Deep Learning From Scratch to practice

WSU Python Working Group

by Jikhan Jeong

Focus :

1. When we use Deep Learning (Theory)

2. How it work (Basic Scratch)

3. How to apply (Practice)

Contents :

1. History (3min)
2. Intro (2min)
3. Single Perceptron : Concept + Code Practice (20min)
4. XOR problem : Concept + Code Practice(20min)
5. Backpropagation (Concept Only due to time) (10min)
6. Practice : CNN Image recognition in MNIST (5min)

*Due to time limitation, CNN is roughly explained My computer is so slow, it is possible that CNN Outcome may not seeable

Contents :

History of Deep Learning

- Progression (1943-1960)
  - First Mathematical model of neurons, Pitts & McCulloch (1943)
  - Beginning of artificial neural networks–Perceptron, Rosenblatt (1958)
- Degression (1960-1980)
  - Perceptron can't even learn the XOR function
  - We don't know how to train MLP
  - 1963 Backpropagation (Bryson et al.)
- Progression (1980-)
  - 1986 Backpropagation reinvented
- Degression (1993-)
  - SVM: Support Vector Machine is developed by Vapnik et al.[1995]
  - Graphical models are becoming more and more popular
  - Training deeper networks consistently yields poor results.
  - However, Yann LeCun (1998) developed deep convolutional neural networks
- Progression (2006-)
  - Deep Belief Networks (DBN) by Hinton et al. (2006)
  - Deep Autoencoder based networks by Greedy Layer-Wise Training of Deep Networks. Bengio et al.
  - Convolutional neural networks running on GPUs
  - AlexNet (2012). Krizhevsky et al.

- Single Perceptron cannot solve OXR problem

-> Falling of Neural Network

- Solve it with Multilayer Perception with Hidden Layer

http://www.cs.cmu.edu/~10701/slides/Perceptron_Reading_Material.pdf

Contents :

Machine Learning Map

Neural Networks

Multi-Layer Perceptron

Deep Neural Networks
- DBN
- **CNN** -> Image Recognition
- RNN -> NLP, etc

2-Layer Perceptron ~ Regression
- Linear
- Logistic
- Softmax

Reinforcement Learning

Deep Reinforcement Learning (DQN)

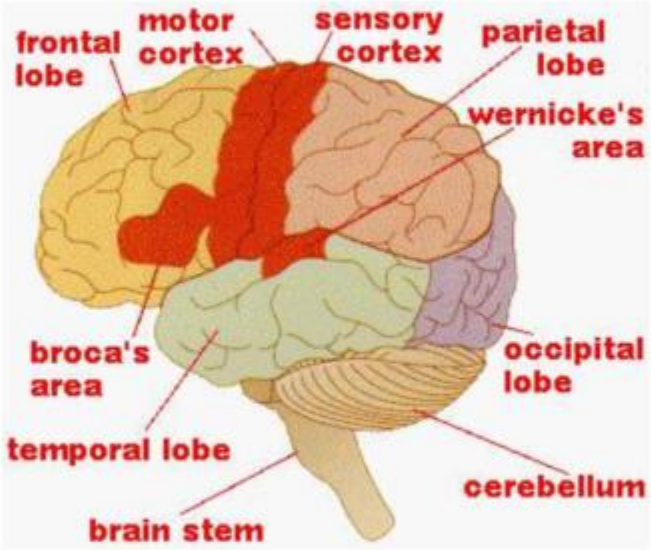
Supervised Learning
Unsupervised Learning

Discriminative Model
Generative Model

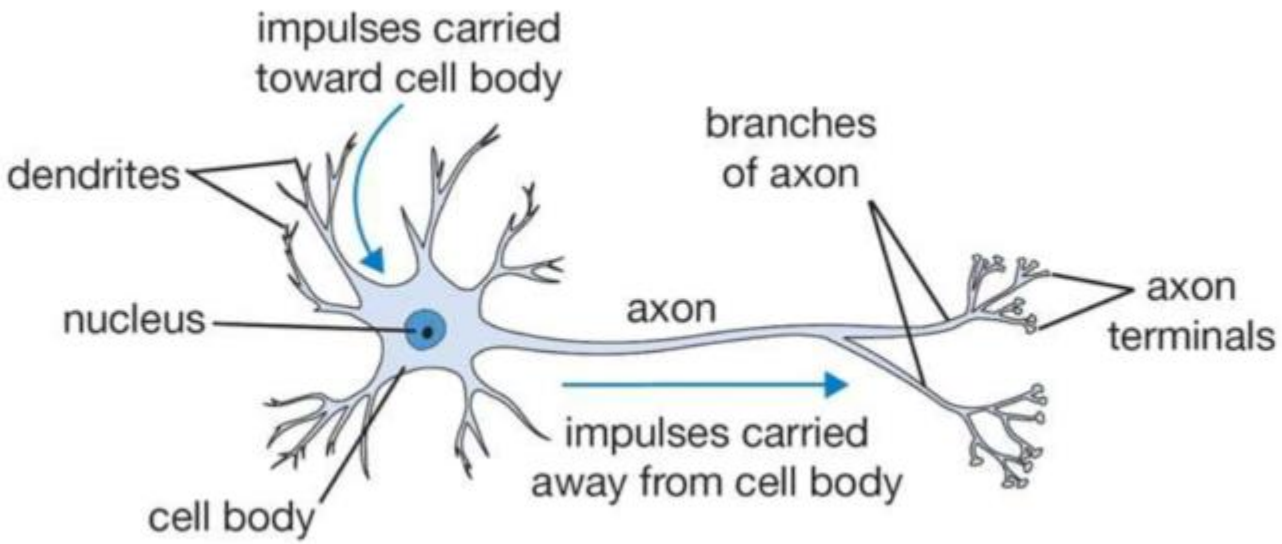
Contents :

1. History
2. Intro
3. Single Perceptron : Concept + Code Practice

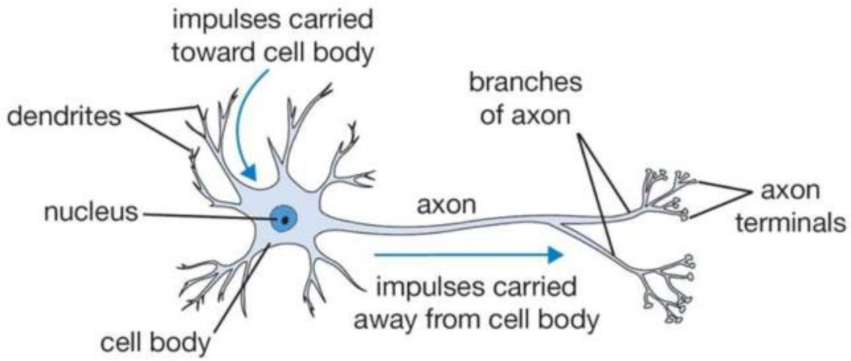
Out Brain



One Neuron

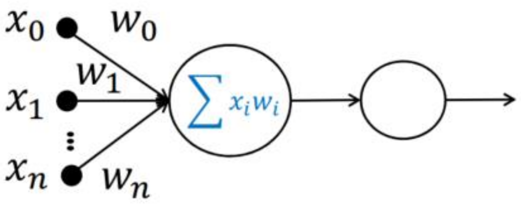


One Neuron



Information set = feature sets = $\{X_0, X_1, \ldots, X_n\}$

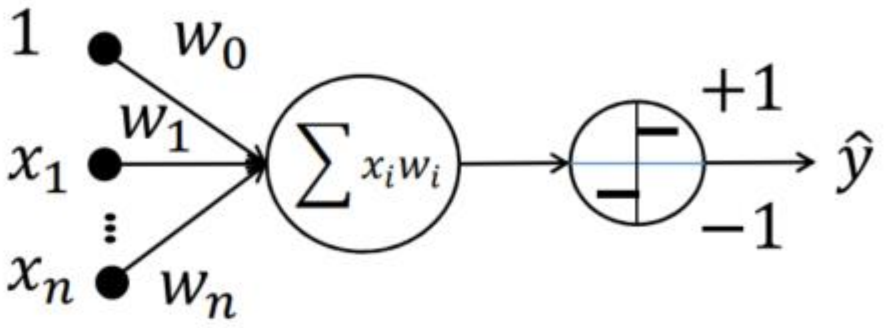
For example X_0 = height
X_1 = weight
X_2 = voice
X_n = ...



