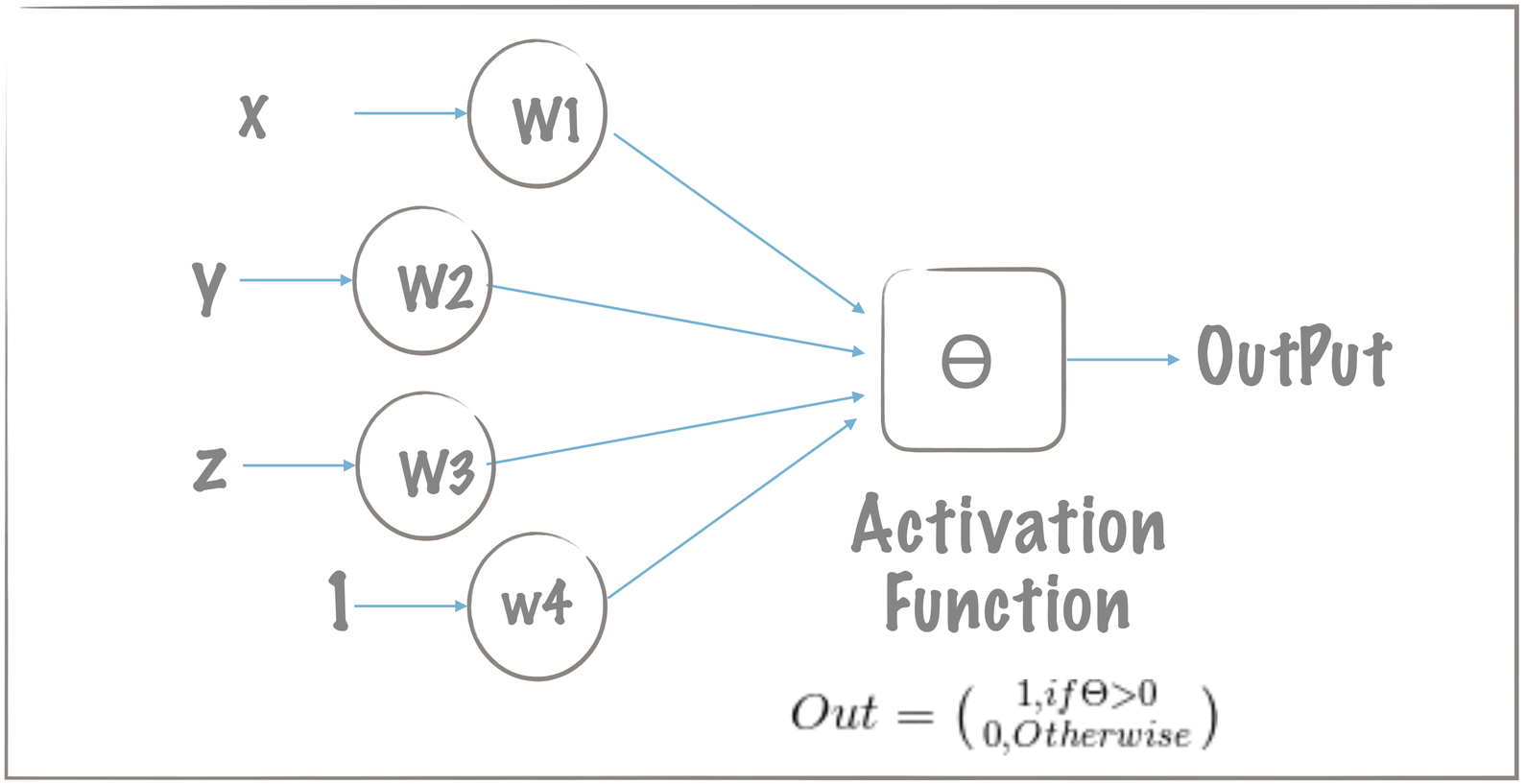
Let's train a Perceptron to mimic this pattern

Training Set T(out)

x	y	z	out
0	0	1	0
1	1	1	1
1	0	1	1
0	1	1	0

Training a Perceptron

Perceptron Model

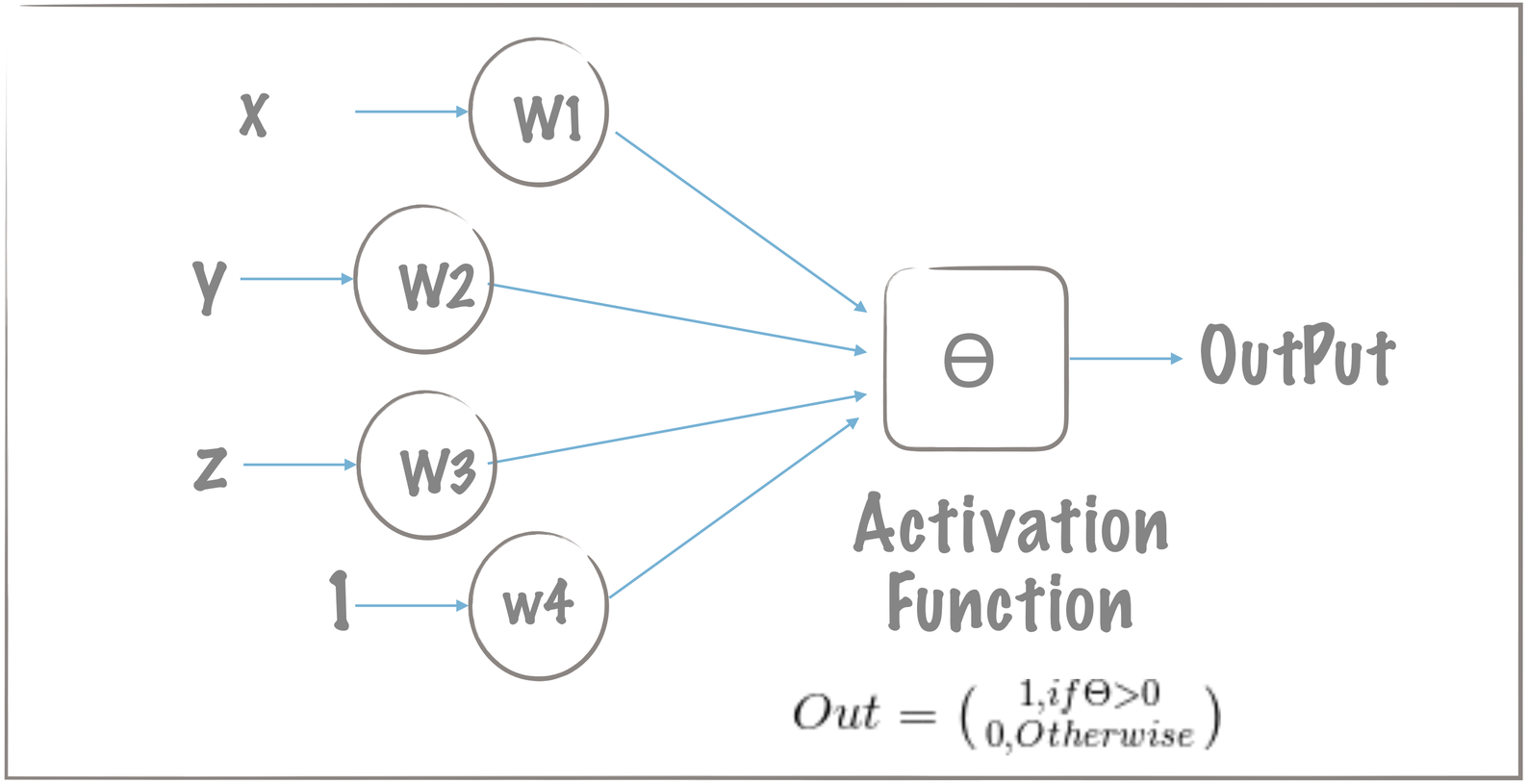


Training a Perceptron

Training Rules : $W_i = W_i + \Delta W_i$

$\Delta W = -\eta * (\text{target output} - \text{perceptron output}) * X$

η is learning rate for perceptron



Training a Perceptron

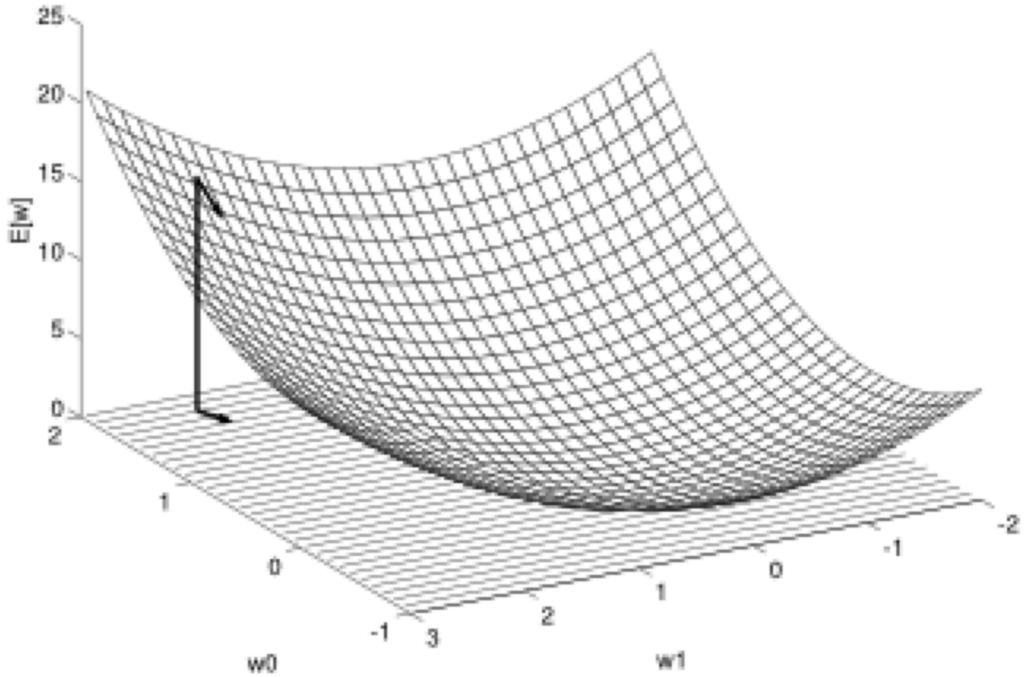
Let's learn w_i such that it minimizes the squared error

