Deep Learning #2

30 Dec 2020

자율주행시스템 개발팀
신 주 석

Recap

◆ ML Routine

1) Application에 적합한 ML Method 선정: Linear Regression, Logistic Regression(Classification), etc.
2) Hypothesis 정의
3) Cost (Loss) Function 정의 => Cost(Loss)를 최소화하는 최적화 알고리즘 적용 (e.g. Gradient Descent Alg.)
4) Data 수집 => Annotation => Training & Validation => Tuning => Testing (성능 평가)

◆ Linear Regression

– **Hypothesis:** $H(x) = Wx + b$

– **Cost Function:** $cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$

◆ Logistic regression

– **Hypothesis:** $H(X) = \frac{1}{1 + e^{-W^T X}}$

– **Cost Function:**

$$cost(W) = \frac{1}{m}\sum \quad c(H(x), y)$$

$$c(H(x), y) = \begin{cases} -log(H(x)) & : y = 1 \\ -log(1 - H(x)) & : y = 0 \end{cases}$$

$$\boldsymbol{c(H(x), y) = -ylog(Hx)) - (1 - y)\log(1 - H(x))}$$

$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1 - y)log(1 - H(x))$$

Recap: Cost Function of the Logistic Regression

- **Cost Function**

$$cost(W) = \frac{1}{m}\sum \quad c(H(x), y)$$

$$c(H(x), y) = \begin{cases} -log(H(x)) & : y = 1 \\ -log(1 - H(x)) & : y = 0 \end{cases}$$

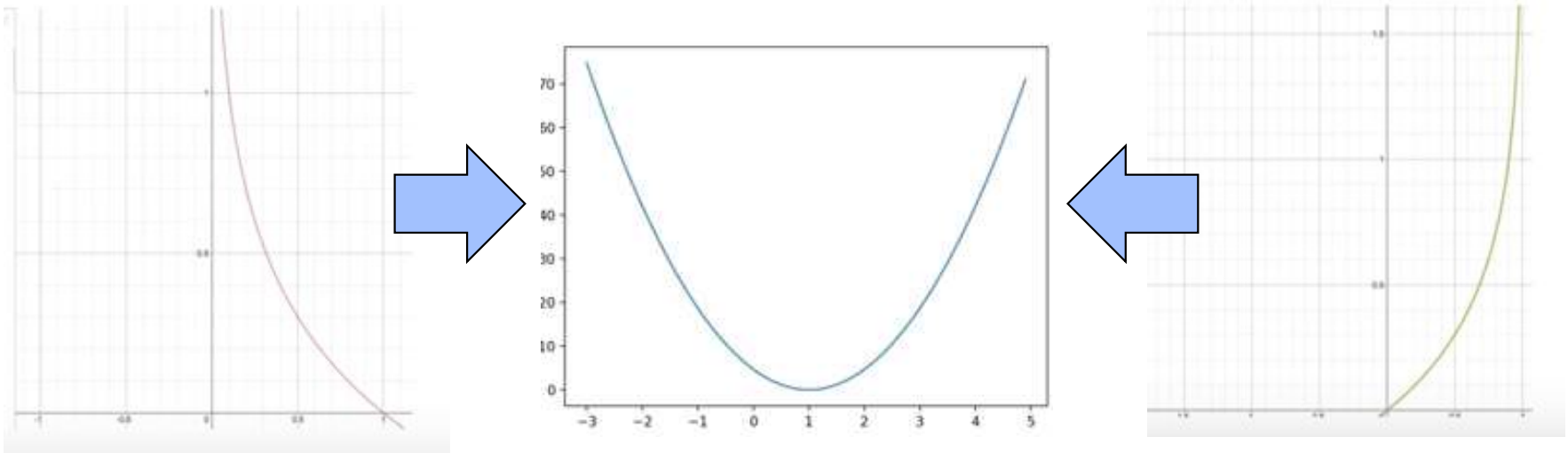
$$\boldsymbol{c(H(x), y) = -ybg\ (Hx)) - (1 - y)\mathbf{bg}\ (1 - H(x))}$$

$-log(H(x))$

$y = 1, c = -bg\ (H(x))$

$y = 0, c = -bg\ (1 - H(x))$

$-log(1 - H(x))$



Recap: Cost Function of the Logistic Regression

- **Minimize Cost: Gradient Descent Algorithm**

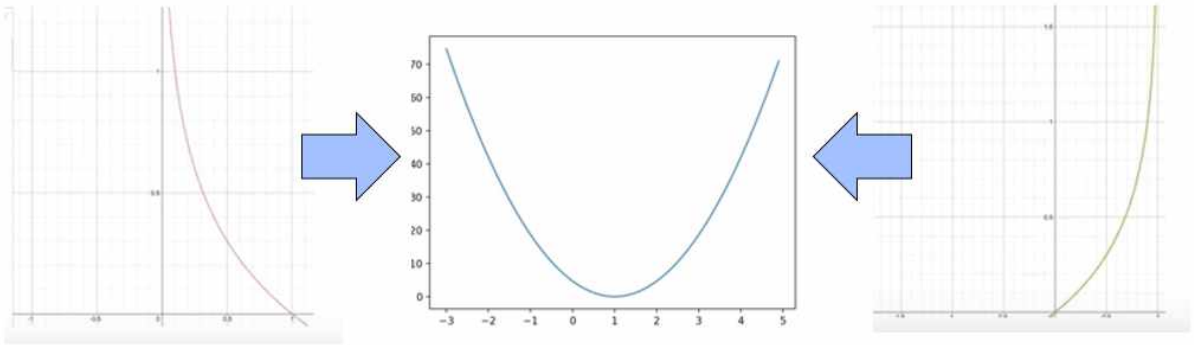
$$cost(W) = \frac{1}{m}\sum \quad c(H(x), y)$$

$$c(H(x), y) = \begin{cases} -log(H(x)) & : y = 1 \\ -log(1 - H(x)) & : y = 0 \end{cases}$$

$$\boldsymbol{c(H(x), y) = -ybg\ (Hx)) - (1 - y)\mathbf{bg}\ (1 - H(x))}$$

$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1 - y)log(1 - H(x))$$

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$



```
# cost function
cost = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis)))
```

Recap: Logistic Regression: 실습

◆ Classifying acceptance at a University (read file)

– **Data_logistic.txt**

» Test Dataset: Last 10 rows

» Train Dataset: n-10 rows

» Design Nodes

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

Hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1-y)log(1-H(x))$$

```
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(20001):
        cost_val, _ = sess.run([cost, train], feed_dict={X: train_data_x, Y: train_data_y})
        if i % 2000 == 0:
            print(i, cost_val)

    acc = sess.run(accuracy, feed_dict={X: test_data_x, Y: test_data_y})
```

```
0 0.806795
2000 0.238209
4000 0.182931
6000 0.162146
8000 0.151287
10000 0.144676
12000 0.140276
14000 0.137172
16000 0.134888
18000 0.133155
20000 0.131809
```

```
print("Accuracy: ", acc)
```

◆ Classifying diabetes

– **Data-03-diabetes.csv (Test Dataset: Last 20 rows)**

```
data = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
```

```
print("Accuracy: ", acc)
```

```
Accuracy:  0.85
```