$$\sum x_i w_i$$

$$= x_0 \times w_0 + x_1 \times w_1 + x_n \times w_n$$

Perceptron

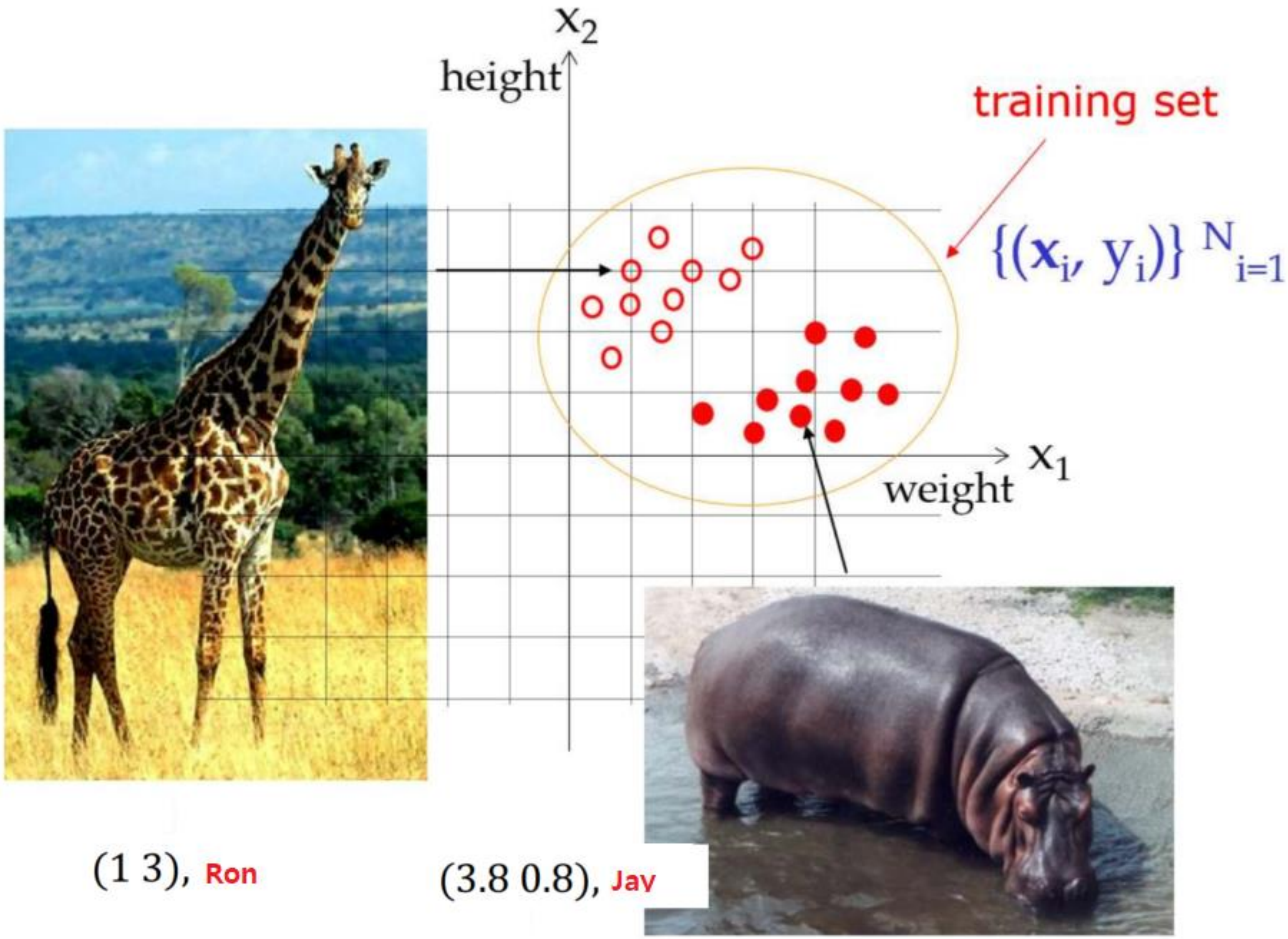


Just linear equation

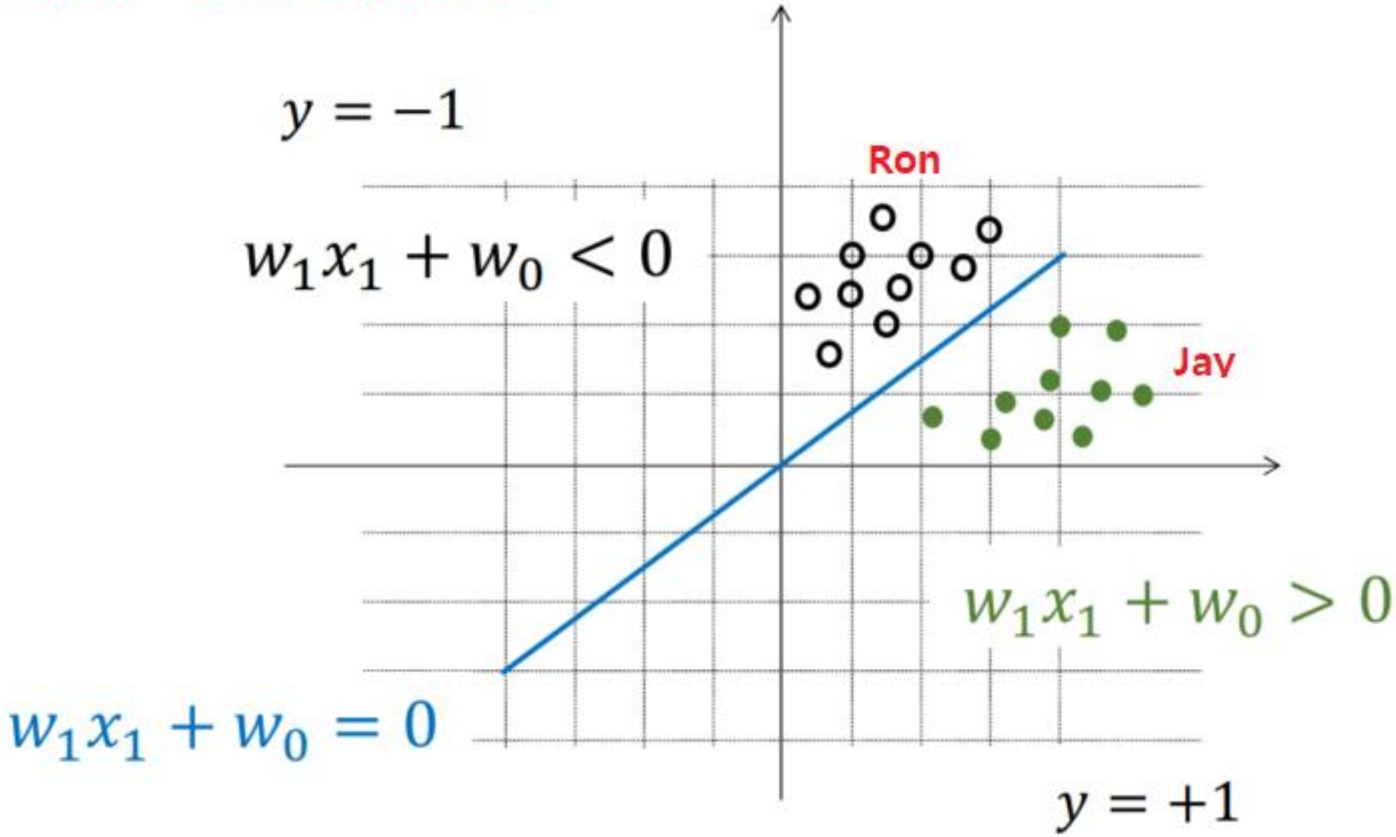
$$\hat{y}(x_1, \cdots, x_n) = \begin{cases} +1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases} .$$

$$\hat{y}(\boldsymbol{x}) = \begin{cases} +1 & \text{if } \boldsymbol{w}^T \boldsymbol{x} > 0 \\ -1 & \text{otherwise} \end{cases} .$$

Binary Classification



Training Set => Learning Weight



$$w_1 x_1 + w_0 x_0 = 0$$

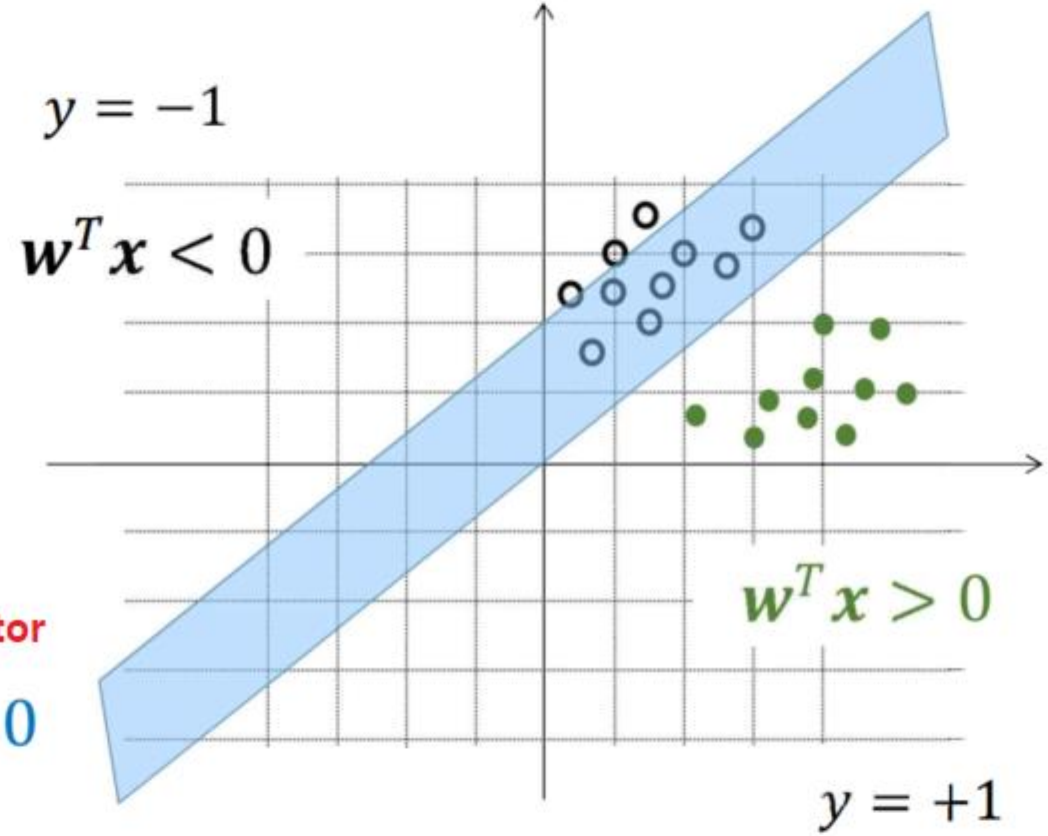
$$(w_0 \; w_1)\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = 0$$

$$\boldsymbol{w}^{\boldsymbol{T}} = (w_0 \; w_1)$$

$$\boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

Usining Vector

$$\boldsymbol{w}^T \boldsymbol{x} = 0$$



plane

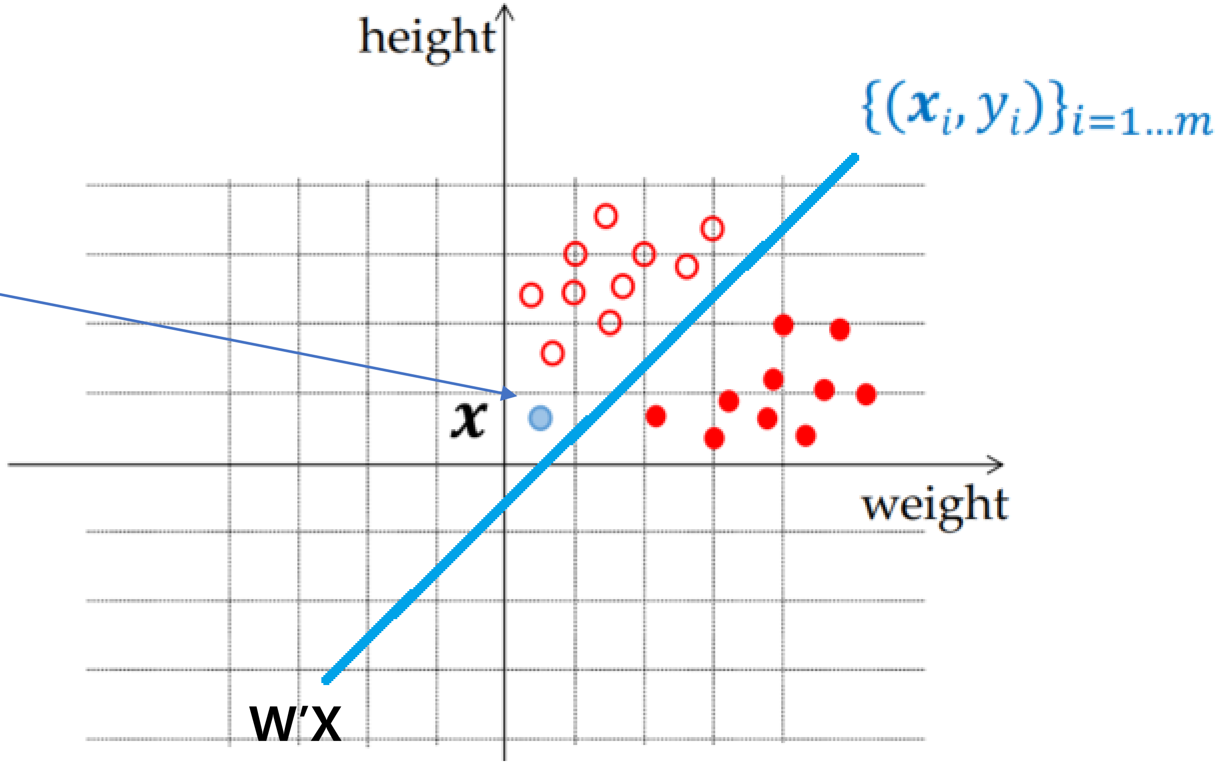
#X = 2

Hyper-plane

#X >2

New Data from test set,

(Prediction) Is he is Ron or Jay?