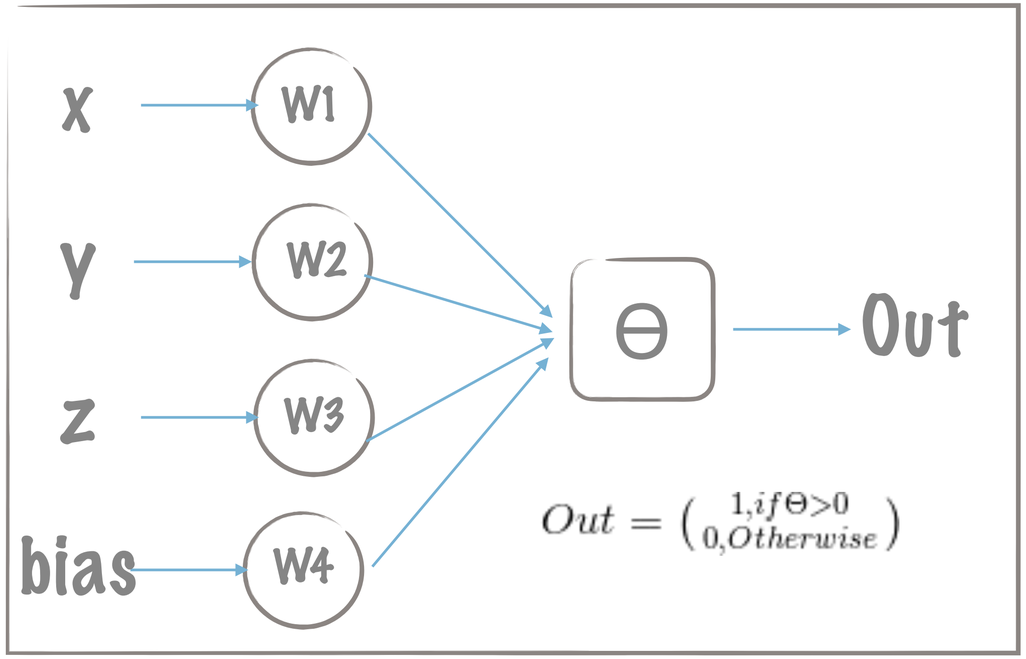
$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

On simplifying

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$



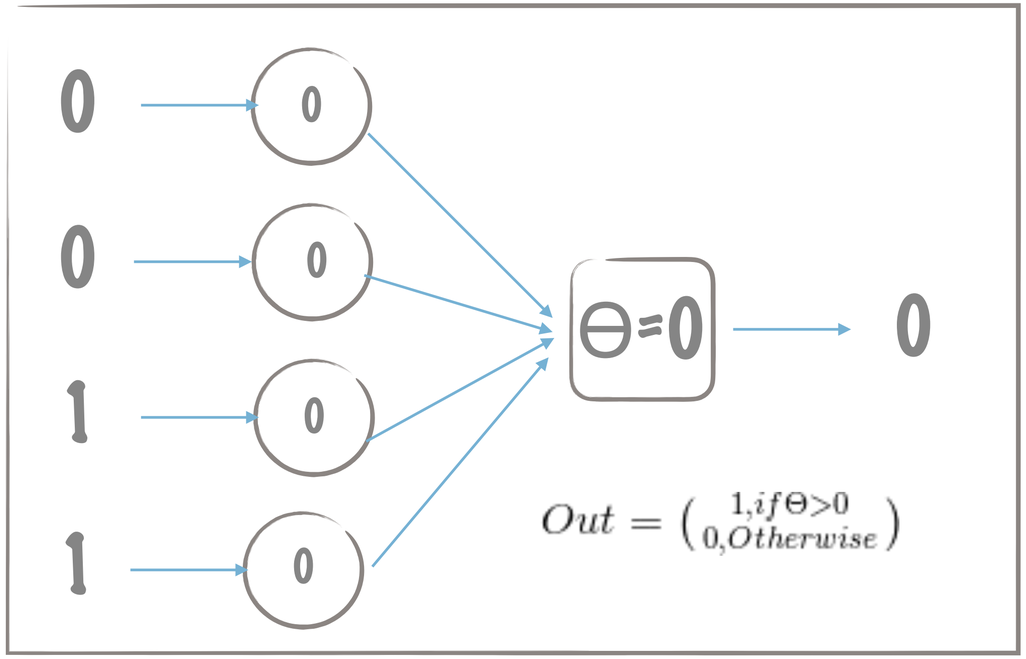


$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set T(out)

	x	y	z	bias	out
epoch 1	0	0	1	1	0
	1	1	1	1	1
	1	0	1	1	1
	0	1	1	1	0
epoch 2	0	0	1	1	0
	1	1	1	1	1
	1	0	1	1	1
	0	1	1	1	0

Assign random values to Weight Vector([W1,W2,W3,W4])



$$Wi = Wi + \Delta Wi$$
$$\Delta Wi = -\eta * [P(out) - T(Out)] * xi$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
	0	0	1	1
epoch 1	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch 2	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

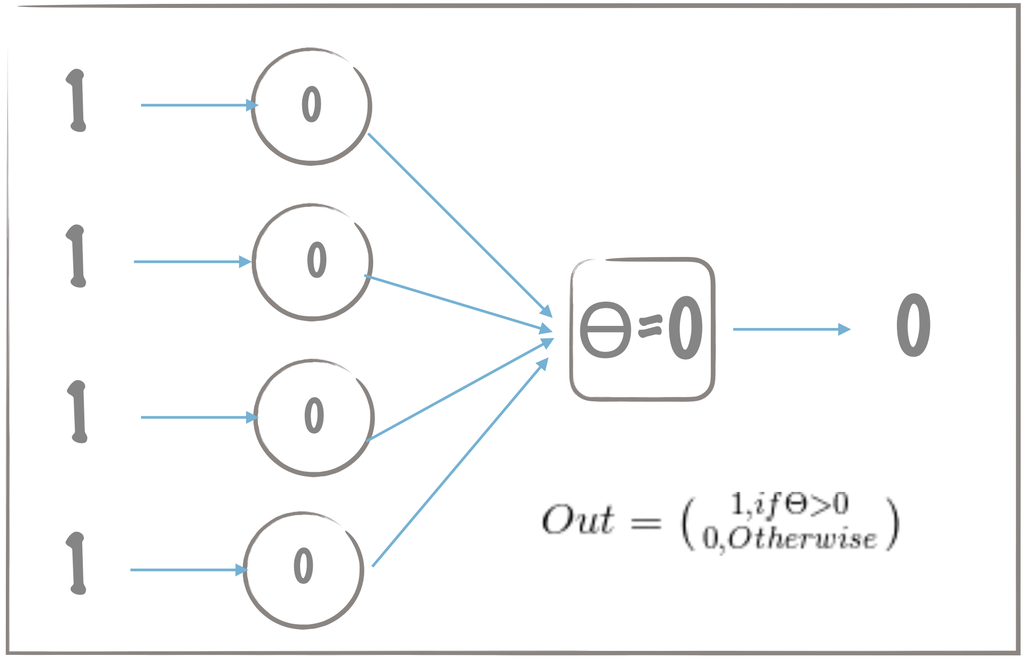
w1	w2	w3	w4
0	0	0	0

P(out)

out
0

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch 1	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch 2	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

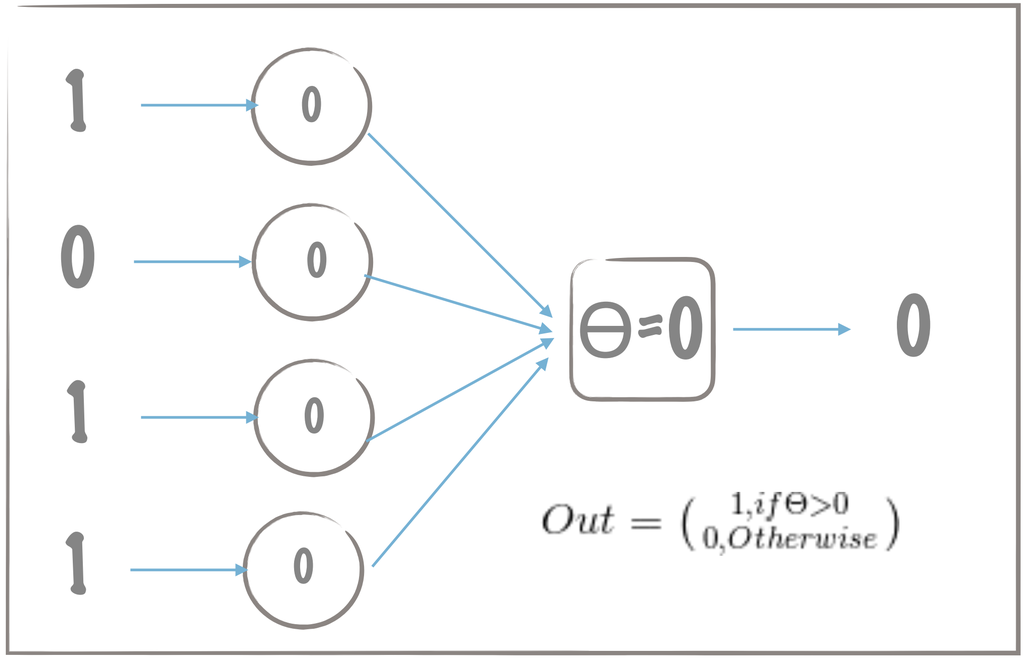
w1	w2	w3	w4
0	0	0	0
0	0	0	0

P(out)

out
0
0

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0
1	1	1	1



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch 1	0	0	1	1
	1	1	1	1
	1	0	1	1
epoch 2	0	1	1	1
	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

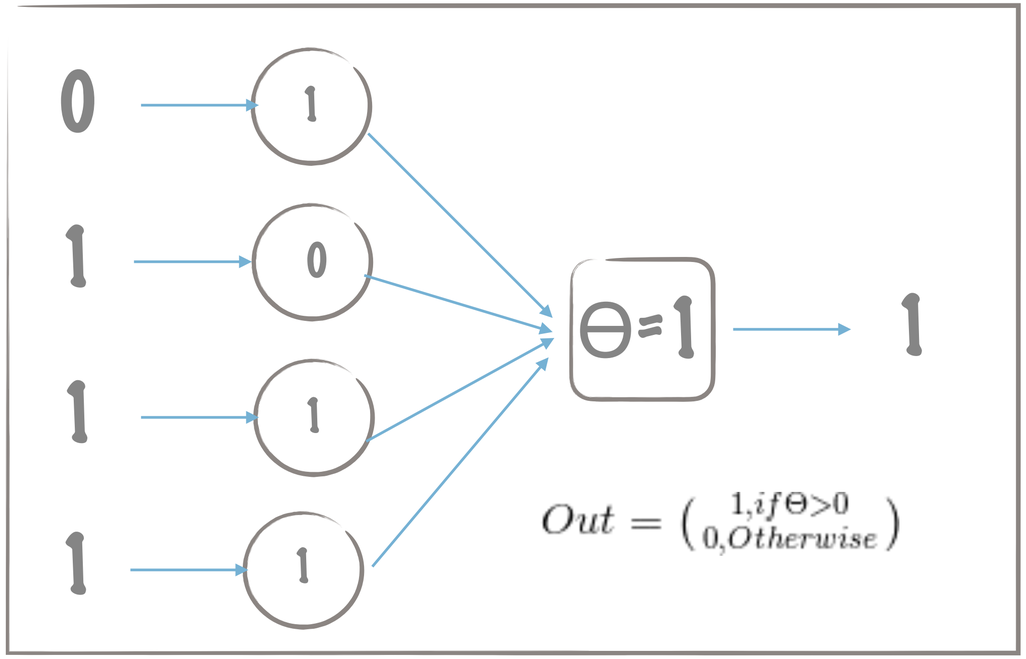
w1	w2	w3	w4
0	0	0	0
0	0	0	0
1	1	1	1

P(out)

out
0
0
1

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0
1	1	1	1
0	0	0	0



$$Wi = Wi + \Delta Wi$$
$$\Delta Wi = -\eta * [P(out) - T(Out)] * xi$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch 1	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch 2	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