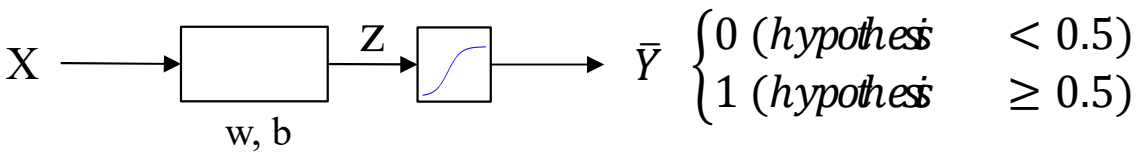
- **Logistic regression**

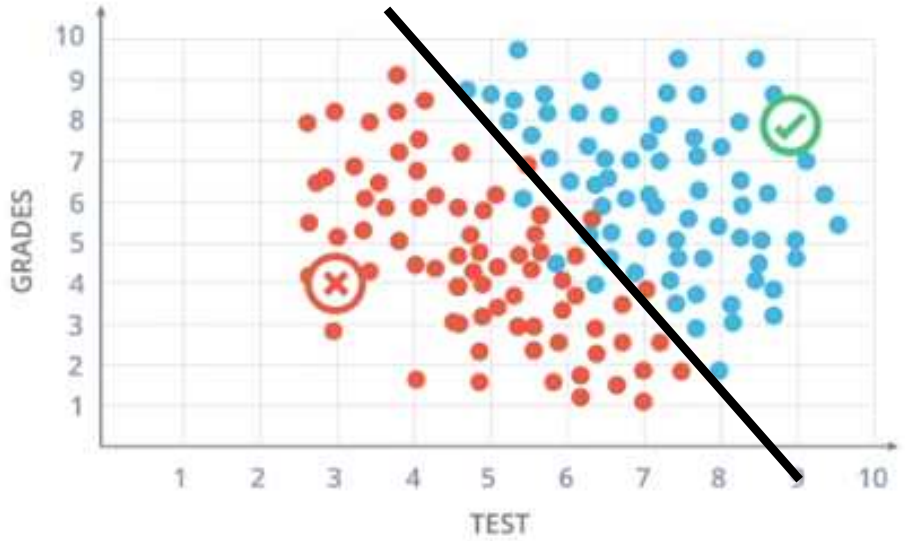
$$H_L(x) = W\ x + b$$

$$Z = H_L(x)$$

$$g(Z) = \frac{1}{1 + e^{-z}}$$

$$H_{LR} = g(H_L(x))$$

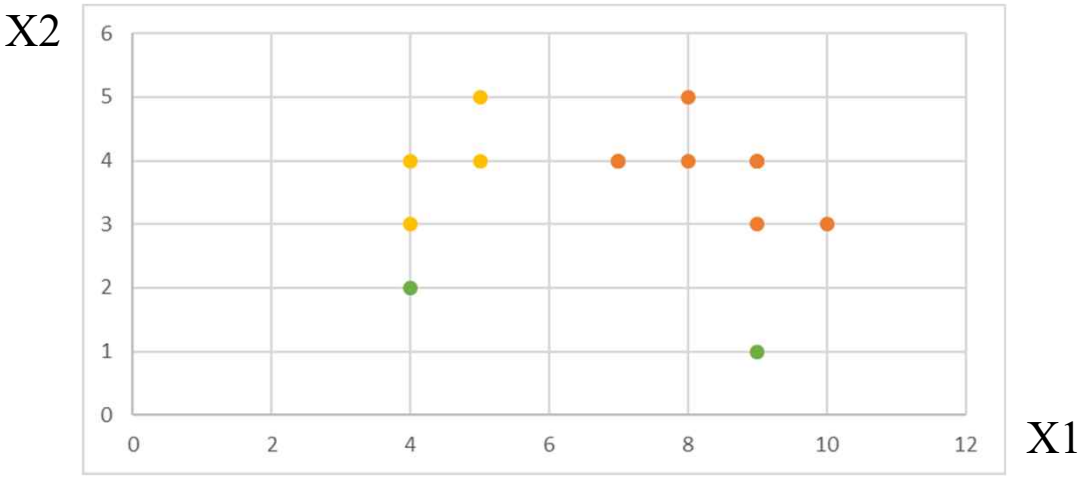




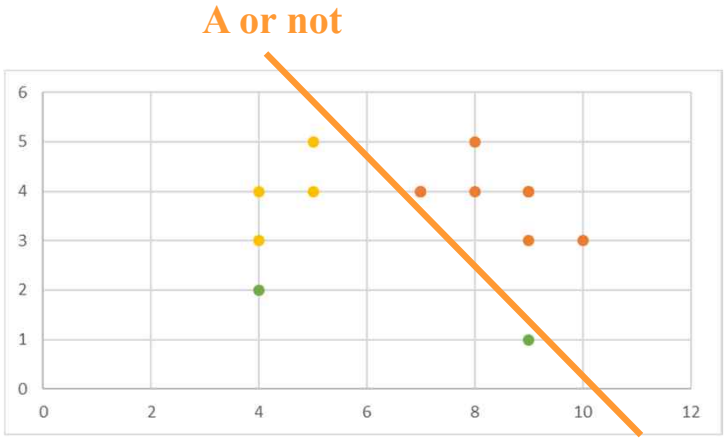
Multinomial Classification

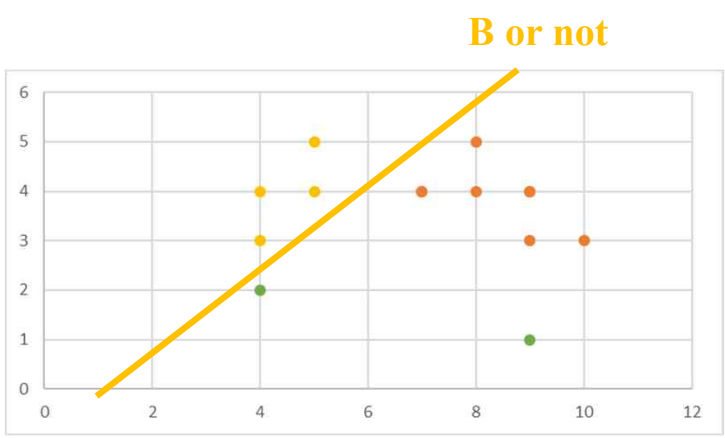
- **Example of the Multinomial Classification**

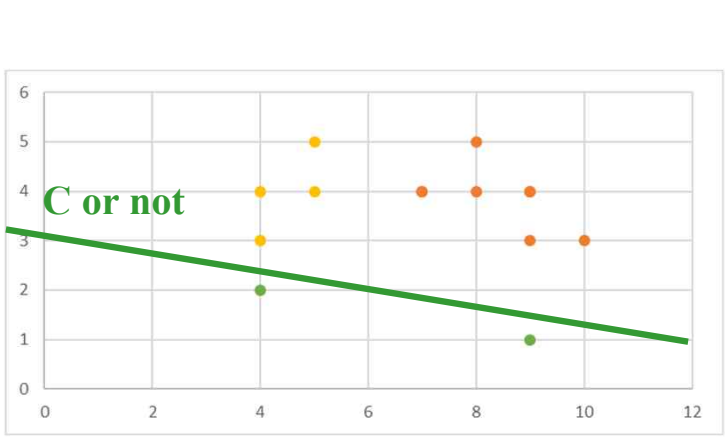
x1 (test)	x2 (attend)	y (grade)
10	3	A
9	4	A
9	4	A
9	3	A
8	4	A
8	5	A
7	4	A
7	4	A
5	5	B
5	4	B
4	4	B
4	3	B
4	2	C
9	1	C