Hyperplane is **fitted** from training set

-> It means we learns **weight**

Actual Algorithm

Algorithm 5 PerceptronTrain($\mathbf{D}$, *MaxIter*)

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Algorithm 6 PerceptronTest($w_0, w_1, \dots, w_D, b, \hat{x}$)

$a \leftarrow \sum_{d=1}^{D} w_d \, \hat{x}_d + b$ // compute activation for the test example
return sign(a)

Reference: http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

Time to code

Reference: http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

Actual Code

Algorithm 5 PERCEPTRONTRAIN($\mathbf{D}$, $MaxIter$)

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + y x_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Algorithm 6 PERCEPTRONTEST($w_0, w_1, \dots, w_D, b, \hat{x}$)

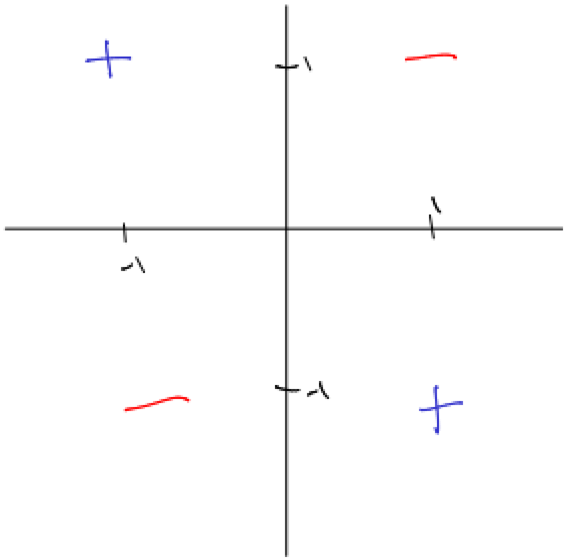
$a \leftarrow \sum_{d=1}^{D} w_d \, \hat{x}_d + b$ // compute activation for the test example
return SIGN(a)

```
def perceptron(feature, label, iters):

    w = np.zeros(feature.shape[1]) # array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    final_iter = iters

    for iter in range(iters): # For Each Training Iteration to Max

        error = 0
        yes = 0

        for i in range(len(feature)): ## all sample n = 155

            y = label[i]          # actual value in i row (1)
            x = feature.loc[i]     # feature vector in i row
            a = np.dot(w, x) # Predicted value (2)
            m = np.sign(y*a) # (Margins) m>0 correct, m<0 wrong
            if m <= 0:       # (wrong guess) if the prediciton is wrong
                w = w + np.dot(x, y)  # update the weight, b is not considered
                error += 1   # update the numer of error
            else:            # (correct guess)
                yes += 1
                pass
        print("iter: {}".format(iter), "Accuracy: {}".format(1-error/len(feature)))
    return  w, error, yes, final_iter,1-error/len(X)

perceptron(X,Y,1000)
```

Reference: http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

Contents :

Can you make a one line to classify + or – with perceptron?