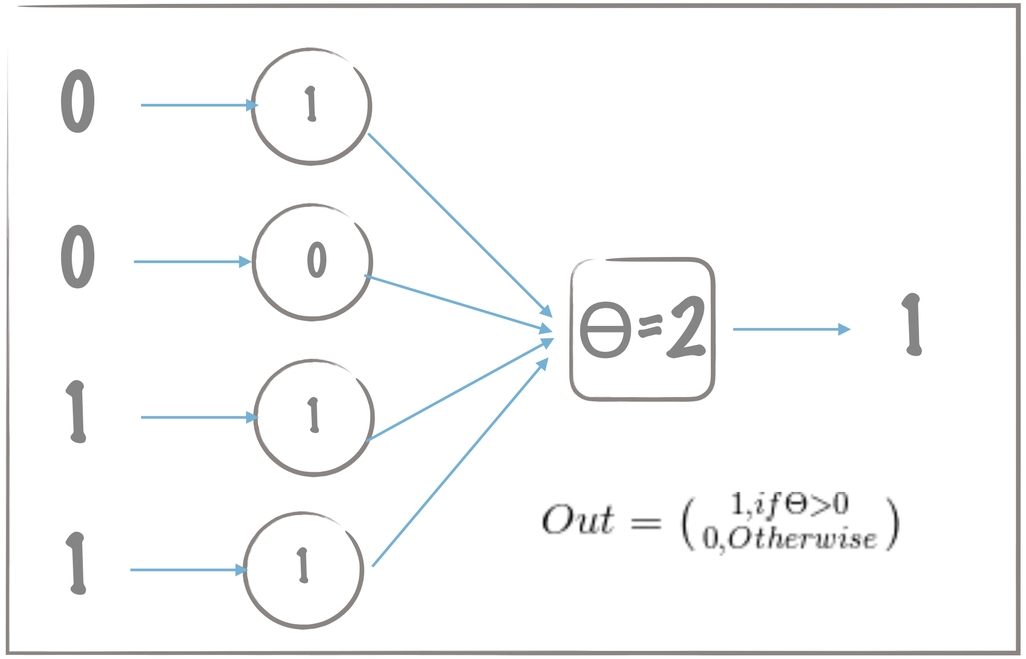
w1	w2	w3	w4
0	0	0	0
0	0	0	0
1	1	1	1
1	1	1	1

P(out)

out
0
0
1
1

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0
1	1	1	1
0	0	0	0
0	-1	-1	-1



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch 1	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch 2	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

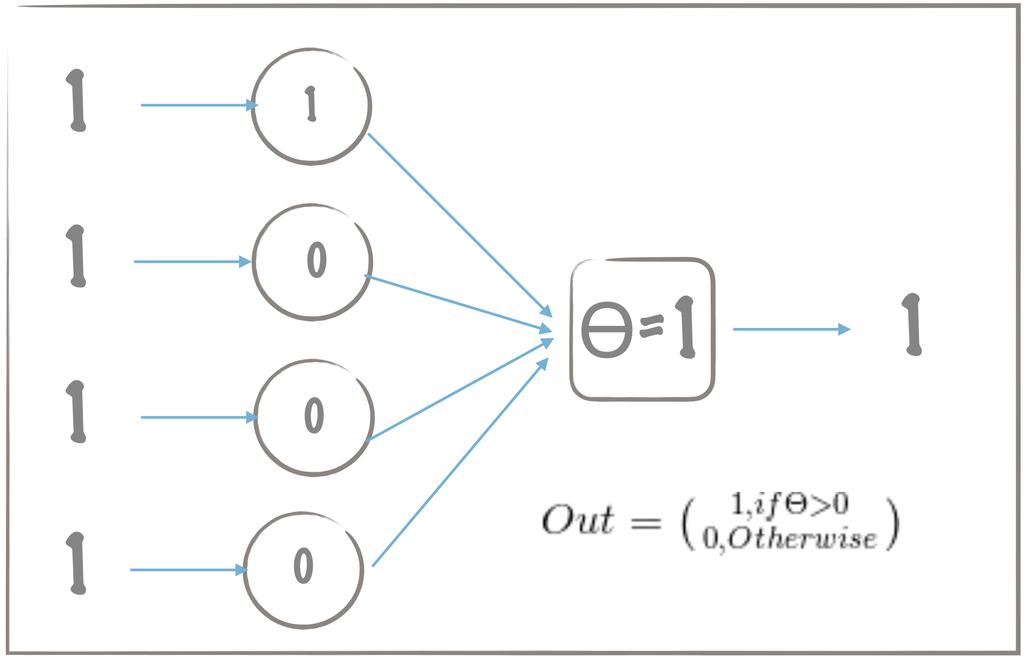
w1	w2	w3	w4
0	0	0	0
0	0	0	0
1	1	1	1
1	1	1	1
1	0	0	0

P(out)

out
0
0
1
1
0

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0
1	1	1	1
0	0	0	0
0	-1	-1	-1
0	0	0	0



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch 1	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch 2	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

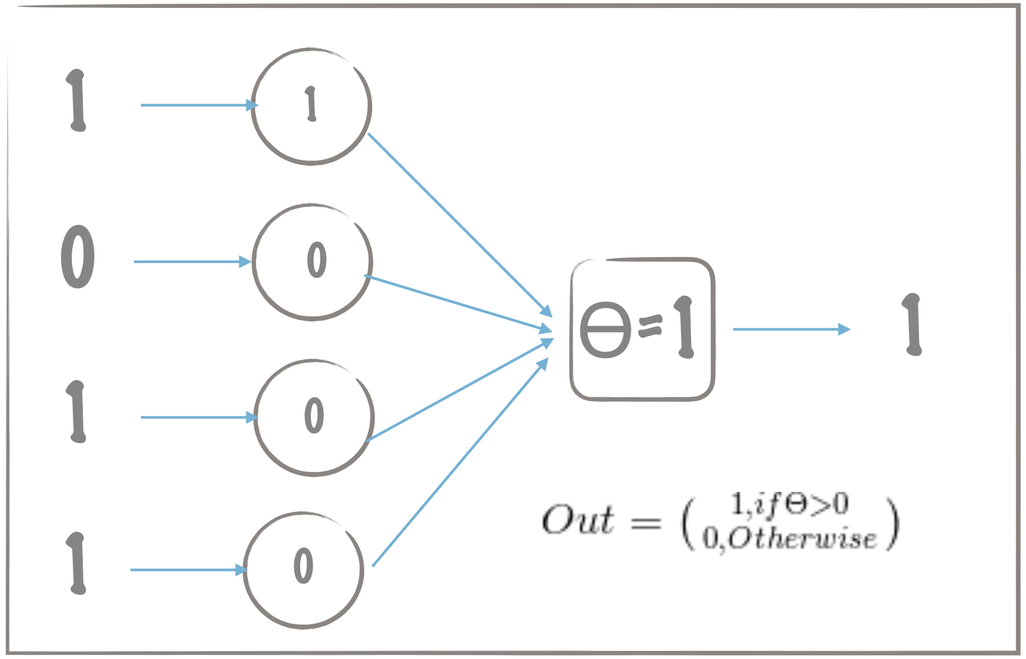
w1	w2	w3	w4
0	0	0	0
0	0	0	0
1	1	1	1
1	1	1	1
1	0	0	0
1	0	0	0

P(out)

out
0
0
1
1
1
1

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0
1	1	1	1
0	0	0	0
0	-1	-1	-1
0	0	0	0
0	0	0	0



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

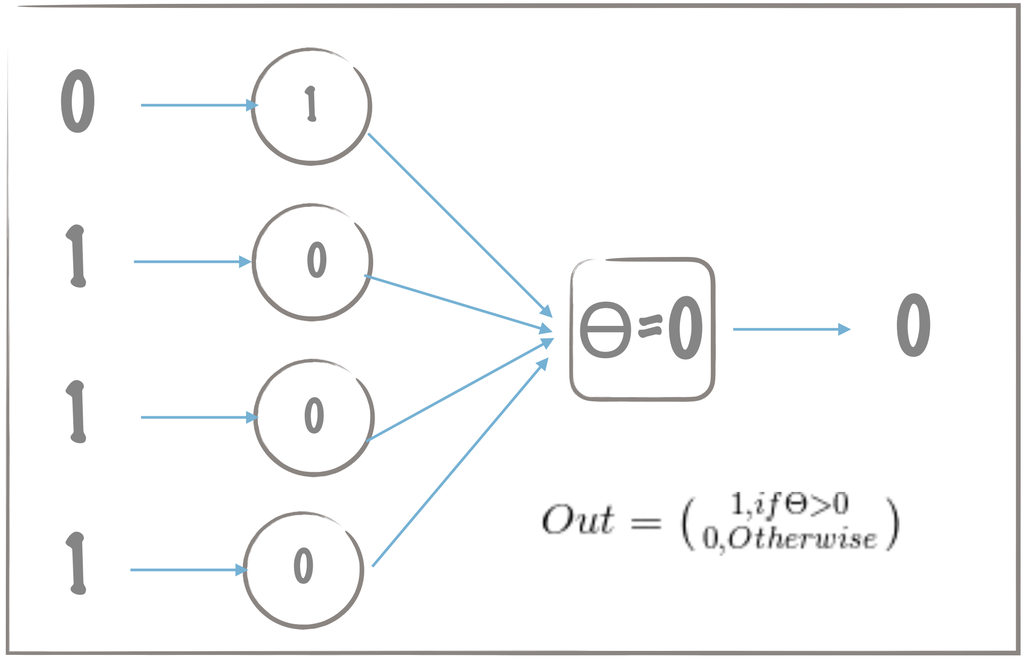
T(out)