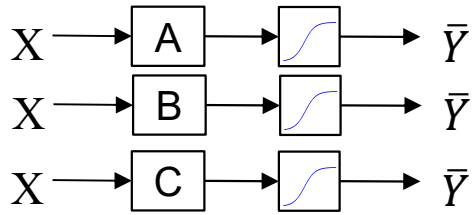


- **Combines the binary Classification**







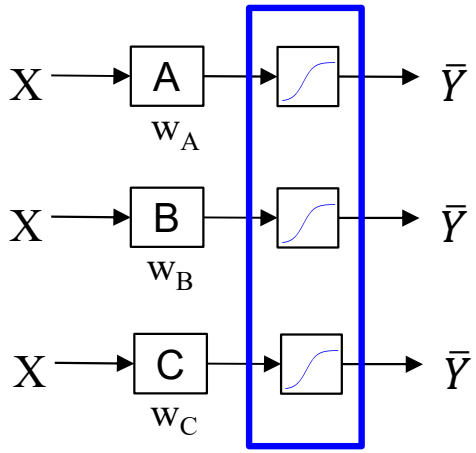


Multinomial Classification

$$[W_{A1} \quad W_{A2}] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [W_{A1}x_1 + W_{A2}x_2]$$

$$[W_{B1} \quad W_{B2}] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [W_{B1}x_1 + W_{B2}x_2]$$

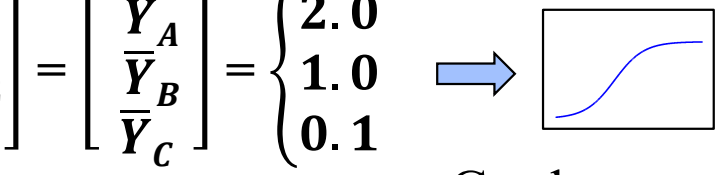
$$[W_{C1} \quad W_{C2}] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [W_{C1}x_1 + W_{C2}x_2]$$



$$\begin{bmatrix} W_{A1} & W_{A2} \\ W_{B1} & W_{B2} \\ W_{C1} & W_{C2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + W_{A2}x_2 \\ W_{B1}x_1 + W_{B2}x_2 \\ W_{C1}x_1 + W_{C2}x_2 \end{bmatrix} = \begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{Bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{Bmatrix} \Rightarrow \boxed{} \Rightarrow \begin{Bmatrix} 0.88 \\ 0.73 \\ 0.52 \end{Bmatrix}$$

- **Softmax**

$$\begin{bmatrix} W_{A1} & W_{A2} \\ W_{B1} & W_{B2} \\ W_{C1} & W_{C2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + W_{A2}x_2 \\ W_{B1}x_1 + W_{B2}x_2 \\ W_{C1}x_1 + W_{C2}x_2 \end{bmatrix} = \begin{bmatrix} \overline{Y}_A \\ \overline{Y}_B \\ \overline{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases}$$



$$\begin{cases} 0.88 \\ 0.73 \\ 0.52 \end{cases}$$

Grade

$$\begin{bmatrix} \overline{Y}_A \\ \overline{Y}_B \\ \overline{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases}$$

Softmax

Score → Probability

$$\begin{cases} 0.096 \\ 0.26 \\ 0.64 \end{cases}$$

A

B

C

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$

Why we use exponential term?

- **Cross-Entropy (Cost Function)**

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$ 예측 값

실제 값(Label data) Y

0.096
0.24
0.64

$D(S, Y) = -\sum_i Y_i * bg\ (Si)$

element wise Multiplication

Grade	
0.0	A
0.0	B
1.0	C

$$Cost = \frac{1}{N}\sum_i D\ (S_i, Y_i)$$

$$-\sum_i Y_i * bg\ (\bar{Y}_i) = \sum_i Y_i * -bg\ (\bar{Y}_i)$$

$$Y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\bar{Y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix} * \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$$

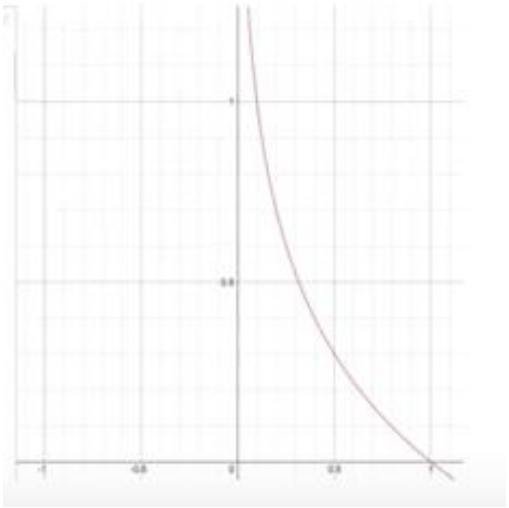
$$\bar{Y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \infty$$

$$Y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\bar{Y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$$

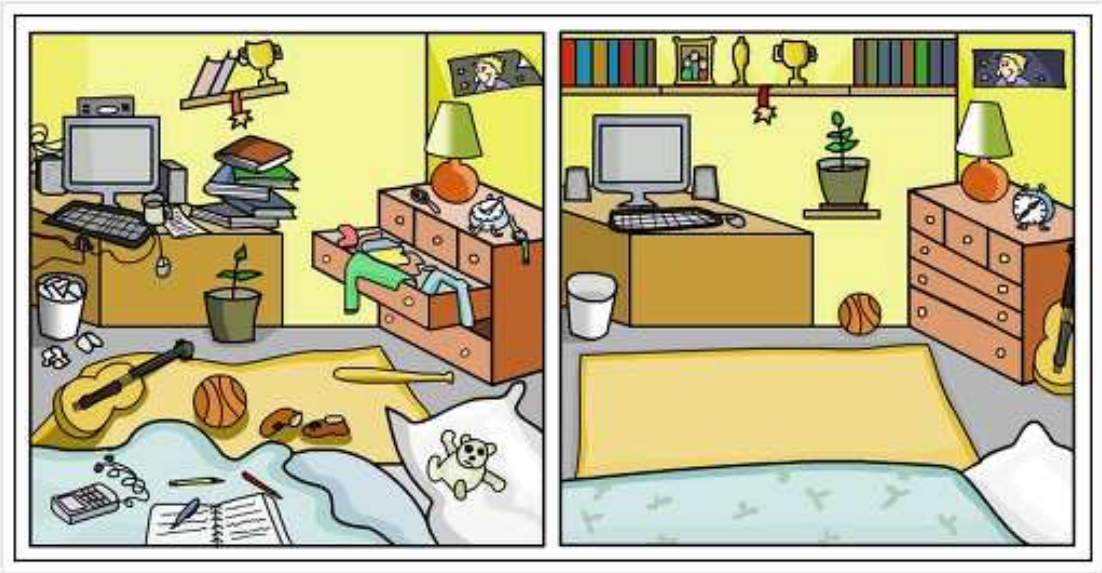
$$\bar{Y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \infty$$