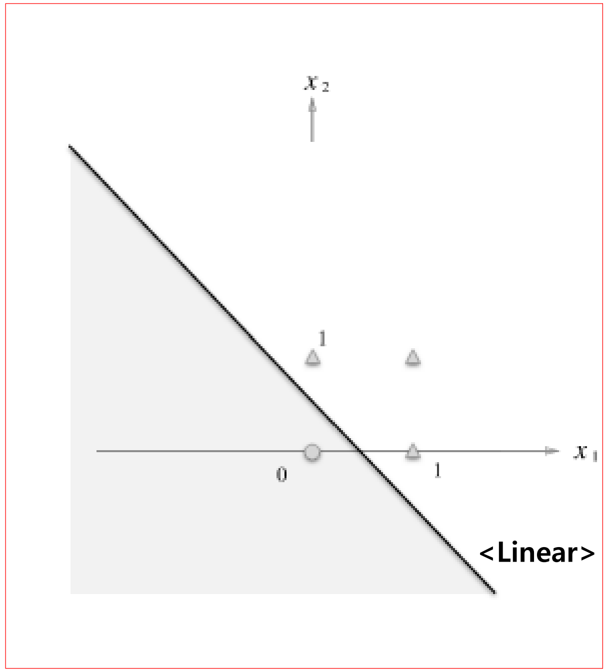
XOR probelm

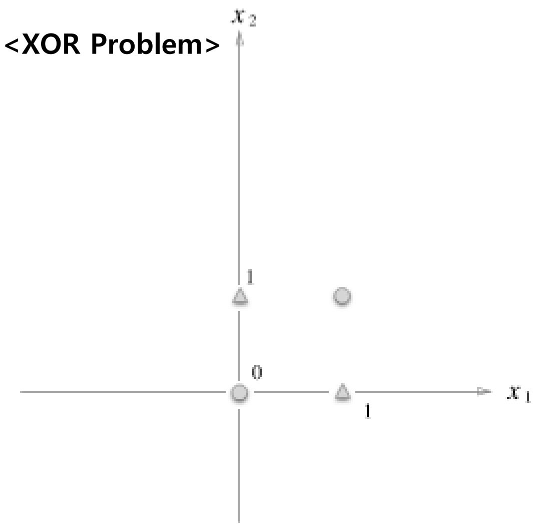
NO

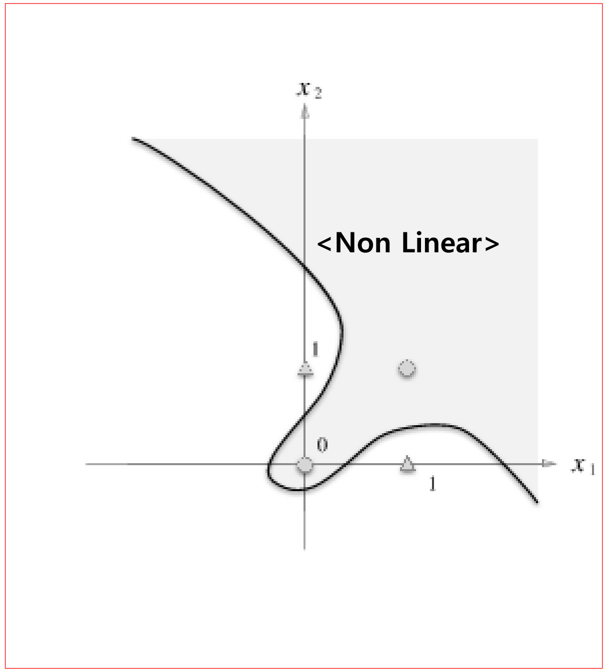
Yes

Single Perceptron -> Linear

Multi-Perceptron -> Non-linear



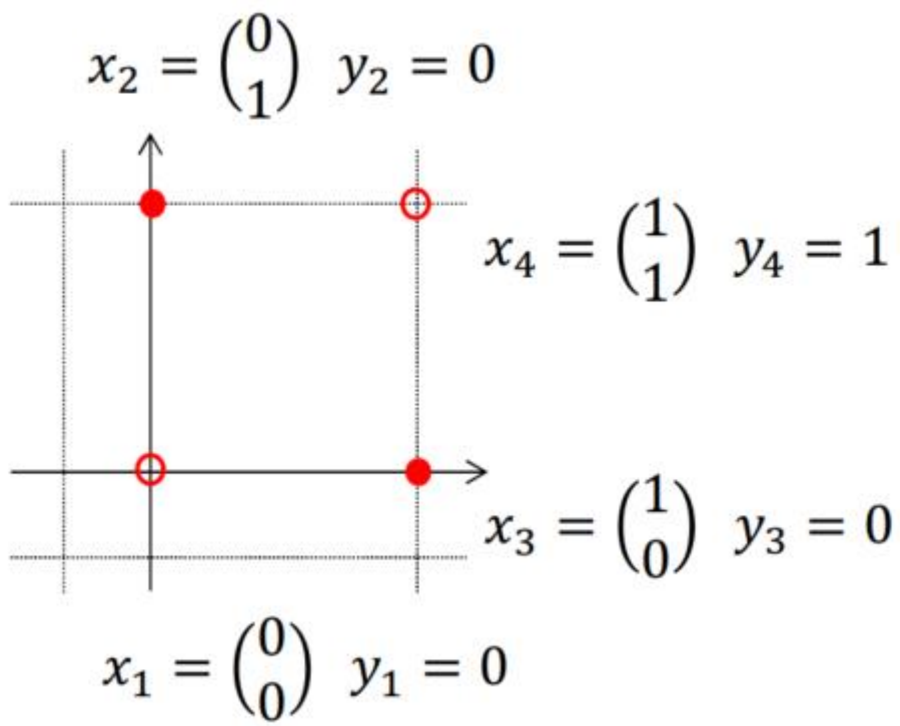




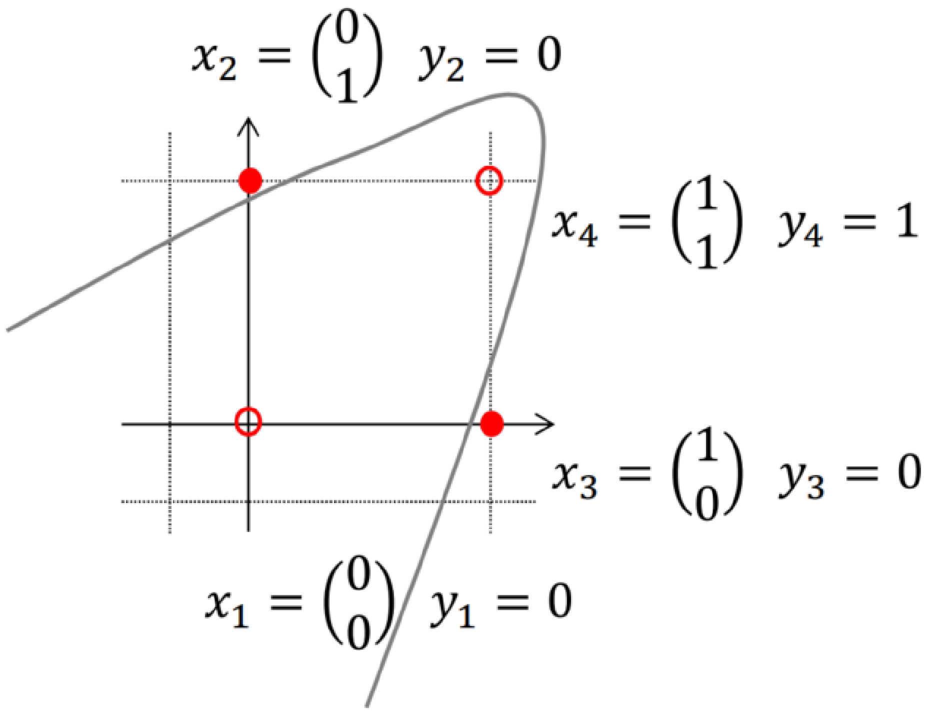
Reference: jiwon Seo, Deep Learning Edu 5th
chapter 2. Perceptron(2.4~)

Code Practice

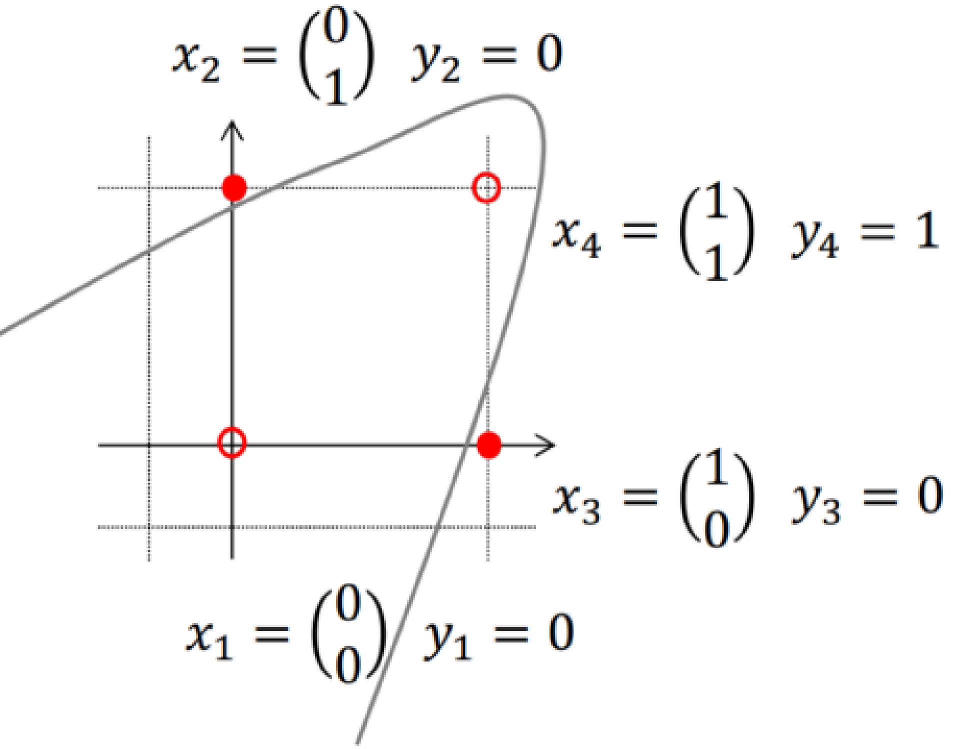
Single Perceptron -> Linear

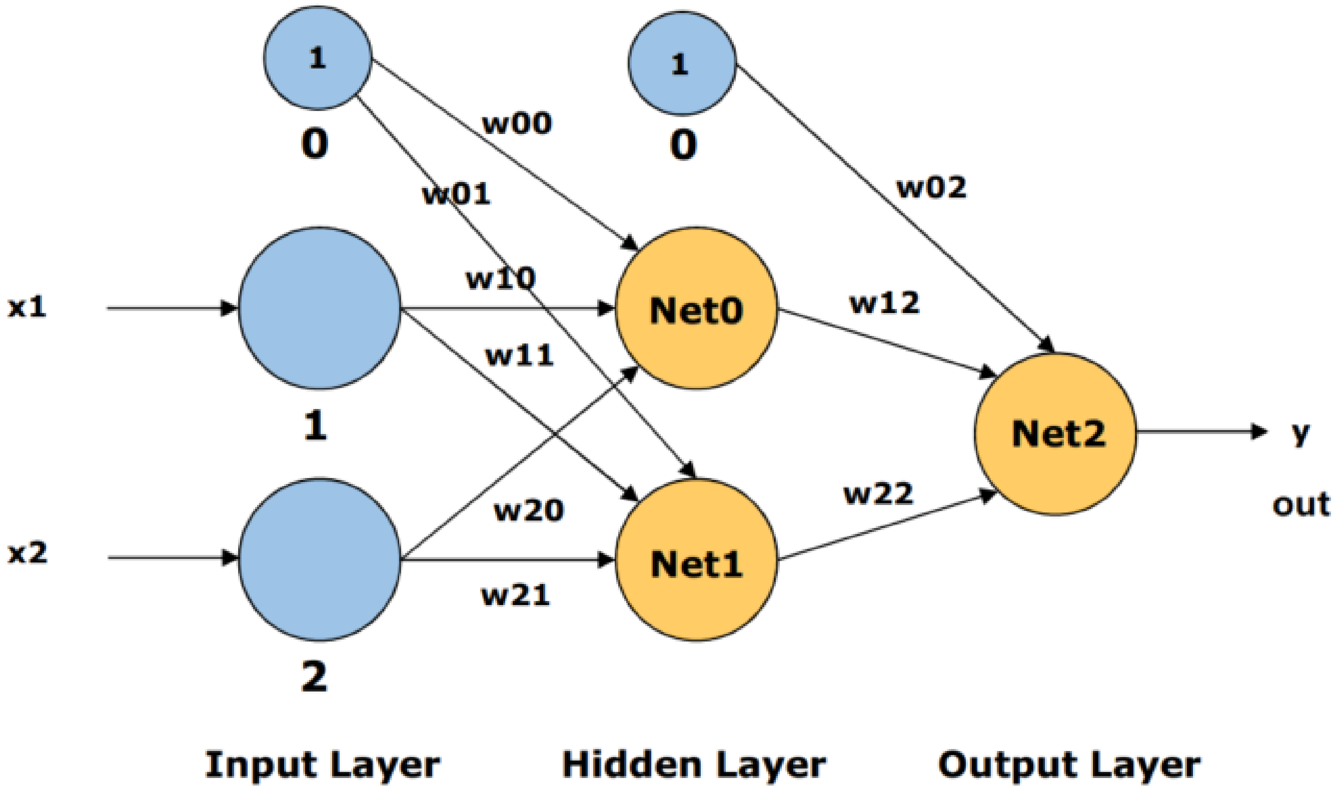


Multi-Perceptron -> Non-linear



3 Perceptron = 2 perceptron in hidden layer
1 perceptron in output layer

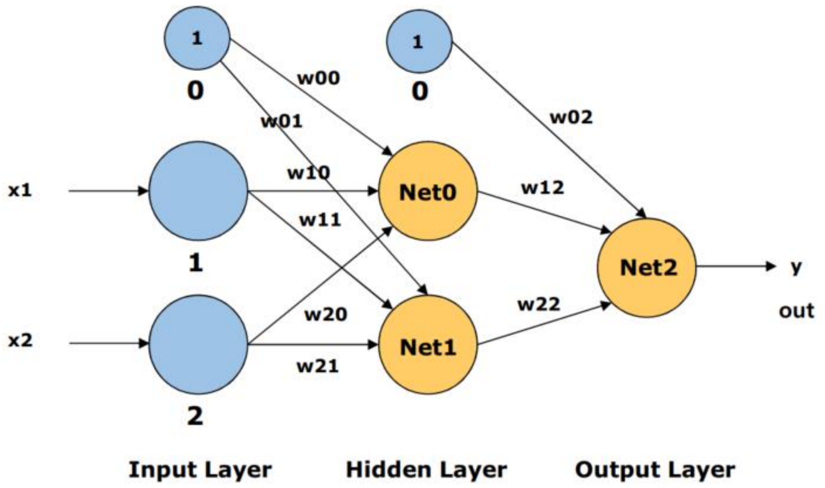




Assumes we already know the right **weights** for each perceptron for now

Coding Time

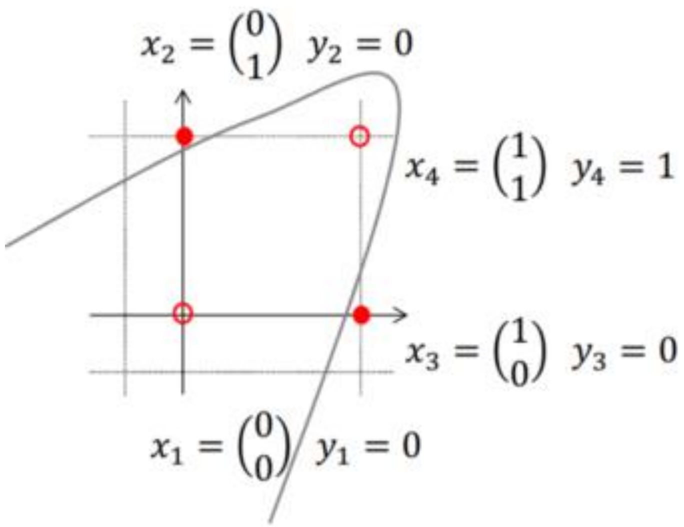
Just quick running for checking whether it works or not




```python
# Perceptron

def perceptron(x,w,b):
    y= np.sum(w*x) +b
    if y<=0:
        return 0
    else:
        return 1

# Non-and Gate (1)

def NAND(x1, x2):
    return perceptron(np.array([x1, x2]), w11, b1)    ## perceptron 1 (hidden layer)

# OR Gate      (2)
def OR(x1, x2):
    return perceptron(np.array([x1, x2]), w12, b2)    ## perceptron 2 (hidden layer)

# And Gate     (3)

def AND(x1, x2):
    return perceptron(np.array([x1, x2]), w2, b3)     ## perceptron 3 (output layer)

# XOR Gate     (3) with (1) and (2)  ---> solving XOR

def XOR(x1, x2):
    return AND(NAND(x1, x2), OR(x1, x2))              ## percetpron 1,2 -> perceptron 3

if __name__ == '__main__':
    for x in [(0,0),(1,0),(0,1),(1,1)]:
        y = XOR(x[0],x[1])
        print(x,y)
```

```
percetpron 1,2 -> perceptron 3
   ...:
   ...:
   ...: if __name__ == '__main__':
   ...:     for x in [(0,0),(1,0),(0,1),(1,1)]:
   ...:         y = XOR(x[0],x[1])
   ...:         print(x,y)
(0, 0) 0
(1, 0) 1
(0, 1) 1
(1, 1) 0
```