Weights

P(out)

ΔWeights

	x	y	z	bias	out	w1	w2	w3	w4	out	Δw1	Δw2	Δw3	Δw4
epoch 1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	0	0	0	0	0	1	1	1	1
	1	0	1	1	1	1	1	1	1	1	0	0	0	0
	0	1	1	1	0	1	1	1	1	1	0	-1	-1	-1
epoch 2	0	0	1	1	0	1	0	0	0	1	0	0	0	0
	1	1	1	1	1	1	0	0	0	1	0	0	0	0
	1	0	1	1	1	1	0	0	0	1	0	0	0	0
	0	1	1	1	0									



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch 1	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch 2	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

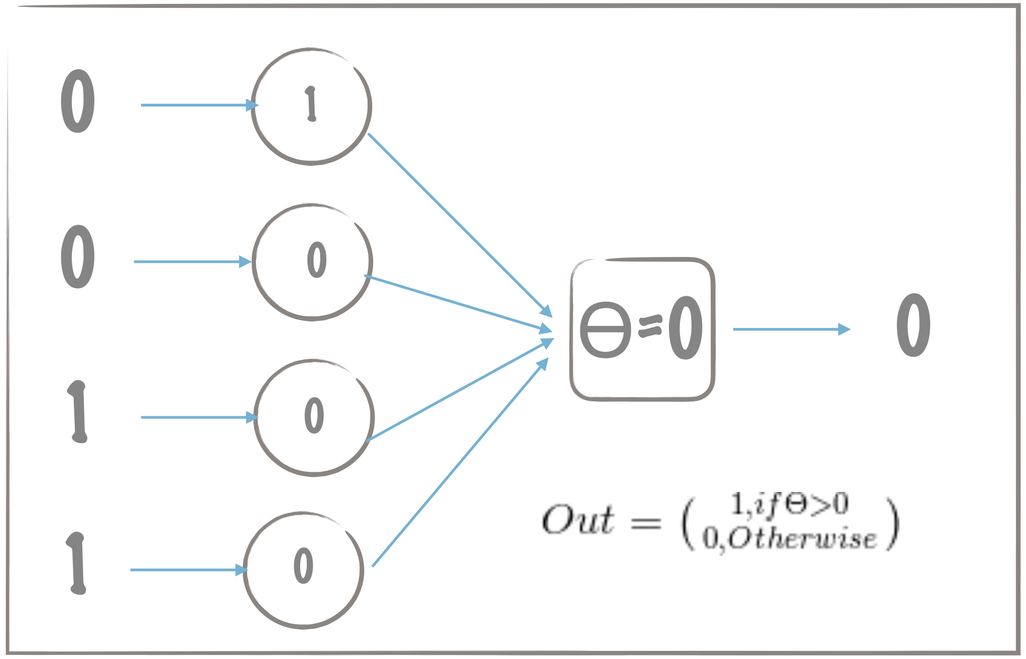
w1	w2	w3	w4
0	0	0	0
0	0	0	0
1	1	1	1
1	1	1	1
1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0

P(out)

out
0
0
1
1
1
1
1
0

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0
1	1	1	1
0	0	0	0
0	-1	-1	-1
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
	0	0	1	1
epoch3	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch4	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

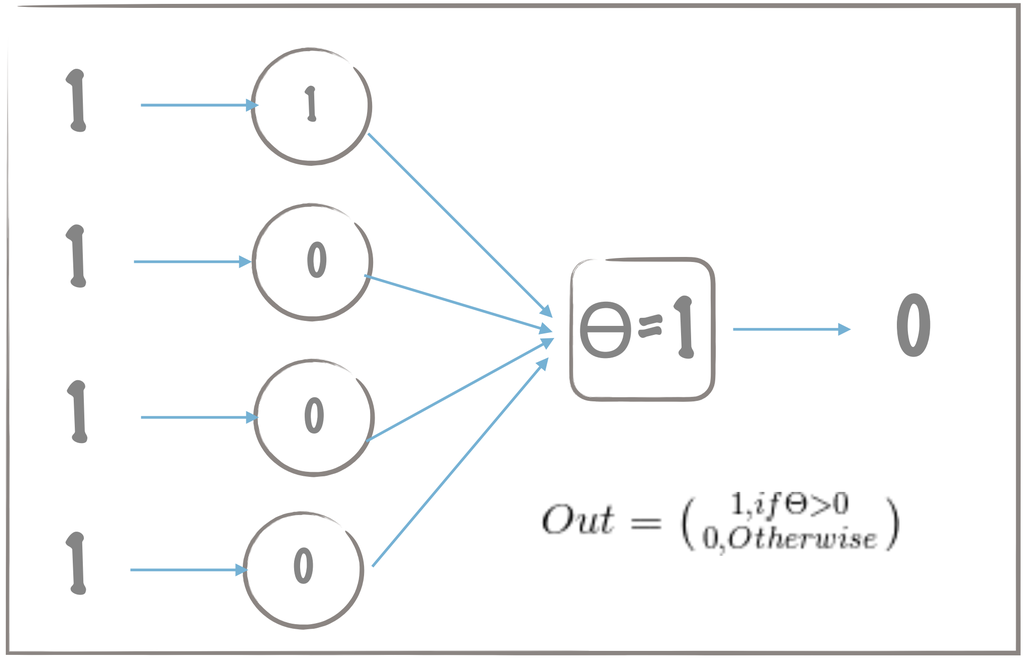
w1	w2	w3	w4
1	0	0	0

P(out)

out
0

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch3	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch4	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

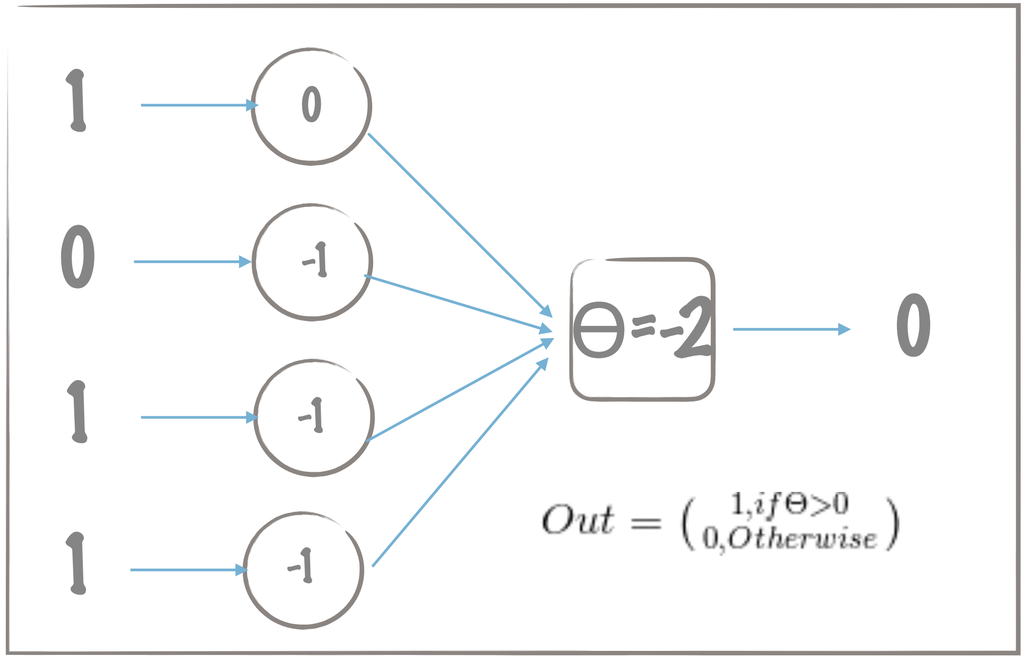
w1	w2	w3	w4
1	0	0	0
1	0	0	0

P(out)

out
0
1

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0
0	0	0	0



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch3	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch4	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

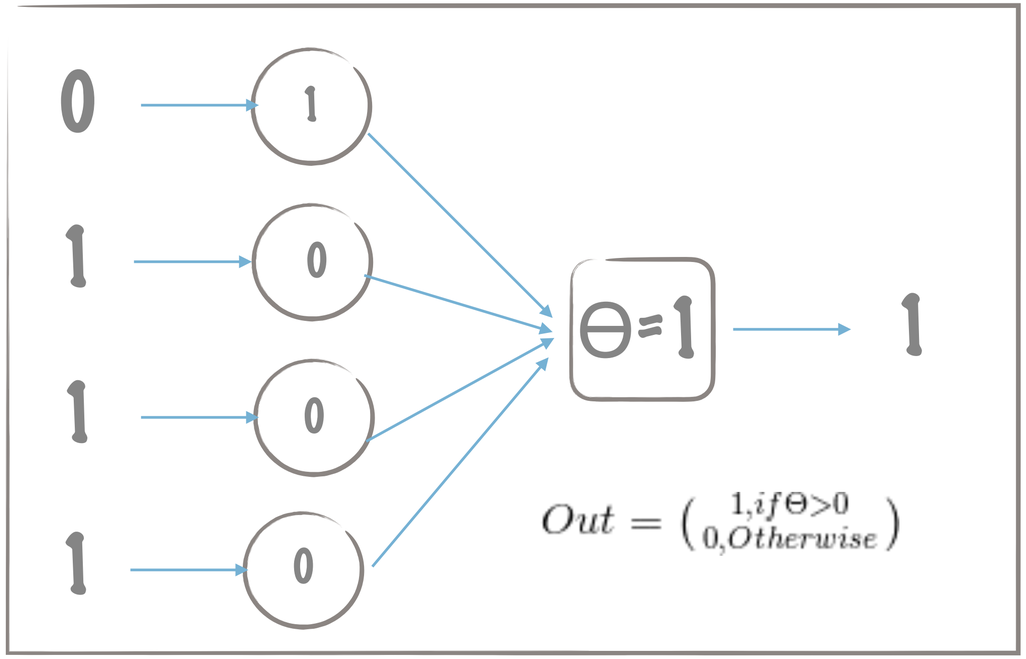
w1	w2	w3	w4
1	0	0	0
1	0	0	0
1	0	0	0

P(out)