$-bg\ (x)$



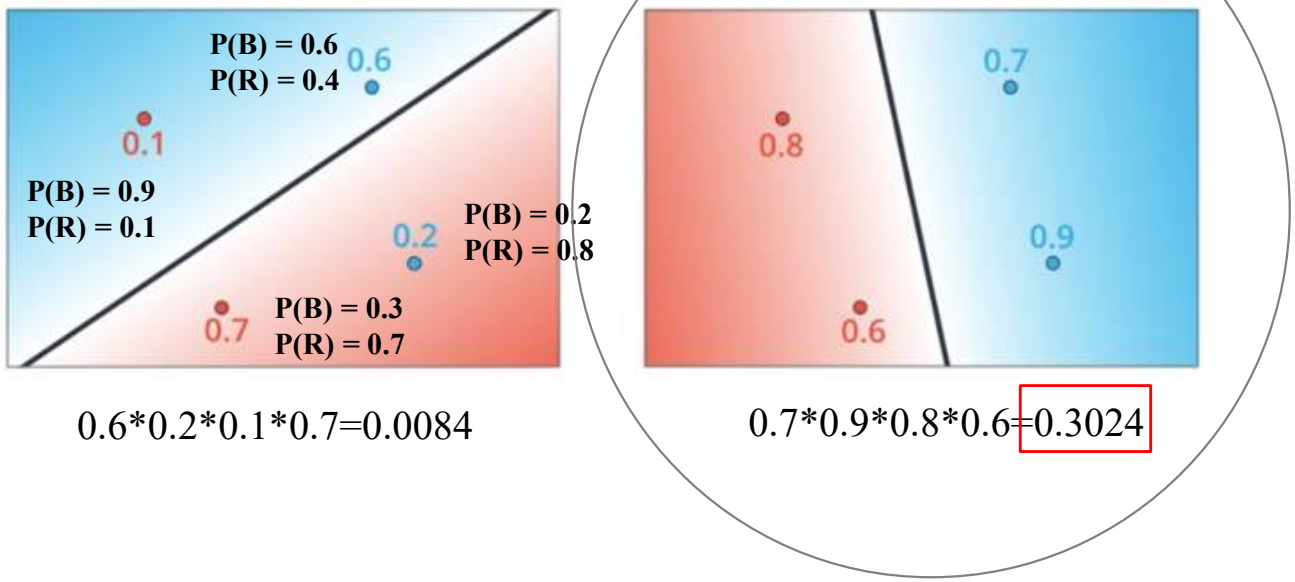
Multinomial Classification

- **Cross-Entropy**



- **Cross-Entropy**

(어느 모델이 좋은 모델인가?)



$0.6*0.2*0.1*0.7=0.0084$

$0.7*0.9*0.8*0.6=0.3024$

Products are bad, but sums are good!!

$\log(0.6) + \log(0.2) + \log(0.1) + \log(0.7)$
$= -0.22 + -0.69 + -1 + -0.15$

$-\log(0.6) - \log(0.2) - \log(0.1) - \log(0.7)$
$= 0.22 + 0.69 + 1 + 0.15 = \mathbf{2.0757}$

Cross Entropy가 낮을수록 좋은 모델

$-\log(0.7) - \log(0.9) - \log(0.8) - \log(0.6)$
$= 0.15 + 0.04 + 0.09 + 0.22 = \mathbf{0.5194}$

Multinomial Classification

- **Classifying type of the animal**

$$\begin{bmatrix} W_{A1} & W_{A2} \\ W_{B1} & W_{B2} \\ W_{C1} & W_{C2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + W_{A2}x_2 \\ W_{B1}x_1 + W_{B2}x_2 \\ W_{C1}x_1 + W_{C2}x_2 \end{bmatrix} = \begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{Bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{Bmatrix}$$

$$\begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{Bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{Bmatrix} \Rightarrow \text{Softmax} \Rightarrow \begin{Bmatrix} 0.65 \\ \vdots \\ 0.24 \\ \vdots \\ 0.11 \end{Bmatrix}$$

Score → Probability

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$

$$D(S, L) = -\sum_i L_i * bg\ (Si) \qquad Cost = \frac{1}{N}\sum_i D\ (S_i, Li)$$

0		1	0	0	0	0	0	0
1		0	1	0	0	0	0	0
2		0	0	1	0	0	0	0
3		0	0	0	1	0	0	0
4		0	0	0	0	1	0	0
5		0	0	0	0	0	1	0
6		0	0	0	0	0	0	1

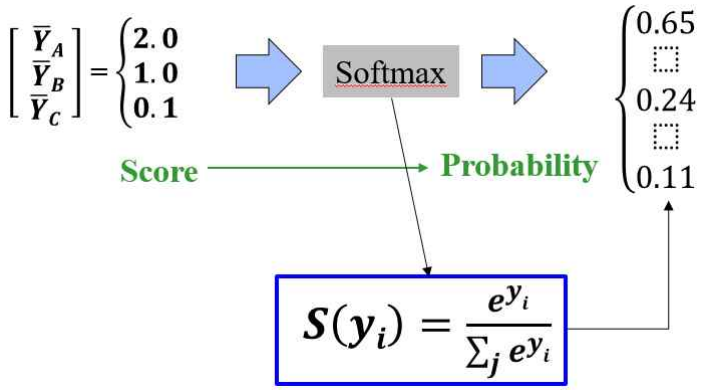
"One-hot encoding"

```
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data.shape, y_data.shape)

classes = 7  # 0 ~ 6

X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1])
Y_one_hot = tf.one_hot(Y, classes)  # N x 1 x 7
print("one_hot_encoding", Y_one_hot)
Y_one_hot = tf.reshape(Y_one_hot, [-1, classes])  # N x 7
print("reshape: one_hot_encoding", Y_one_hot)

W = tf.Variable(tf.random_normal([16, classes]), name='weight')
b = tf.Variable(tf.random_normal([classes]), name='bias')

logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)

# Cross entropy cost/loss
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
                                                        labels=Y_one_hot)
cost = tf.reduce_mean(cross_entropy)
opt = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

prediction = tf.argmax(hypothesis, 1)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
    for i in range(2000):
        sess.run(opt, feed_dict={X: x_data, Y: y_data})
        if i % 100 == 0:
            loss, acc = sess.run([cost, accuracy], feed_dict={
                                 X: x_data, Y: y_data})
            print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(
                i, loss, acc))

    print("Training Done!!!")
    pred = sess.run(prediction, feed_dict={X: x_data})
    for p, y in zip(pred, y_data.flatten()):
        print("[{}] Prediction: {} True Y: {}".format(p == int(y), p, int(y)))
```