(x1,x2) y -→ Y is 0 or 1 means Ron or Jay

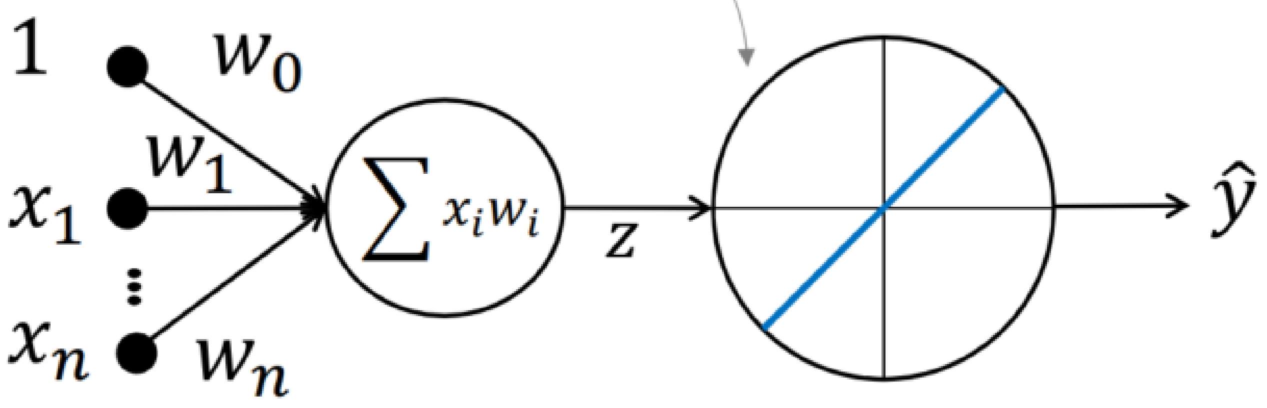


It works!!!!

activation function

The same with single perceptron

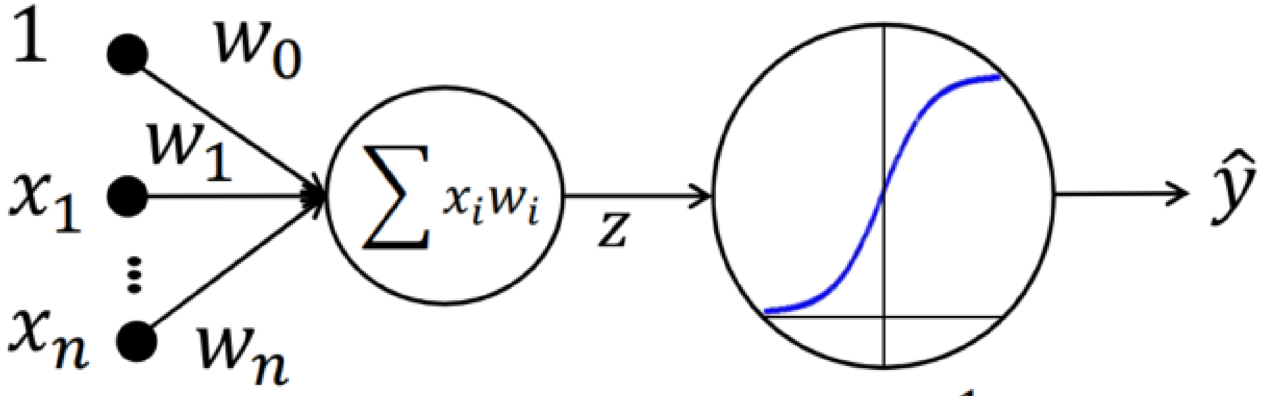
Put activation function



$f(z) = z$

Sum of linear function, Z, go to
Linear activation function

-> Still Linear



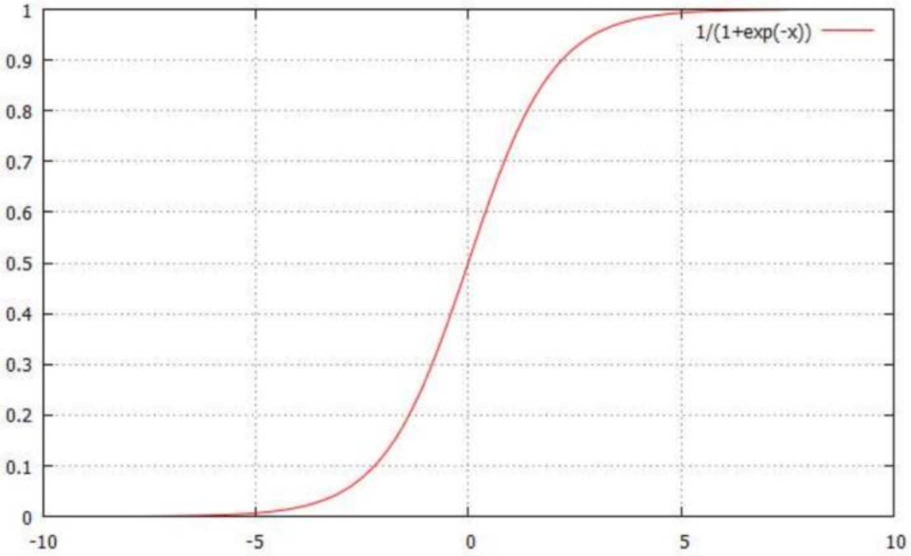
$f(z) = \frac{1}{1 + e^{-z}}$ Sigmoid

Sum of linear function, Z, go to
Non-linear(Sigmoid, ReLu, Tanh, etc)

→ **Non-linear**

Sigmoid function (one candidate for activation function to make the Z to non-linear)

f(z)



z

$$\boxed{f(z)} = \frac{1}{1+e^{-z}} = \frac{e^{z}}{1+e^{z}}$$

f(z) is bounded to +1 and -1

= |f(z)| < 1

Differentiation of sigmoid function is the function of itself

$$\frac{d}{dx}\boxed{f(x)} = \boxed{f(x)}(1-\boxed{f(x)})$$

$$\frac{d}{dx}e^{x} = e^{x}\frac{d}{dx}(x) = e^{x}$$

$$f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{[h(x)]^2}$$

chain rule

$$\boxed{\frac{d}{dx}f(x)} = \frac{d}{dx}\frac{1}{1+e^{-x}} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1-1+e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$$

$$= \left(\frac{1}{1+e^{-x}}\right) - \left(\frac{1}{1+e^{-x}}\right)^2 = \left(\frac{1}{1+e^{-x}}\right)\left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= \boxed{f(x)\left(1 - f(x)\right)}$$

Assumes we already know the right **weights** for each perceptron for now

How we can?

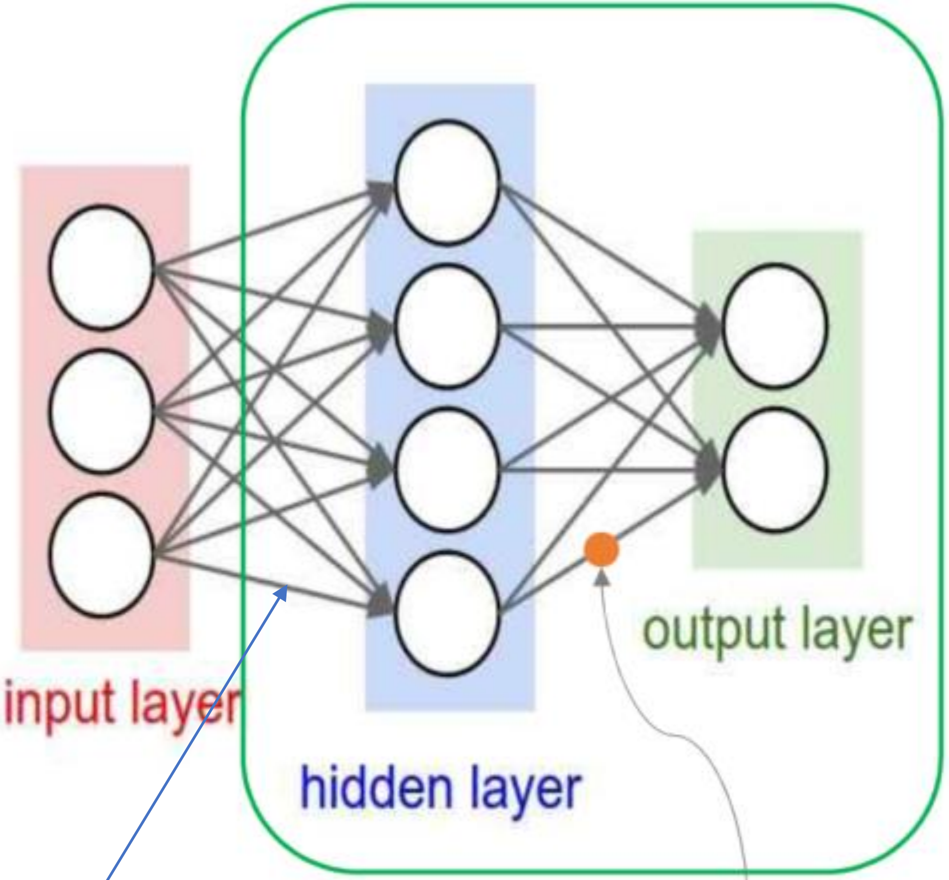
Contents :

1. History
2. Intro
3. Single Perceptron : Concept + Code Practice
4. XOR problem : Concept + Code Practice
5. Backpropagation (Concept Only due to time)