out
0
1
1

ΔWeights

Δw1	Δw2	Δw3	Δw4
0	0	0	0
0	0	0	0
0	0	0	0



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

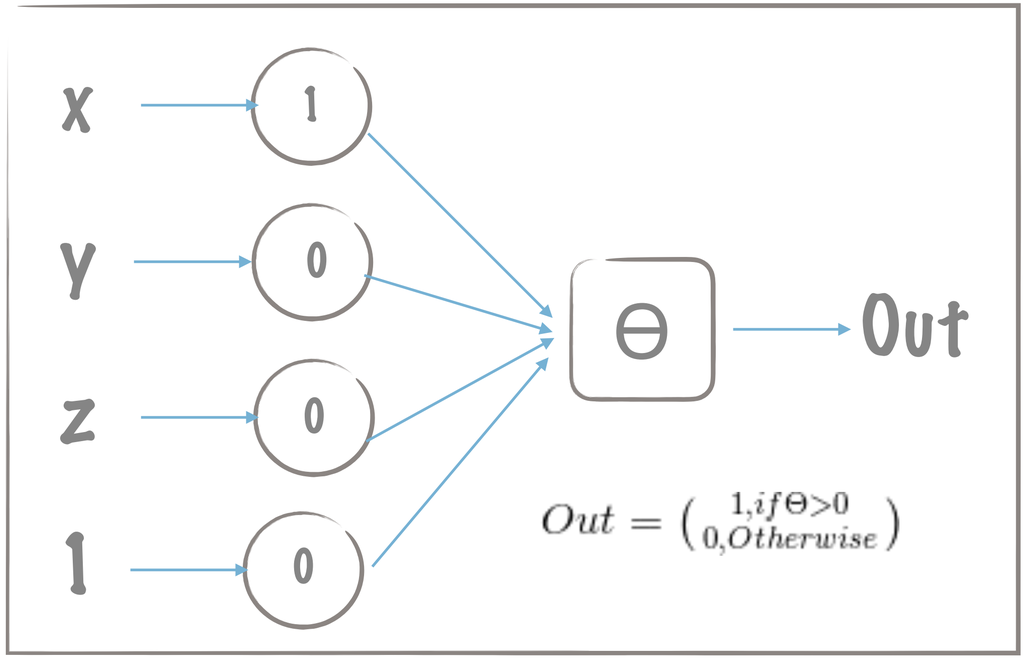
T(out)

Weights

P(out)

ΔWeights

	x	y	z	bias	out	w1	w2	w3	w4	out	Δw1	Δw2	Δw3	Δw4
epoch3	0	0	1	1	0	1	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	0	0	0	1	0	0	0	0
	1	0	1	1	1	1	0	0	0	1	0	0	0	0
	0	1	1	1	0	1	0	0	0	0	0	0	0	0
epoch4	0	0	1	1	0									
	1	1	1	1	1									
	1	0	1	1	1									
	0	1	1	1	0									



$$W_i = W_i + \Delta W_i$$
$$\Delta W_i = -\eta * [P(out) - T(Out)] * x_i$$
η is learning rate for perceptron

Training Set

	x	y	z	bias
epoch3	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1
epoch4	0	0	1	1
	1	1	1	1
	1	0	1	1
	0	1	1	1

T(out)

out
0
1
1
0
0
1
1
0

Weights

w1	w2	w3	w4
1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0

P(out)

out
0
1
1
0

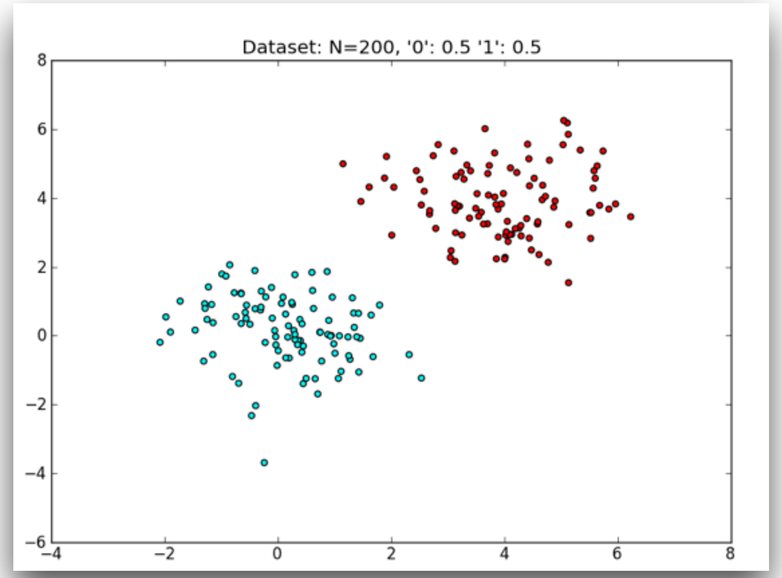
ΔWeights

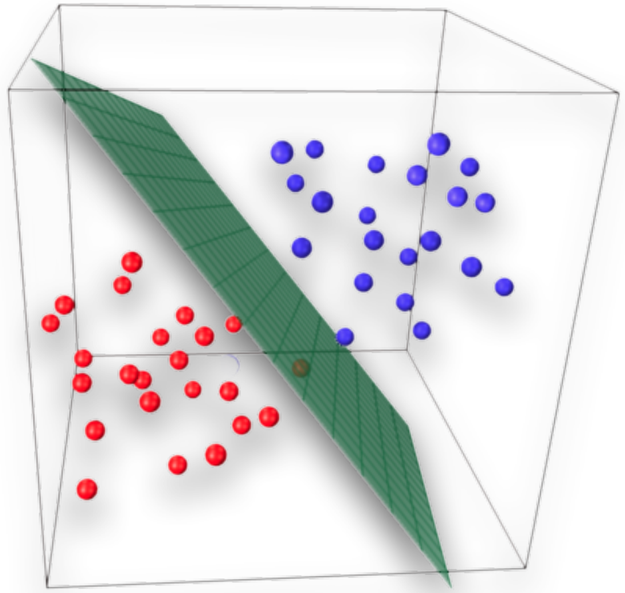
Δw1	Δw2	Δw3	Δw4
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