- **Classifying digit using MNIST Dataset**

$$\begin{bmatrix} W_{A1} & W_{A2} \\ W_{B1} & W_{B2} \\ W_{C1} & W_{C2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + W_{A2}x_2 \\ W_{B1}x_1 + W_{B2}x_2 \\ W_{C1}x_1 + W_{C2}x_2 \end{bmatrix} = \begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{Bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{Bmatrix}$$



$$D(S, L) = -\sum_i L_i * bg\ (Si) \qquad Cost = \frac{1}{N}\sum_i D\ (S_i, Li)$$

```python
import tensorflow as tf
import random
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, classes])

W = tf.Variable(tf.random_normal([784, classes]))
b = tf.Variable(tf.random_normal([classes]))

hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# adjust learning_rate
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
opt = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

MINIST Dataset

55,000	train-images-idx3-ubyte.gz:	training set images (9912422 bytes)
	train-labels-idx1-ubyte.gz:	training set labels (28881 bytes)
10,000	t10k-images-idx3-ubyte.gz:	test set images (1648877 bytes)
	t10k-labels-idx1-ubyte.gz:	test set labels (4542 bytes)

Multinomial Classification

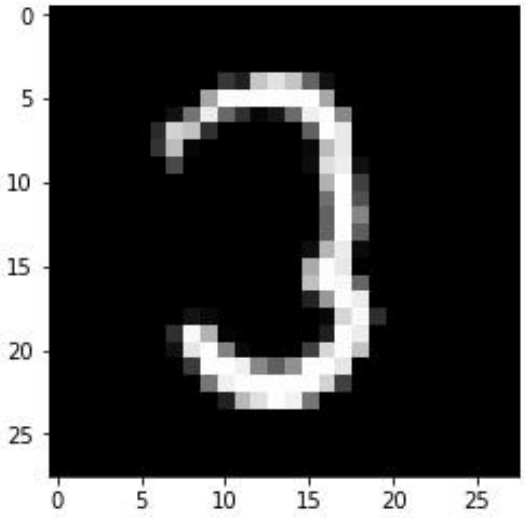
◆ Classifying digit using MNIST Dataset

In the neural network terminology:

- one **epoch** = one forward pass and one backward pass of *all* the training examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

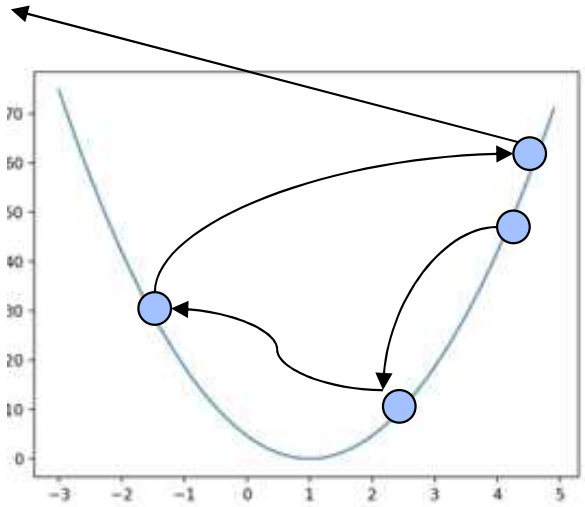
Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

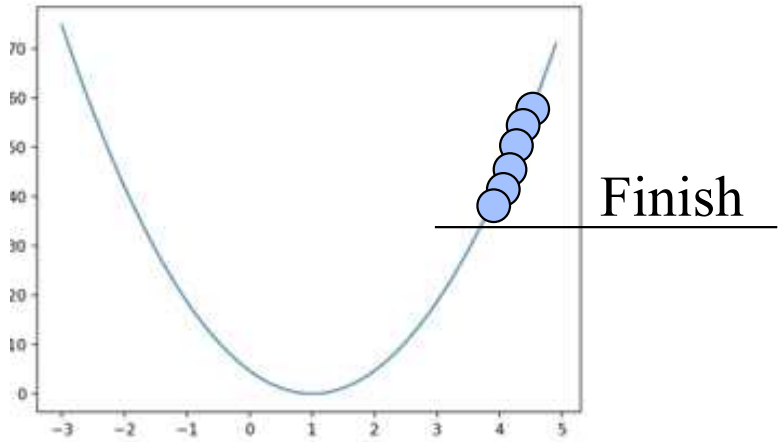
```
Training Done!!!!
Accuracy:  0.8906
Label:  [3]
Prediction:  [3]
```




```python
# Test model
correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

# parameters (modify this)
training_epochs = 10
batch_size = 256

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0
        iteration = int(mnist.train.num_examples / batch_size)

        for i in range(iteration):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            c, _ = sess.run([cost, opt], feed_dict={
                            X: batch_xs, Y: batch_ys})
            avg_cost += c / iteration

        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

    print("Training Done!!!!")

    # Test the model using test sets
    print("Accuracy: ", sess.run(accuracy, feed_dict={
          X: mnist.test.images, Y: mnist.test.labels}))

    # Get one and predict
    r = random.randint(0, mnist.test.num_examples - 1)
    print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
    print("Prediction: ", sess.run(
        tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

    plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), 'gray')
    plt.show()
```

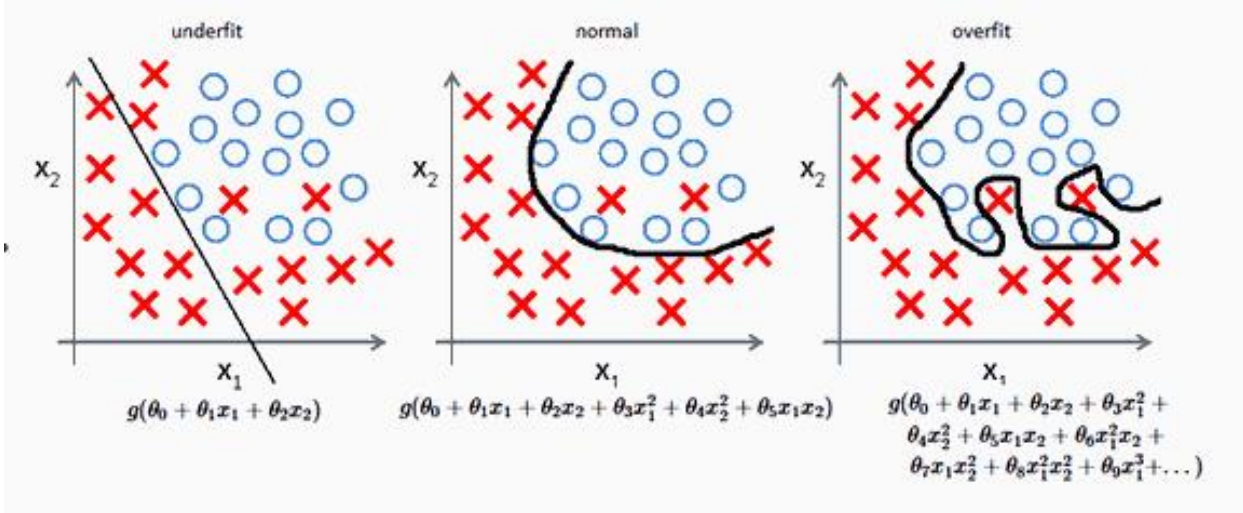
- **Learning rate**
  - **Large Learning rate: Overshooting**
  - **Small Learning rate: takes too long time, stops at local minimum**





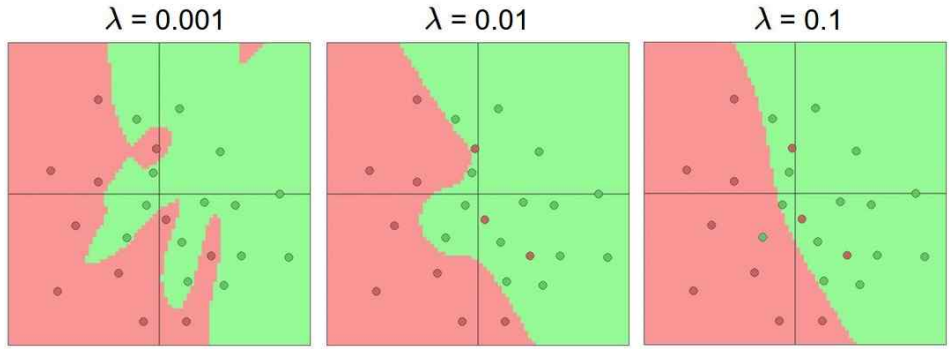
Solution:
- Observe the cost function

- **Overfitting**
  - Our model is very good with training data set
  - Not good at test dataset or in real use



Solution:

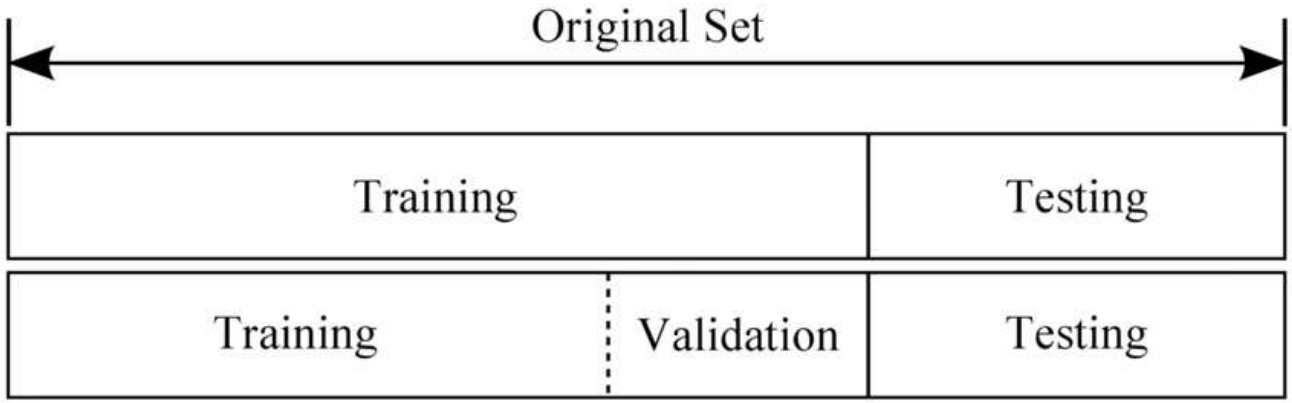
- **More Training Data**
- **Reduce the number of features**
- **Regularization**



$$min_\theta \frac{1}{2m} \left[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

http://mlwiki.org/index.php/Overfitting

- **Dataset 구성**

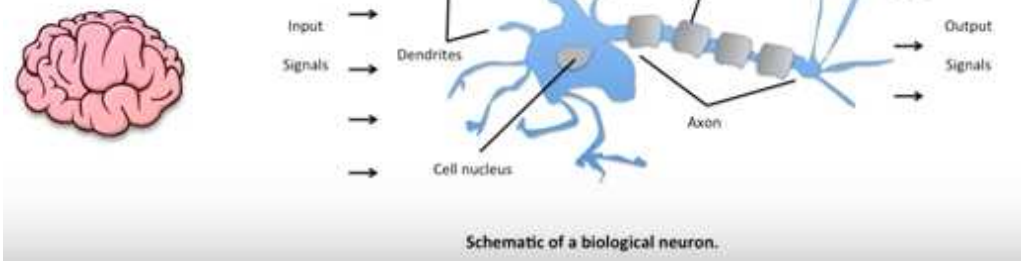


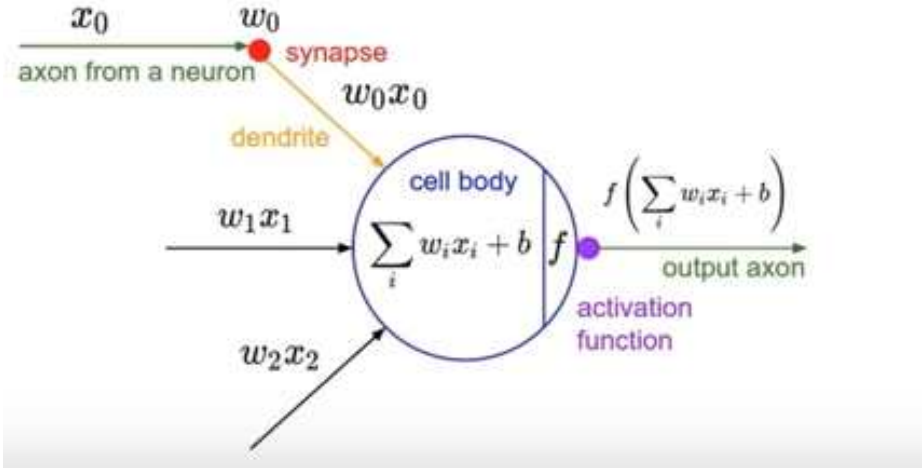
MINIST Dataset

55,000
10,000

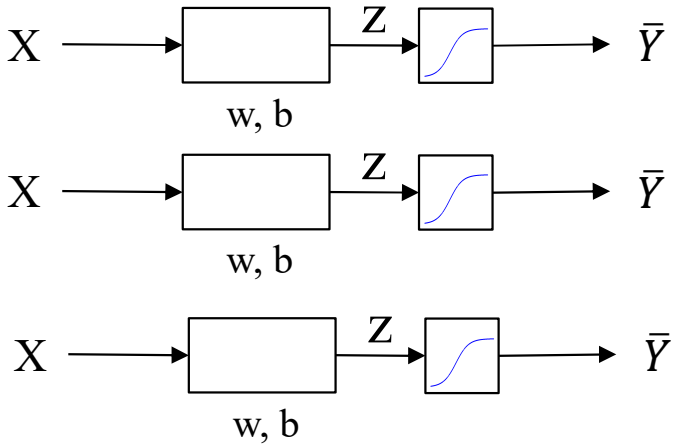
train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

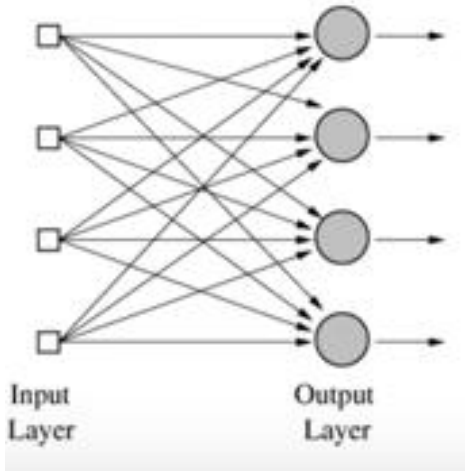
- **History**
  - **Neural Network (기계가 인간처럼 생각할 수 없을까?)**





  - **Logistic regression**

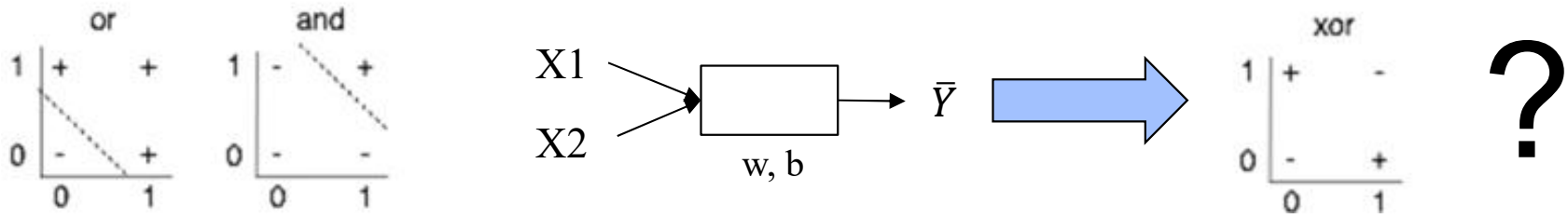




- **History**
  - **Dr. Frank (The New York Times, July 08, 1958)**