Backpropagation

-> Takes 25 Years, But we will spend 3 mins

- Progression (1943–1960)
  - First Mathematical model of neurons, Pitts & McCulloch (1943
  - Beginning of artificial neural networks–Perceptron, Rosenblat
- Degression (1960–1980)
  - Perceptron can't even learn the XOR function
  - We don't know how to train MLP
  - 1963 Backpropagation (Bryson et al.)
- Progression (1980–)
  - 1986 Backpropagation reinvented



we know the value in the outpub layer

$$E(\mathbf{w}) = \frac{1}{2}\sum_{d \in D}(y_d - \hat{y}_d)^2$$

$$\frac{\partial E}{\partial w_j}$$

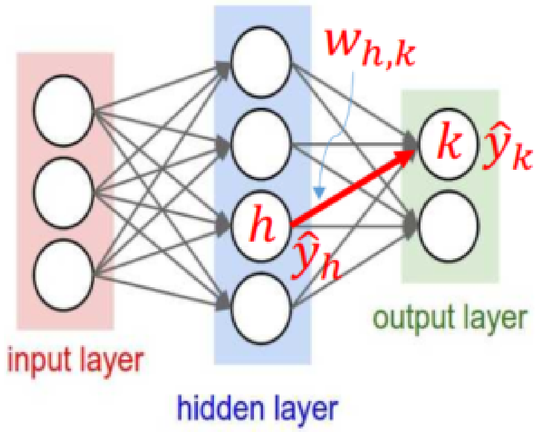
possible to

differentiation

$$\frac{\partial E}{\partial w_i}$$

Very hard to differentiation in The hidden layer

Solve it with backpropagation



Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do

  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

     $$\delta_k \leftarrow \hat{y}_k(1-\hat{y}_k)(y_k-\hat{y}_k)$$

  3. For each hidden unit h

     $$\delta_h \leftarrow \hat{y}_h(1-\hat{y}_h)\sum_k w_{h,k}\,\delta_k$$

  4. Update each network weight $w_{i,j}$

     $$w_{i,j} \leftarrow w_{i,j} + \boxed{\Delta w_{i,j}}$$

     where

     $$\boxed{\Delta w_{i,j}} = \eta\delta_j\hat{y}_i \quad ,\hat{y}_i = x_i \text{ when } i \text{ is of the first layer}$$

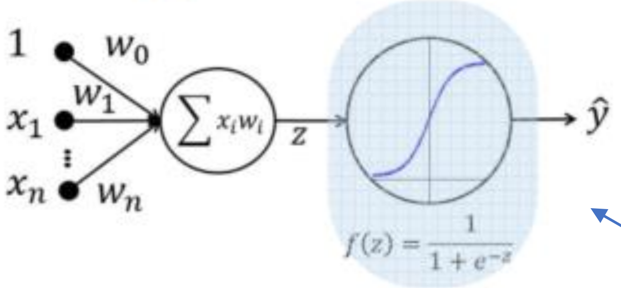
$$\boxed{\Delta w_i} = -\eta\frac{\partial E}{\partial w_i}$$
$$= \eta\,\underbrace{\hat{y}_d(1-\hat{y}_d)(y_d-\hat{y}_d)}_{\delta}(x_{i,d})$$

Updating weight

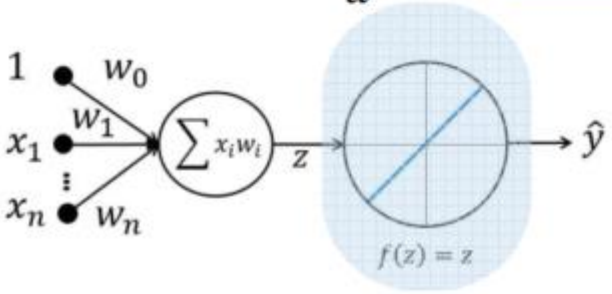
$$w_i = w_i + \boxed{\Delta w_i}$$

Neuron net

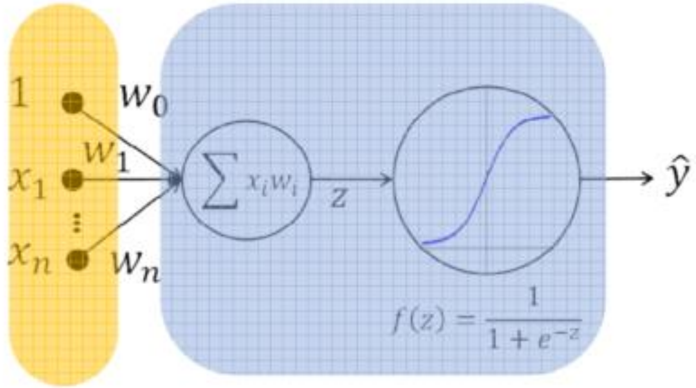
$$\boxed{\Delta w_i} = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_d \hat{y}_d(1-\hat{y}_d)(y_d-\hat{y}_d)(x_{d,i})$$



Single Perceptron $\boxed{\Delta w_i =} -\eta \frac{\partial E}{\partial w_i} = \eta \sum_d 1 \cdot (y_d-\hat{y}_d)(x_{d,i})$



Difference caused by different types of activation function



$$\boxed{\hat{y} = f(z)} = \frac{1}{1+e^{-z}}$$

$$\boxed{z = \sum w_i x_i}$$

$$\boxed{\frac{\partial}{\partial z} f(z)} = f(z)\big(1 - f(z)\big)$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (y_d - \hat{y}_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(y_d - \hat{y}_d)^2$$

chain rule

$$= \frac{1}{2}\sum_d 2(y_d - \hat{y}_d)\frac{\partial}{\partial w_i}(y_d - \hat{y}_d)$$

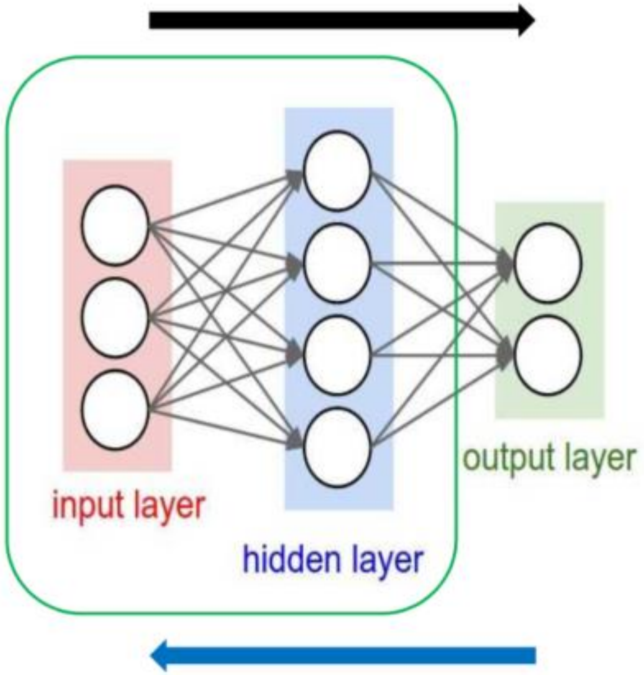
$$= \sum_d (y_d - \hat{y}_d)\frac{\partial}{\partial w_i}(-\hat{y}_d)$$

$$= -\sum_d (y_d - \hat{y}_d)\frac{\partial \hat{y}_d}{\partial z}\frac{\partial z}{\partial w_i}$$

$$= -\sum_d (y_d - \hat{y}_d)\frac{\partial f(z)}{\partial z}\frac{\partial\big(w_0 + w_1 x_{1,d} + \cdots + w_n x_{n,d}\big)}{\partial w_i}$$

$$= -\sum_d (y_d - \hat{y}_d)\hat{y}_d(1 - \hat{y}_d)x_{d,i}$$

Error happens



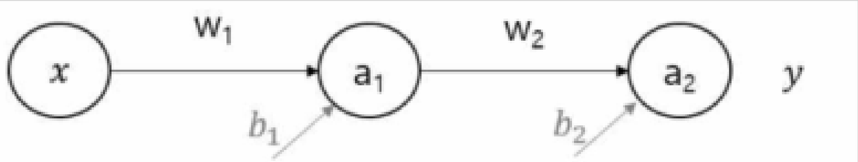
Backpropagation
= update weights

$$\boxed{\frac{\partial E}{\partial w_i}} = \frac{\partial}{\partial w_i}\boxed{\frac{1}{2}\sum_{d}(y_d - \hat{y}_d)^2}$$

Error = **C**ost Function

= f(True Y in sample – Predicted Label)

Hidden layer Output layer



$a_0 = x$ $\quad z_1 = w_1 x + b_1$ $\quad z_2 = w_2 a_1 + b_2$

$a_1 = \sigma(z_1)$ $\quad a_2 = \sigma(z_2)$

$C = \frac{1}{2}(y - a_2)^2$

$$C = \frac{1}{2}(y - a_2)^2$$

$$a_2 = \sigma(z_2)$$

$$z_2 = w_2 a_1 + b_2$$

$$a_1 = \sigma(z_1)$$

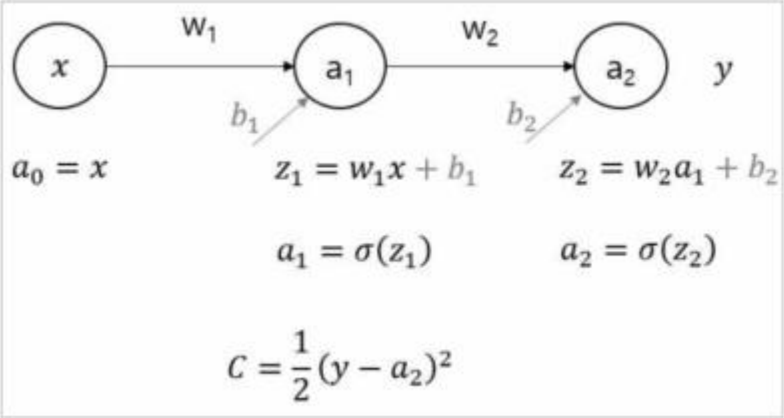
$$z_1 = w_1 x + b_1$$

Out layer $\frac{\partial C}{\partial w_2} = \frac{\boxed{\partial C}}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\boxed{\partial w_2}} = (a_2 - y)\sigma(z_2)\big(1 - \sigma(z_2)\big)a_1 = \underbrace{(a_2 - y)a_2(1 - a_2)}_{\boxed{\delta_{a_2}}}\, a_1$

Hidden layer $\frac{\partial C}{\partial w_1} = \frac{\boxed{\partial C}}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\boxed{\partial w_1}} = \underbrace{(a_2 - y)a_2(1 - a_2)w_2 a_1(1 - a_1)}_{\boxed{\delta_{a_1}}}\, x$

$$\boxed{\delta_{a_1}} = \delta_{a_2} w_2 a_1 (1 - a_1) = \boxed{\delta_{a_2}} w_2 \sigma'(z_1)$$