In epoch3, 'loss' is 0. So, we can assume that Perceptron has learnt the pattern

We trained our Perceptron Model

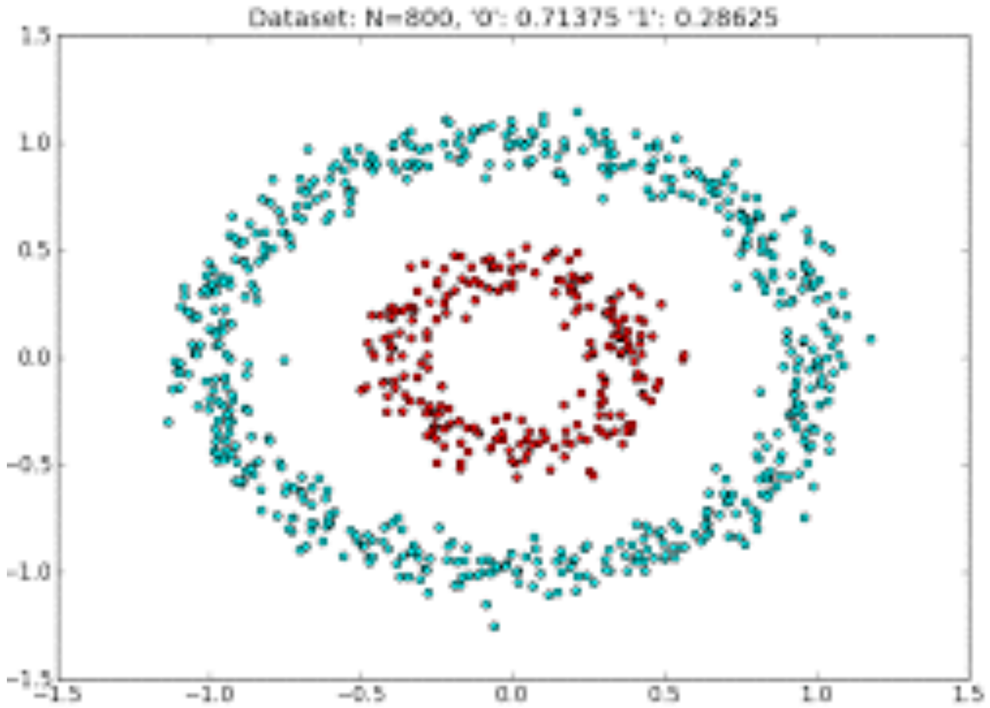
We trained our Perceptron Model





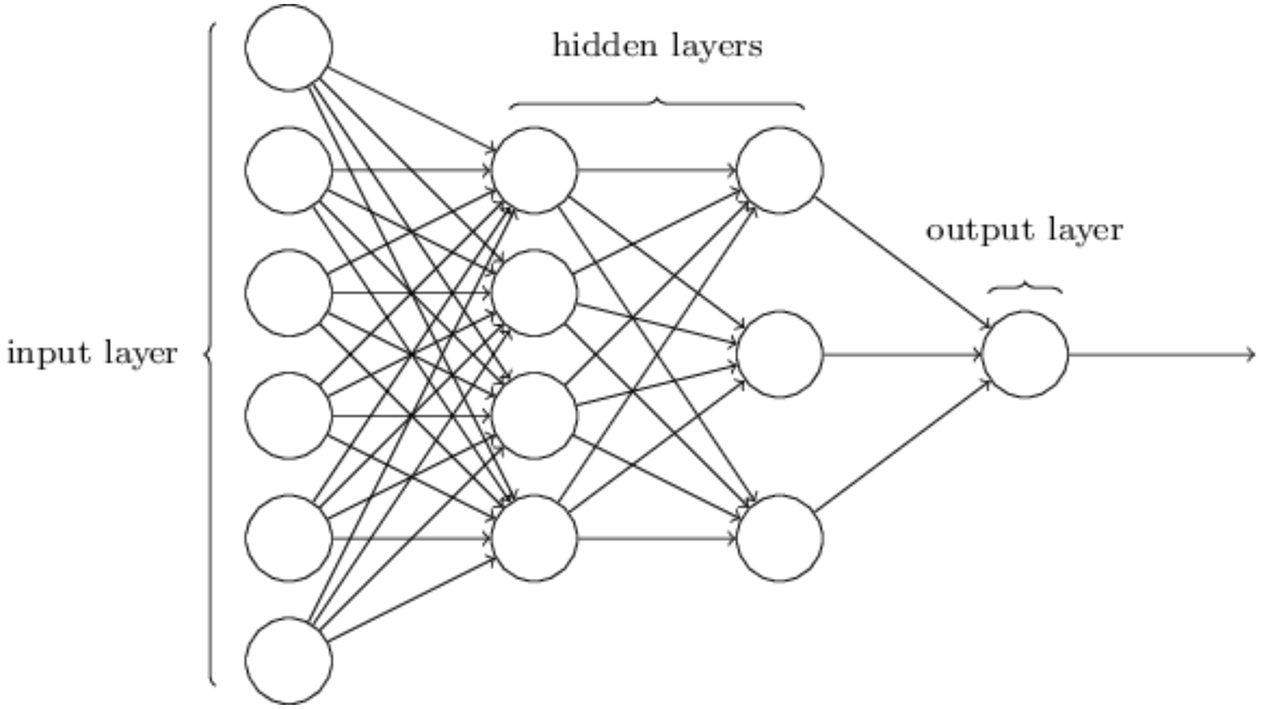
On a Linear Separable Distribution

Limited Functionality of Linear Hyperplane



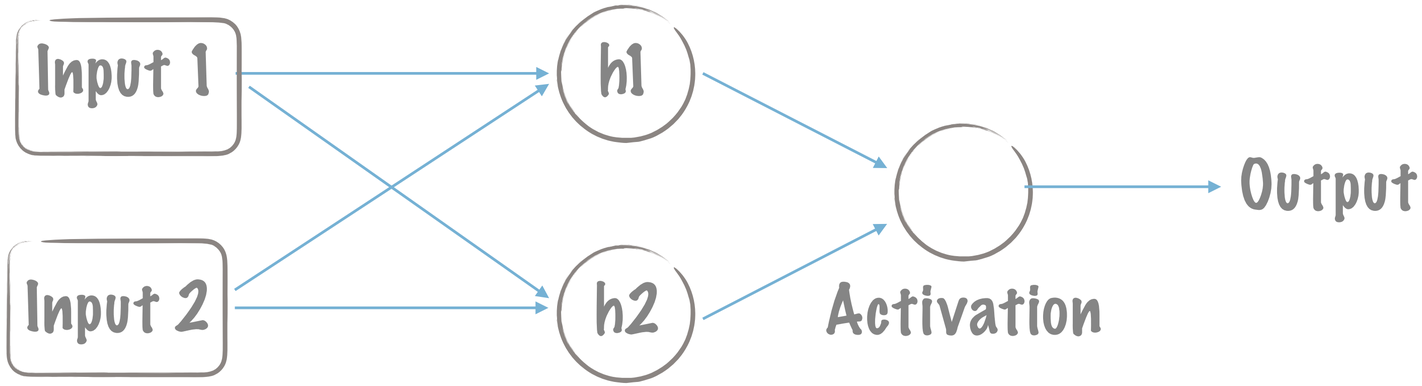
It cannot generalize on data distributed like this

Neural Networks



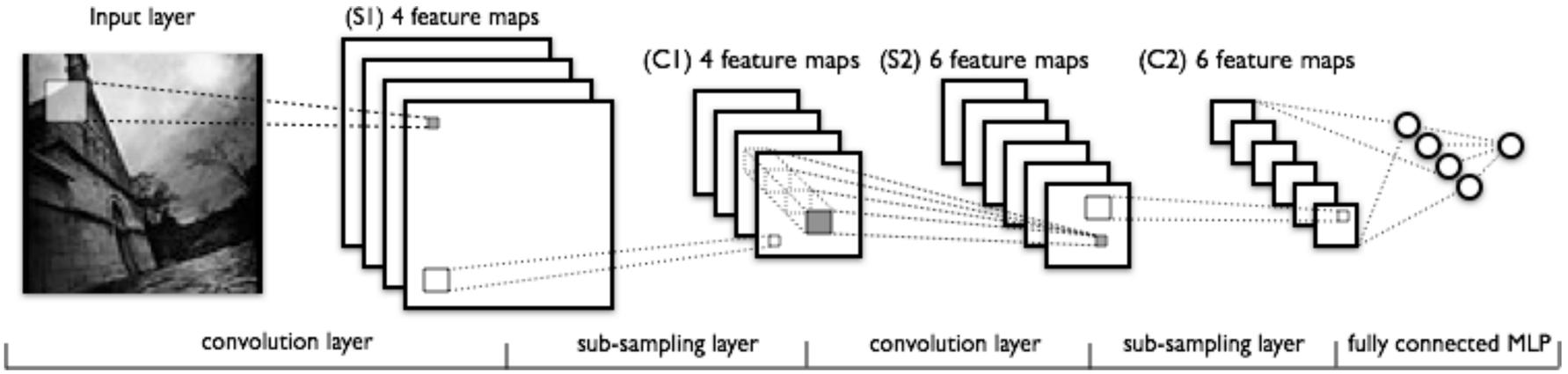
Its like Perceptrons are together in a defined fashion

Two Layered Neural Network



These array of perceptrons form the Neural Nets

Convolution Neural Network

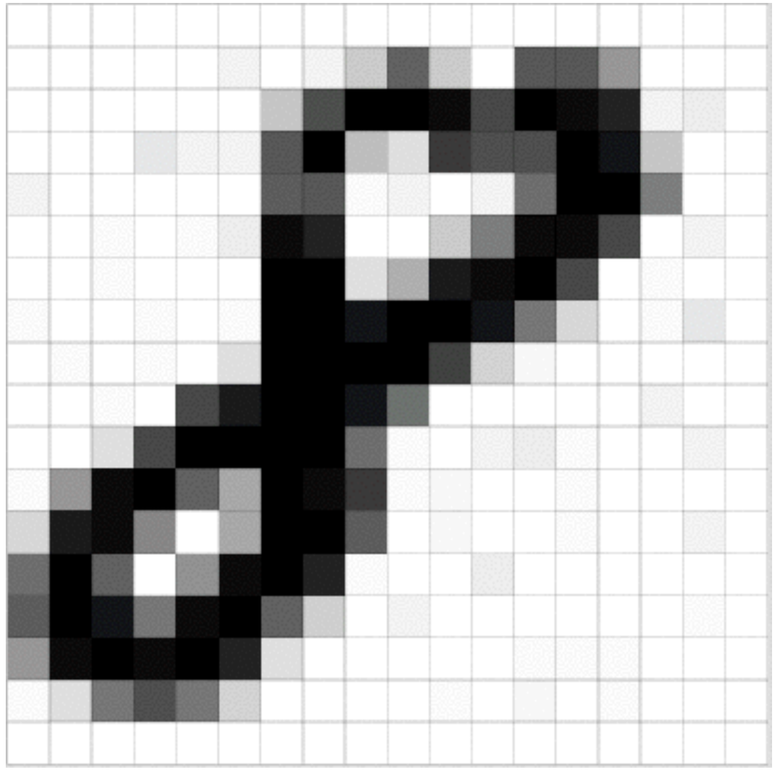


- CNNs are Neural Networks that are sparsely (not fully) connected on the input layer.
- This makes each neuron in the following layer responsible for only part of the input, which is valuable for recognizing parts of images, for example.

Convolutional Neural Network

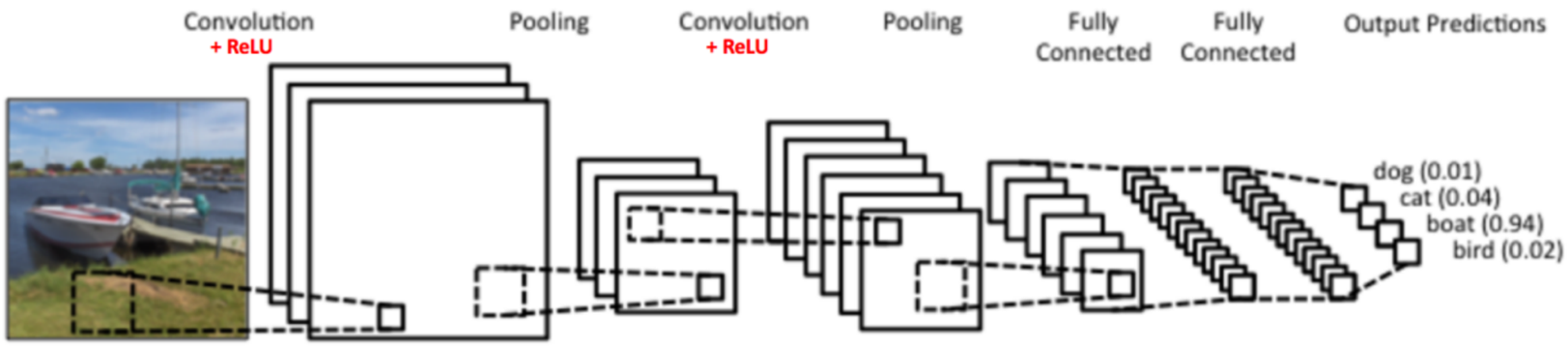
- Convolutional Neural Networks are very similar to ordinary Neural Networks
- The architecture of a CNN is designed to take advantage of the 2D structure of an input image

What is an Image ?



- Image is a matrix of pixel values
- Believe me Image is nothing more than this.

CNN aka Deep Neural Network



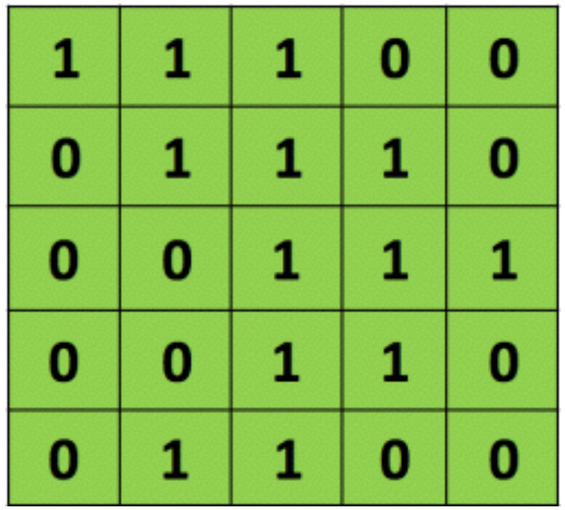
1.Convolution

2.Activation Functions (ReLU)

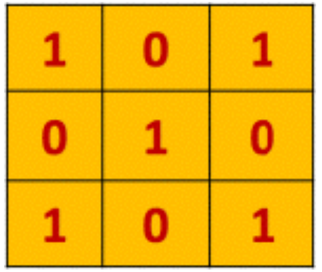
3.Pooling or Sub Sampling

4.Classification (Fully Connected Layer)