> "The Navy revealed the embryo of an electronic computer today that **it expects will be able to walk, talk, see, write, reproduce itself** an be conscious of its existence … Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers"

  - **Simple AND/OR problem: Linearly separable**



http://query.nytimes.com/gst/abstract.html?res=9D01E4D8173DE53BBC4053DFB1668383649EDE

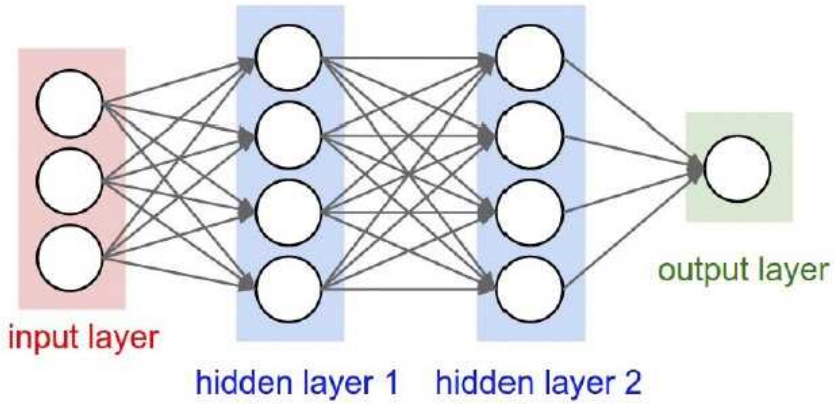
- **History**
  - **Dr. Minsky**

Perceptrons (1969)

by Marvin Minsky, founder of the MIT AI Lab

- We need to use MLP, multilayer perceptrons (multilayer neural nets)
- No one on earth had found a viable way to train MLPs good enough to learn such simple functions.

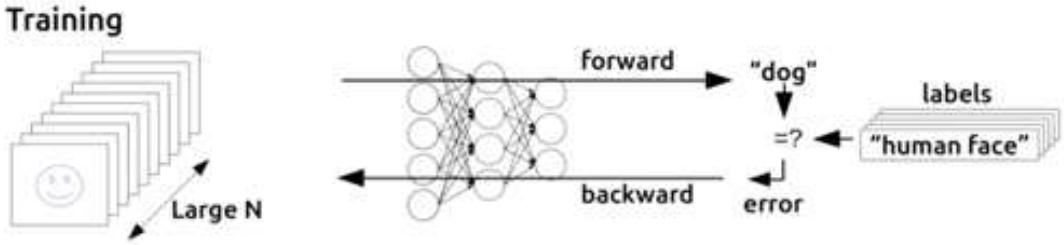
"No one on earth had found a viable way to train*"



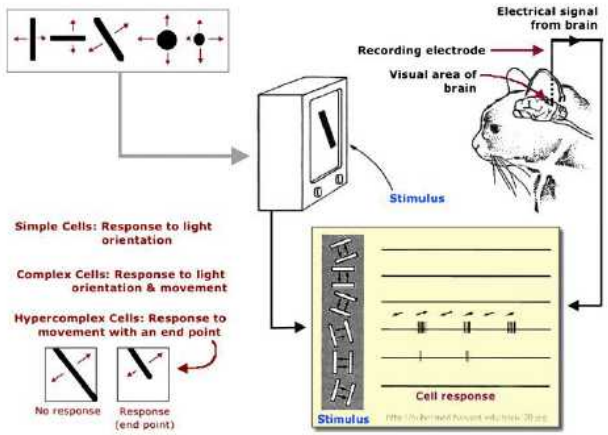
*Marvin Minsky, 1969

- **History**
  - **Backpropagation**

(1974, 1982 by Paul Werbos, 1986 by Hinton)

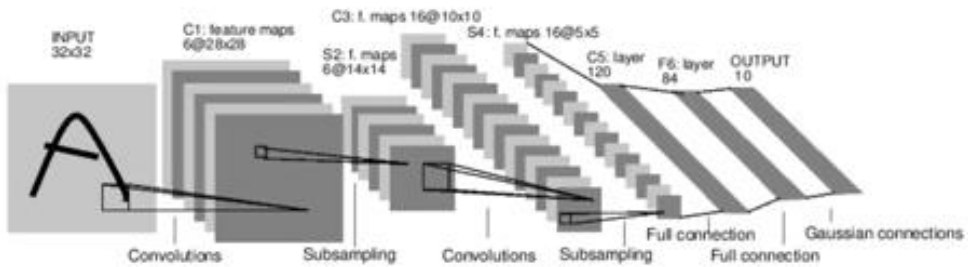


Neural Network
=> Deep Learning



LeCun, 2006

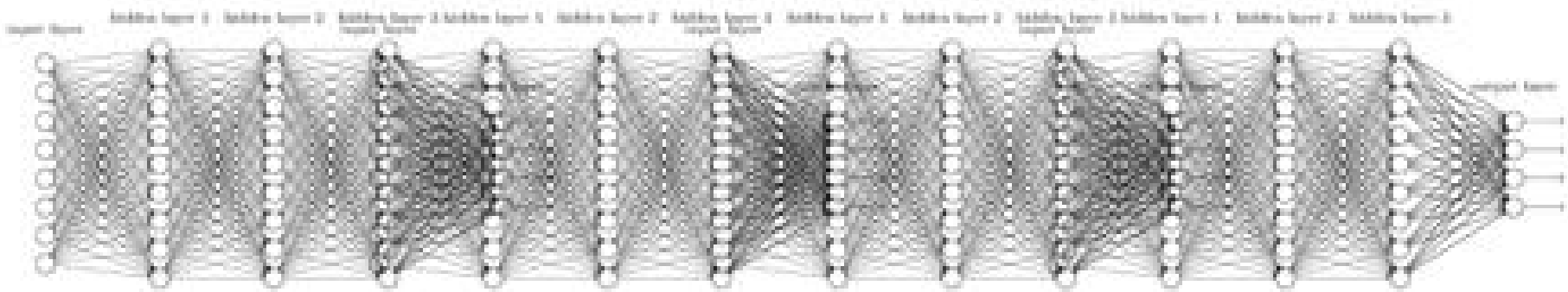
CNN: Convolutional Neural Network



https://devblogs.nvidia.com/parallelforall/inference-next-step-gpu-accelerated-deep-learning/