Hidden layer part also can be a matter of Just Output layer



Differention of activation function

In case of output layer

$$\frac{\partial C}{\partial w_2} = \boxed{\delta_{a_2}} a_1$$

$$\boxed{\delta_{a_2}} = (a_2 - y)\underline{a_2(1 - a_2)} = (a_2 - y)\underline{\sigma'(z_2)}$$

$$\frac{\partial C}{\partial w_1} = \boxed{\delta_{a_1}} a_0$$

$$\boxed{\delta_{a_1}} = \delta_{a_2} w_2 \underline{a_1(1 - a_1)} = \delta_{a_2} w_2 \underline{\sigma'(z_1)}$$

In case of hidden layer

$$\frac{\partial C}{\partial b_2} = \delta_{a_2} \qquad \frac{\partial C}{\partial b_1} = \delta_{a_1}$$

Contents :

1. History
2. Intro
3. Single Perceptron : Concept + Code Practice
4. XOR problem : Concept + Code Practice
5. Backpropagation (Concept Only due to time)
6. Practice : CNN Image recognition in MNIST

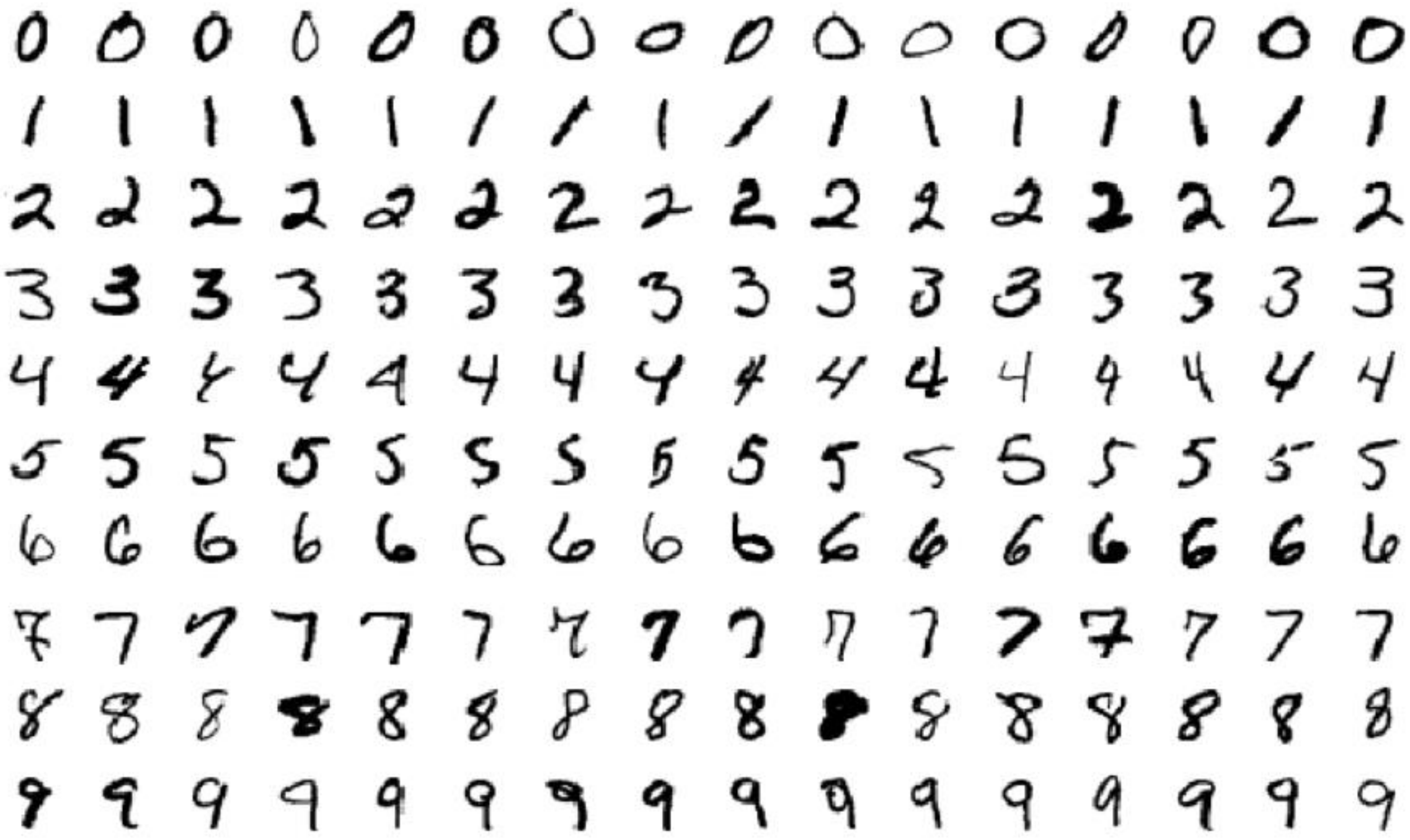
Jump~

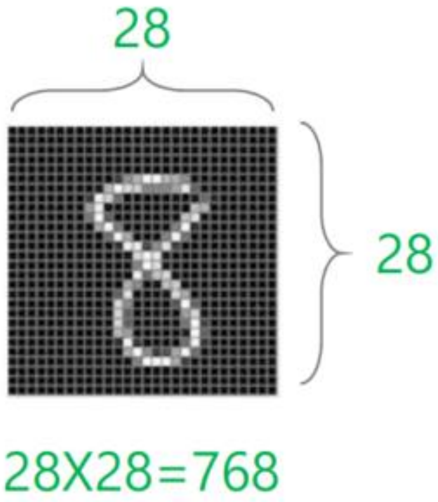
Life is Short:
Life is Short
Using Package