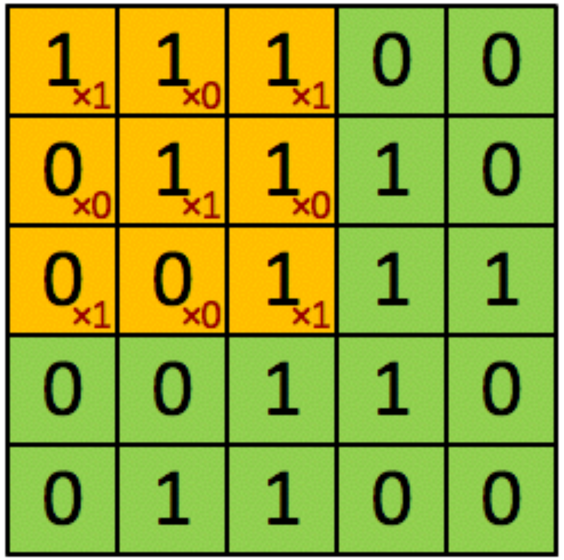
1. What is Convolution



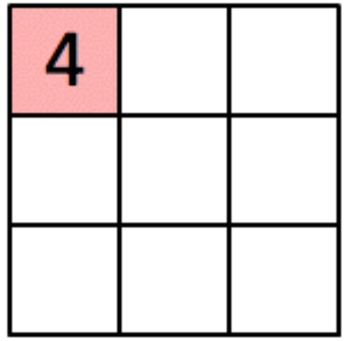
Image



Filter



Image

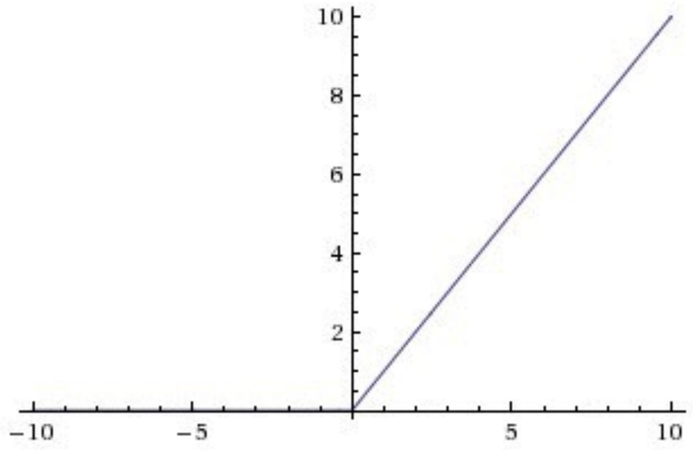


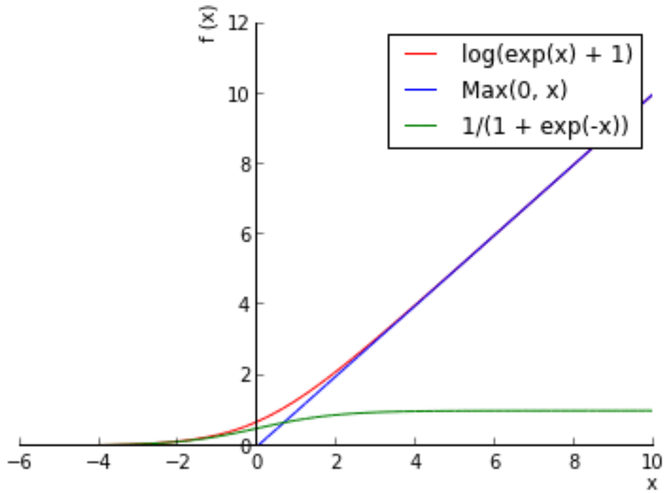
Convolved Feature

This is Convolution

Yeah ! That's it.

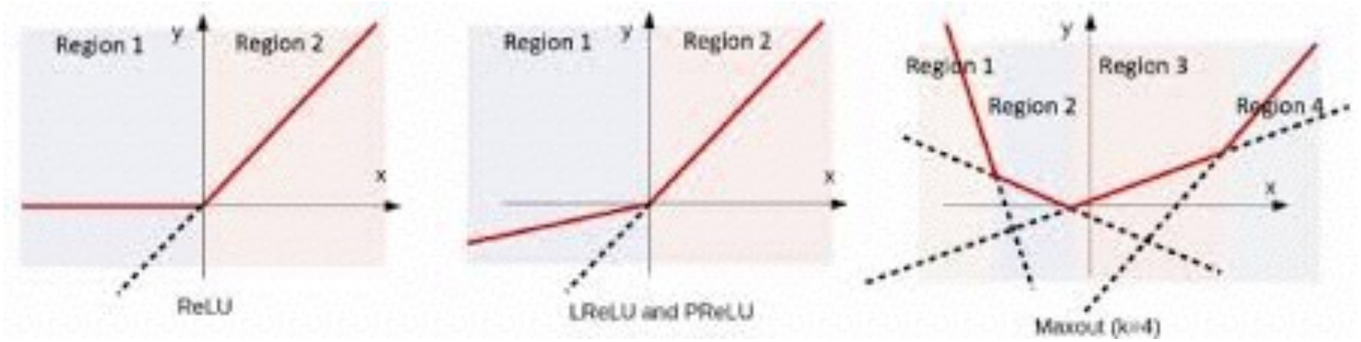
2. Activation Functions (Relu)





Relu

In CNN "Relu" Activation is preferred over other Activation Functions because of it's inherent simplicity

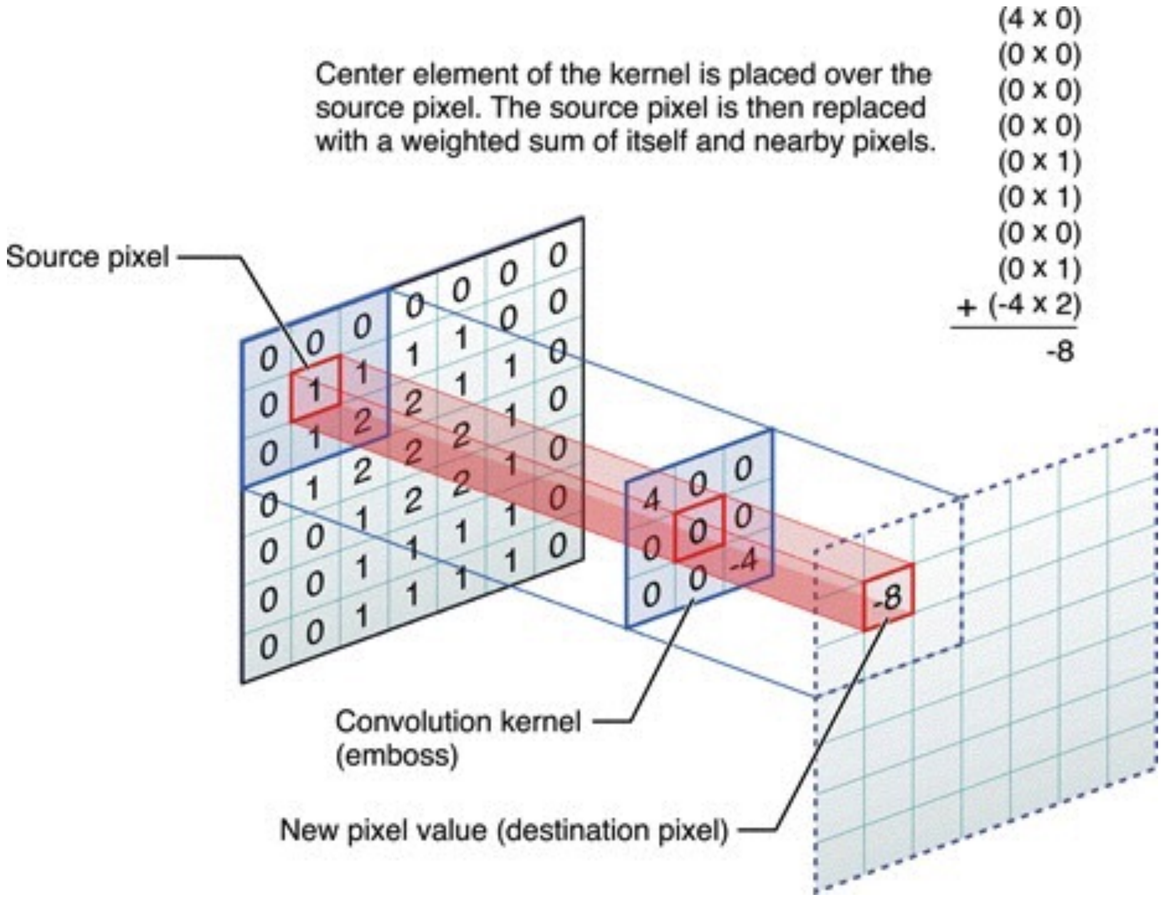


This enables models to attain non-linearity in the decision boundary.

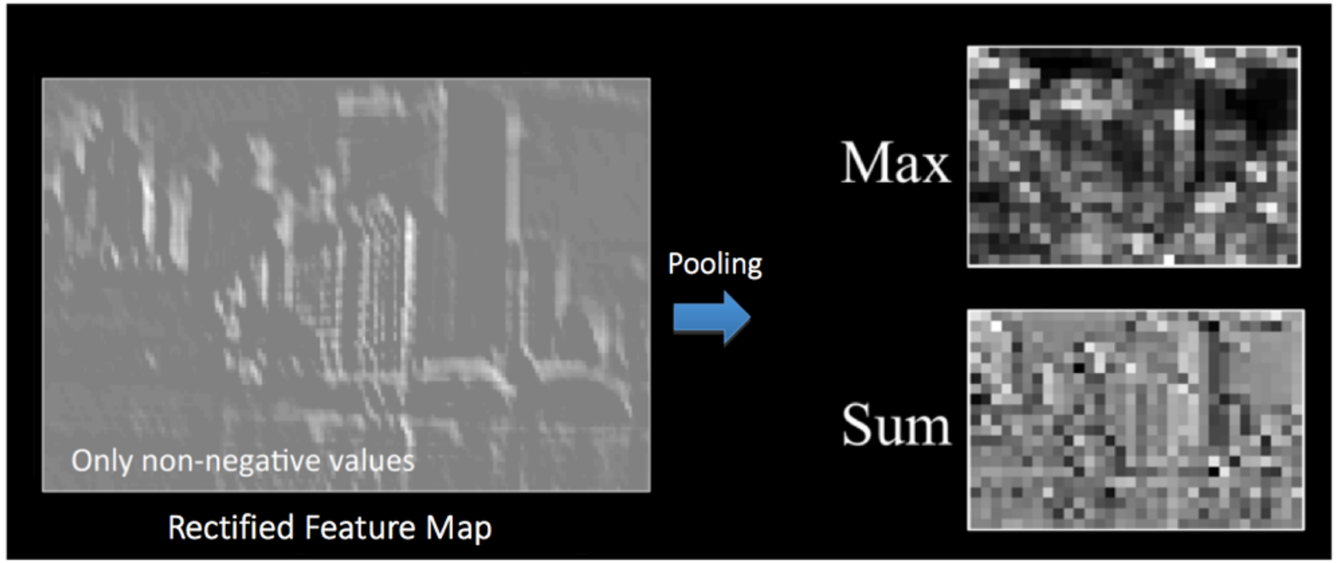
Relu advantages

- Greatly accelerate the convergence of sgd. It is argued that this is due to its linear, non-saturating form.
- Avoids expensive computations

3. Max-Pooling

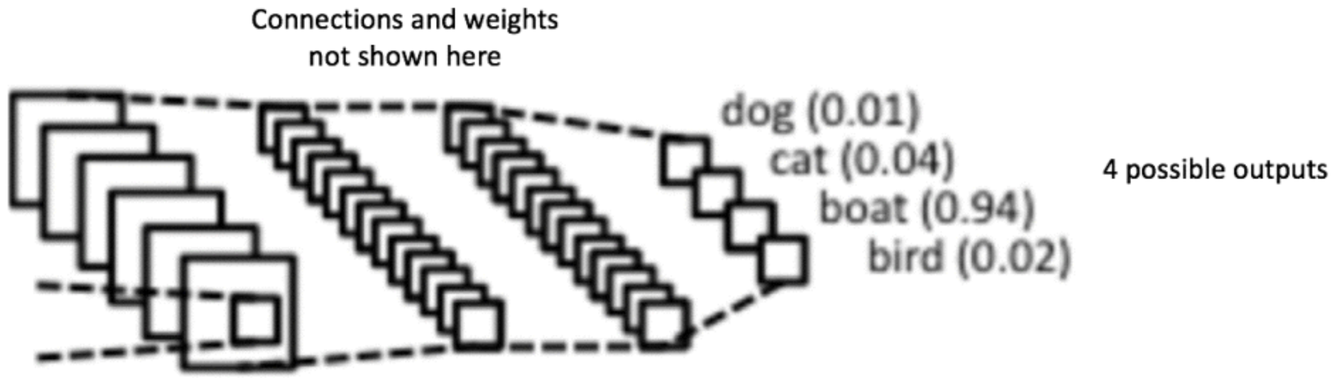


Pooling in Action



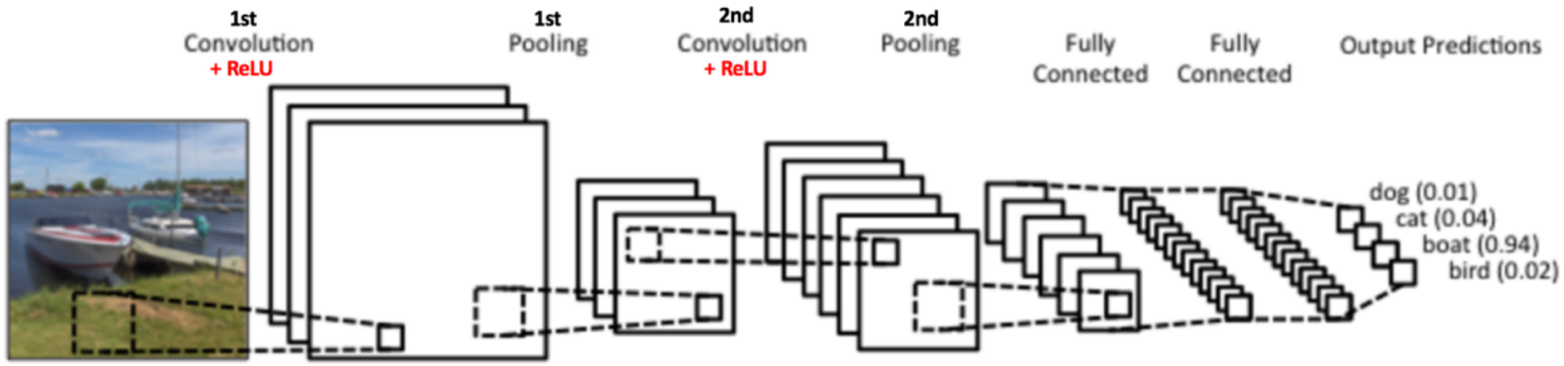
Spatial Pooling (also called **subsampling** or **downsampling**) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

4. Fully Connected Layer



The purpose of the Fully Connected layer is to use **high-level features** for classifying the input image into various classes based on the training dataset.

Connecting dots...



1.Initialize the layers with random values.

2.The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.

3.Lets say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3], while the target probabilities are [0, 0, 1, 0].

4.Since weights are randomly assigned for the first training example, output probabilities are also random.

5.Calculate the total error at the output layer (summation over all 4 classes.

1.Total Error = Σ ½ (target probability – output probability) ^{2}

6.Use Back-propagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error

- Repeat steps 2-6 with all images in the training set.

Give me the code !

Perceptron Training Snapshot

```
yardstick17 ConnectingDots/perceptron master +(3186e0a) python3 perceptron.py
Training Data :
          X  Y
0  [0, 0, 1]  0
1  [1, 1, 1]  1
2  [1, 0, 1]  1
3  [0, 1, 1]  0
Training SetUp :
number_of_epoch:  5000
beta: 0.4
epoch :  0 loss :  2
epoch :  1 loss :  3
epoch :  2 loss :  1
epoch :  3 loss :  0
perceptron for given dataset trained...
   feature_row  true_label        weight vector  predicted_output            theta            delta_W
0    [0, 0, 1]           0    [[0.5, 0.1, 0.3]]                 1    [1.29676530975]  [-0.0, -0.0, -0.4]
1    [1, 1, 1]           1   [[0.5, 0.1, -0.1]]                 1    [1.46034776976]  [-0.0, -0.0, -0.0]
2    [0, 1, 1]           0   [[0.5, 0.1, -0.1]]                 1    [0.99005295995]  [-0.0, -0.4, -0.4]
3    [1, 0, 1]           1  [[0.5, -0.3, -0.5]]                 1   [0.967060119562]  [-0.0, -0.0, -0.0]
4    [1, 0, 1]           1  [[0.5, -0.3, -0.5]]                 1   [0.967060119562]  [-0.0, -0.0, -0.0]
5    [0, 1, 1]           0  [[0.5, -0.3, -0.5]]                 1    [0.19005295995]  [-0.0, -0.4, -0.4]
6    [1, 1, 1]           1  [[0.5, -0.7, -0.9]]                 0  [-0.139652230241]     [0.4, 0.4, 0.4]
7    [0, 0, 1]           0  [[0.9, -0.3, -0.5]]                 1   [0.496765309754]  [-0.0, -0.0, -0.4]
8    [0, 0, 1]           0  [[0.9, -0.3, -0.9]]                 1  [0.0967653097536]  [-0.0, -0.0, -0.4]
9    [1, 0, 1]           1  [[0.9, -0.3, -1.3]]                 1   [0.567060119562]  [-0.0, -0.0, -0.0]
10   [1, 1, 1]           1  [[0.9, -0.3, -1.3]]                 1   [0.260347769759]  [-0.0, -0.0, -0.0]
11   [0, 1, 1]           0  [[0.9, -0.3, -1.3]]                 0  [-0.60994704005]  [-0.0, -0.0, -0.0]
12   [0, 1, 1]           0  [[0.9, -0.3, -1.3]]                 0  [-0.60994704005]  [-0.0, -0.0, -0.0]
13   [1, 1, 1]           1  [[0.9, -0.3, -1.3]]                 1   [0.260347769759]  [-0.0, -0.0, -0.0]
14   [1, 0, 1]           1  [[0.9, -0.3, -1.3]]                 1   [0.567060119562]  [-0.0, -0.0, -0.0]
15   [0, 0, 1]           0  [[0.9, -0.3, -1.3]]                 0  [-0.303234690246]  [-0.0, -0.0, -0.0]
yardstick17 ConnectingDots/perceptron master ●+(3186e0a)
```

CNN training Snapshot

```
..ts/perceptron (zsh) ⌘1   ..ts/perceptron (zsh) ⌘2   tmux (ssh) ⌘3
1403/1403 [==============================] - 122s - loss: 0.0457 - acc: 0.8760 - val_loss: 0.0866 - val_acc: 0.7600                [2/663]
Epoch 40/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0440 - acc: 0.8852Epoch 00039: val_loss did not improve
1403/1403 [==============================] - 121s - loss: 0.0437 - acc: 0.8860 - val_loss: 0.1009 - val_acc: 0.7086
Epoch 41/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0400 - acc: 0.8953Epoch 00040: val_loss improved from 0.07092 to 0.06898, saving model to /home/kushwaha/image_clazzif
ication.model.best.h5
1403/1403 [==============================] - 135s - loss: 0.0396 - acc: 0.8959 - val_loss: 0.0690 - val_acc: 0.8000
Epoch 42/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0382 - acc: 0.8990Epoch 00041: val_loss improved from 0.06898 to 0.06769, saving model to /home/kushwaha/image_clazzif
ication.model.best.h5
1403/1403 [==============================] - 136s - loss: 0.0379 - acc: 0.8995 - val_loss: 0.0677 - val_acc: 0.8171
Epoch 43/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0378 - acc: 0.9041Epoch 00042: val_loss did not improve
1403/1403 [==============================] - 122s - loss: 0.0377 - acc: 0.9045 - val_loss: 0.0720 - val_acc: 0.8000
Epoch 44/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0364 - acc: 0.9055Epoch 00043: val_loss improved from 0.06769 to 0.06753, saving model to /home/kushwaha/image_clazzif
ication.model.best.h5
1403/1403 [==============================] - 137s - loss: 0.0363 - acc: 0.9059 - val_loss: 0.0675 - val_acc: 0.8114
Epoch 45/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0328 - acc: 0.9172Epoch 00044: val_loss did not improve
1403/1403 [==============================] - 122s - loss: 0.0326 - acc: 0.9180 - val_loss: 0.0701 - val_acc: 0.7943
Epoch 46/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0312 - acc: 0.9164Epoch 00045: val_loss did not improve
1403/1403 [==============================] - 122s - loss: 0.0308 - acc: 0.9173 - val_loss: 0.0700 - val_acc: 0.8086
Epoch 47/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0331 - acc: 0.9193Epoch 00046: val_loss did not improve
1403/1403 [==============================] - 122s - loss: 0.0327 - acc: 0.9202 - val_loss: 0.0680 - val_acc: 0.8143
Epoch 48/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0304 - acc: 0.9251Epoch 00047: val_loss did not improve
1403/1403 [==============================] - 122s - loss: 0.0302 - acc: 0.9259 - val_loss: 0.0713 - val_acc: 0.8000
Epoch 49/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0324 - acc: 0.9172Epoch 00048: val_loss did not improve
1403/1403 [==============================] - 122s - loss: 0.0322 - acc: 0.9173 - val_loss: 0.0799 - val_acc: 0.7800
Epoch 50/50
1376/1403 [============================>.] - ETA: 1s - loss: 0.0258 - acc: 0.9353Epoch 00049: val_loss did not improve
1403/1403 [==============================] - 122s - loss: 0.0257 - acc: 0.9359 - val_loss: 0.0695 - val_acc: 0.7971
kushwaha@reviews-horse ~/ConectingDots master(e848d1e)
```

Find the code at
https://github.com/yardstick17/ConnectingDots

Thanks

References

Find me on web

@yardstick17