Tensorflow or Keras

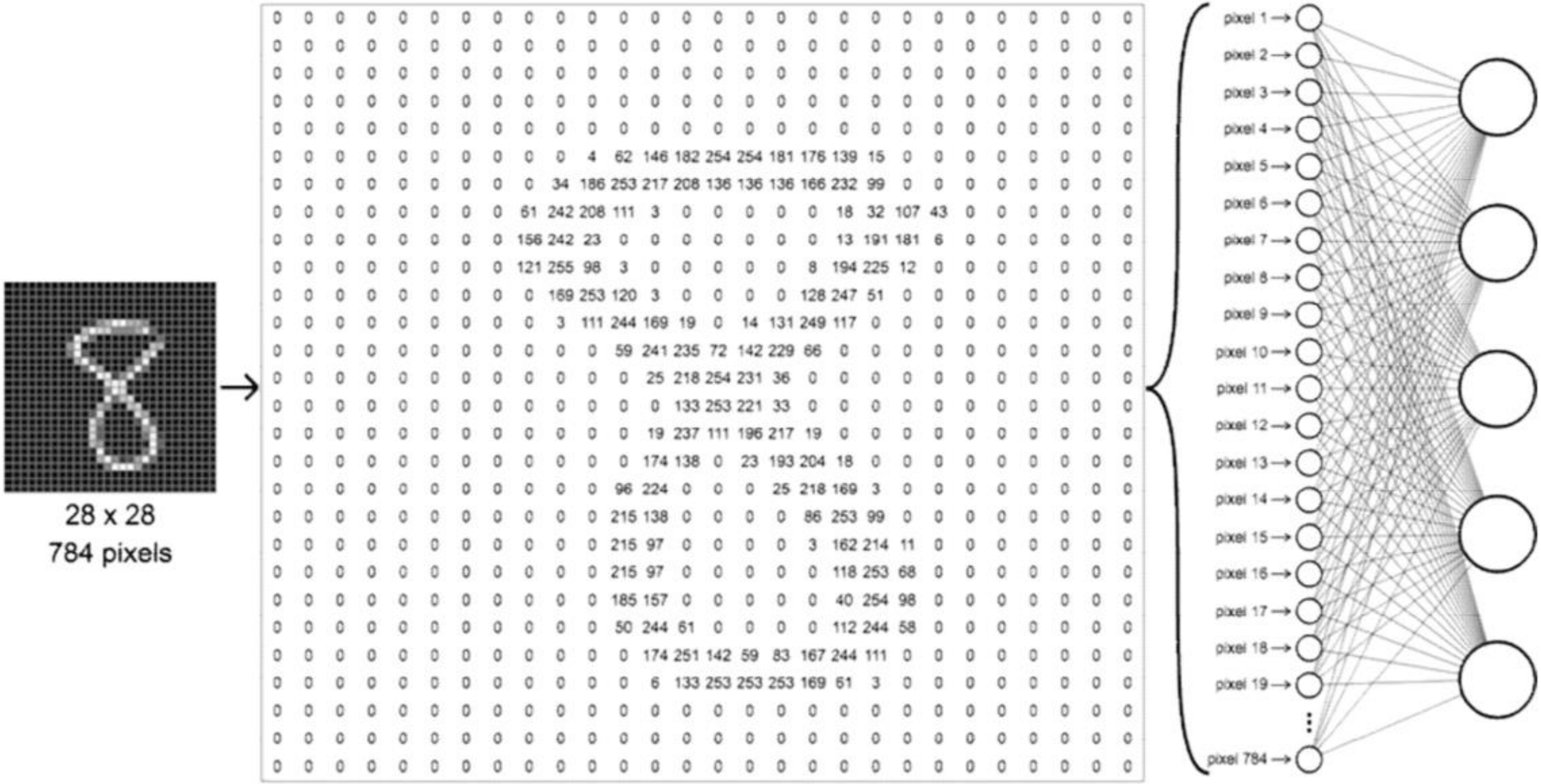
Popular Data Set for Image Recognition

MNIST

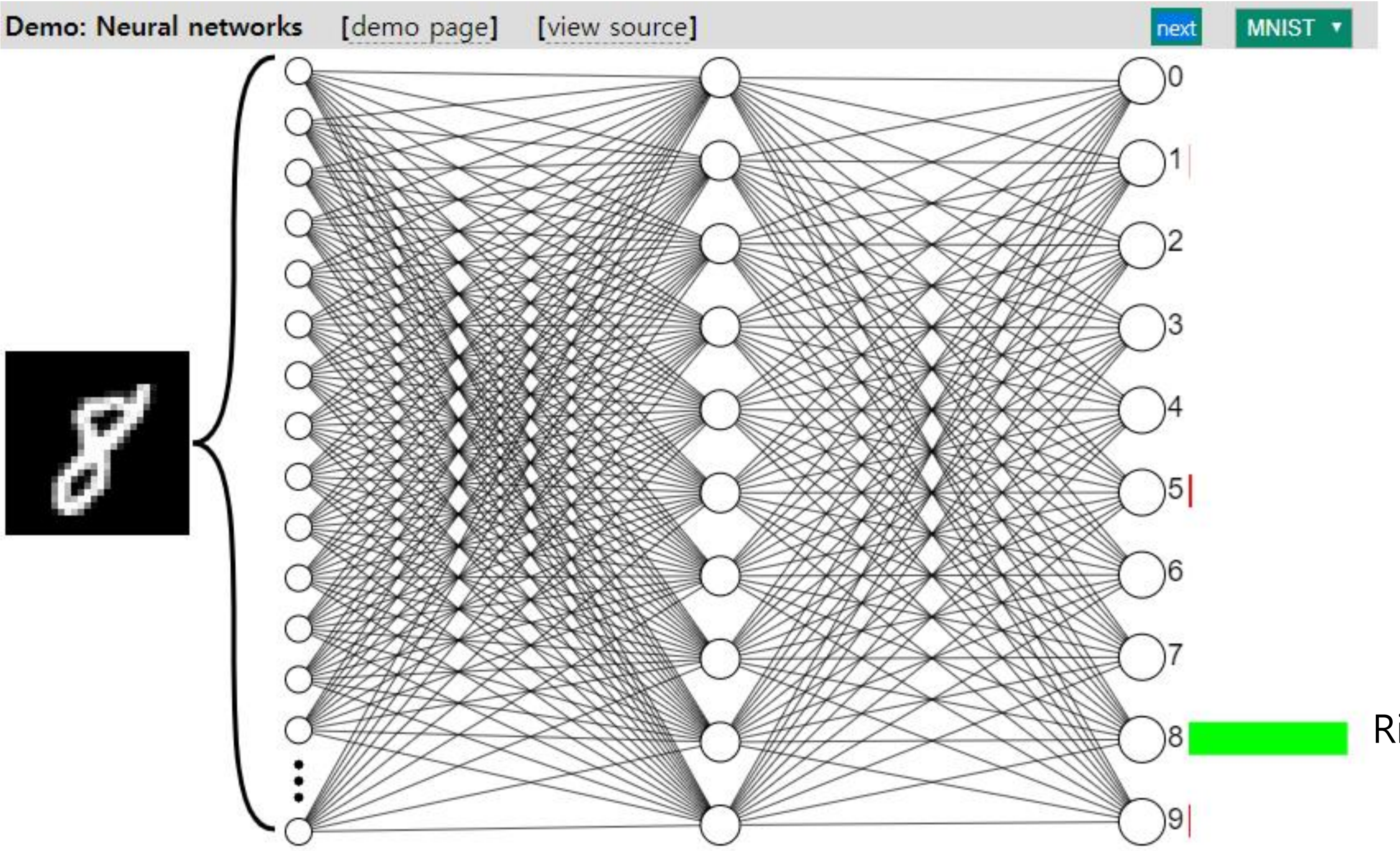




http://yann.lecun.com/exdb/mnist/

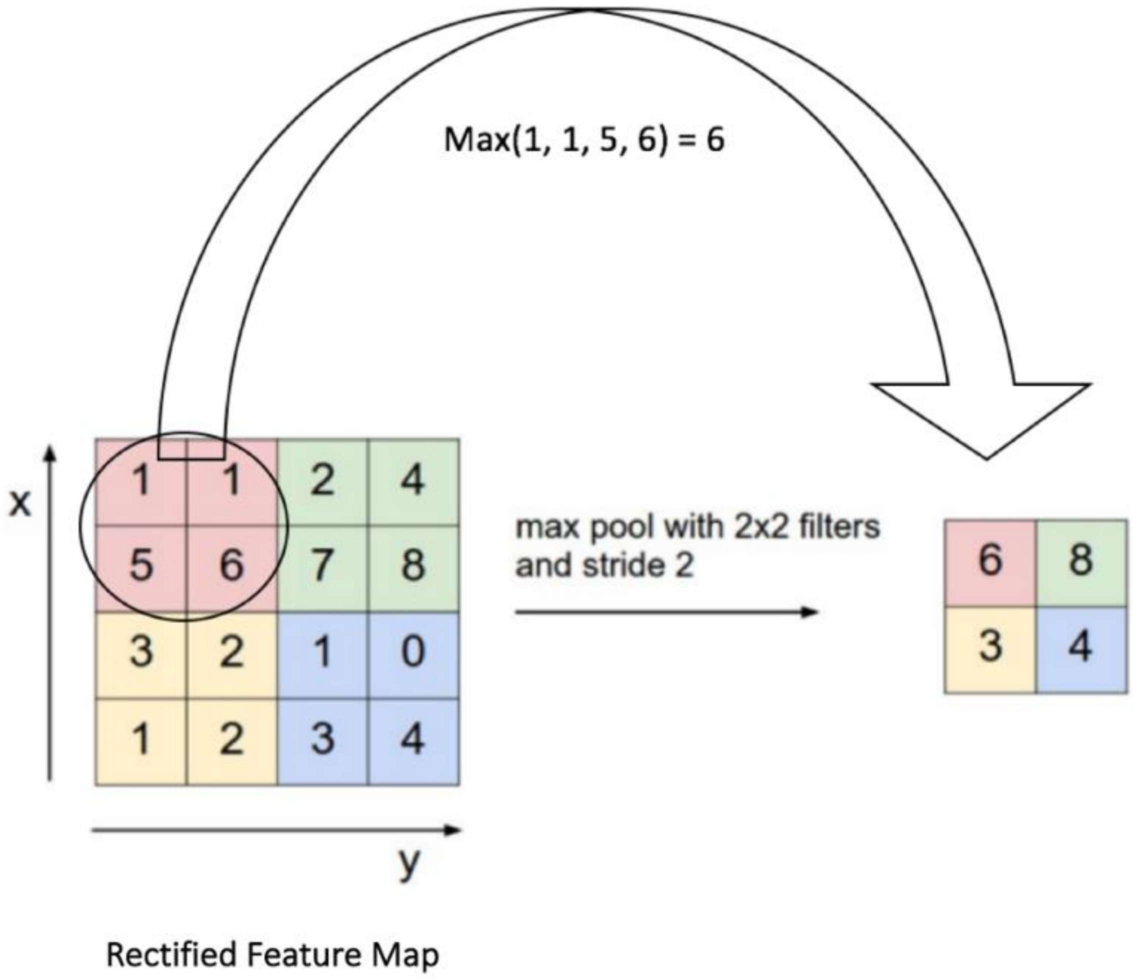


https://ml4a.github.io/ml4a/neural_networks/



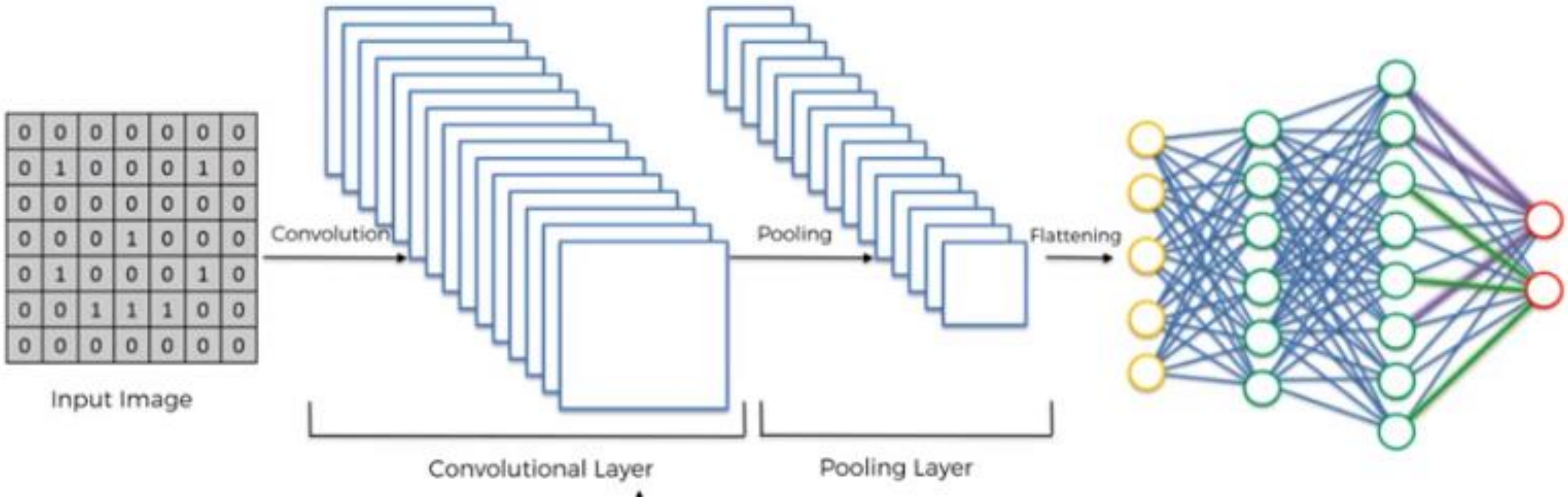
Right Guess

Hidden layer



Max-pooling

https://cdn-images-1.medium.com/max/800/0*6ED-178t3tjE0Wo6



https://cdn-images-1.medium.com/max/800/1*aAz7Nrx4IkdEViyBknpH9Q.png

Using Keras : pip install keras
Using Tensorflow: pip install tensorflow

Big step in CNN : Convolution, Polling(Subsampling), Flattening

1. Setting seed to regeneration of same outcome
2. Load data from keras (data set in keras)
3. Split the data into training set and testset
4. Making y variables as a categorial dummy

range of y = 1-10 -> changes to make 10 dummy as a D1 =1 if y=1 otherwise D1 =0

5. Set the model, key parameter is input dimension, 28*28
   mask is not explained

6. Fighting with overfitting -> drop out -> drop some node in the hidden layers
   -> Early stop

Reference

Basic Neural Network for Deep Learning, http://www.kocw.net/home/search/kemView.do?kemId=1265587
http://cs231n.github.io/neural-networks-1/
Deep learning from scratch (Book), http://www.hanbit.co.kr/store/books/look.php?p_code=B8475831198
Deep Learning for Everyone (Book), http://www.yes24.com/24/goods/57736119
jiwon Seo, Deep Learning Edu 5th chapter 2. Perceptron(2.4~)
http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf
http://yann.lecun.com/exdb/mnist/
https://cdn-images-1.medium.com/max/800/0*6ED-178t3tjE0Wo6
https://cdn-images-1.medium.com/max/800/1*aAz7Nrx4IkdEViyBknpH9Q.png