- **History**
  - **A BIG Problem**

- Backpropagation just did not work well for normal neural nets with many layers
- Other rising machine learning algorithms: SVM, RandomForest, etc.
- **1995** "Comparison of Learning Algorithms For Handwritten Digit Recognition" by LeCun et al. found that this new approach worked better



http://neuralnetworksanddeeplearning.com/chap6.html

- **History**
  - **CIFAR (Canadian Institute for Advanced Research)**

    - CIFAR encourages basic research ***without direct application***, was what motivated **Hinton** to move to Canada in 1987, and funded his work afterward.

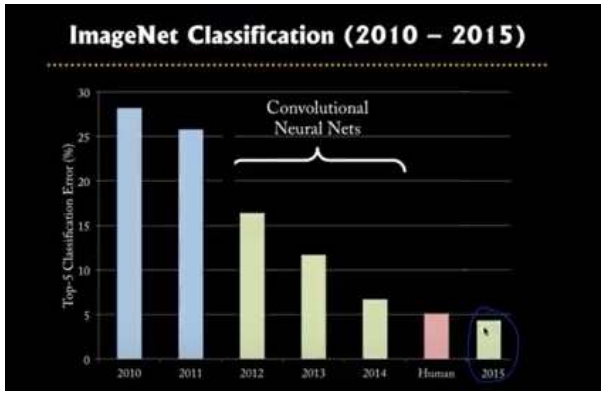
History

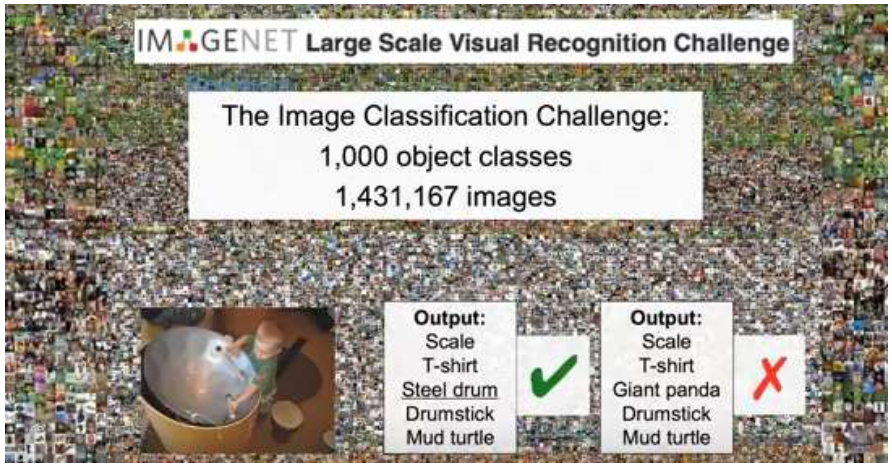
Breakthrough (in 2006 and 2007 by Hinton and Bengio)

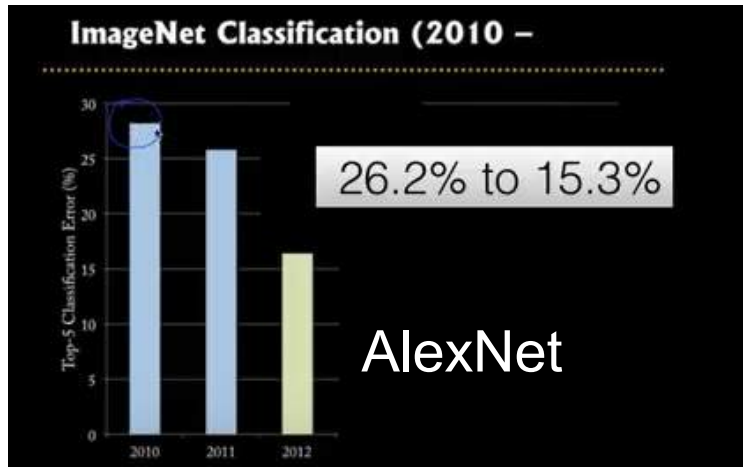
• Neural networks with many layers really could be trained well, **if the weights are initialized in a clever way rather than randomly**. (초기값만 잘 주면 학습이 가능함을 보여줌)

• **Deep machine learning methods** are more efficient for difficult problems than shallow methods.

• Rebranding to **Deep Nets, Deep Learning**

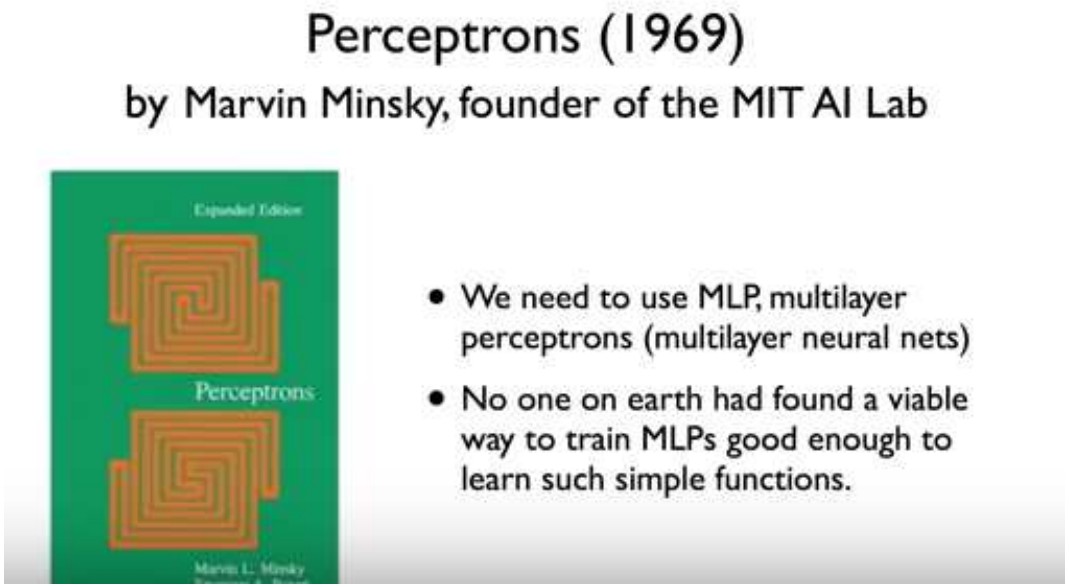


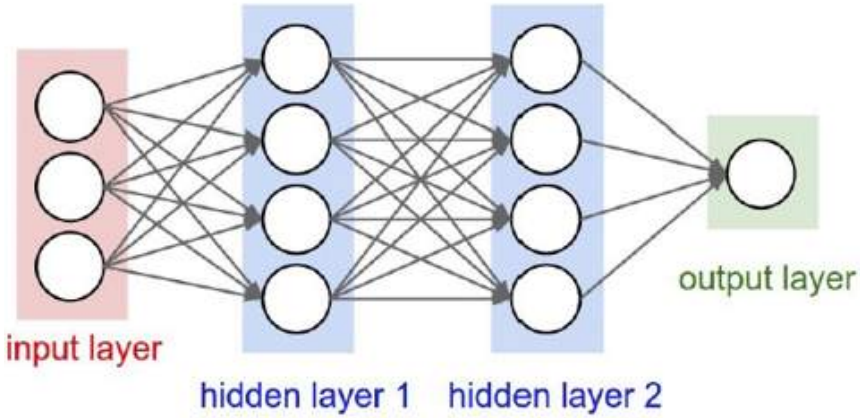




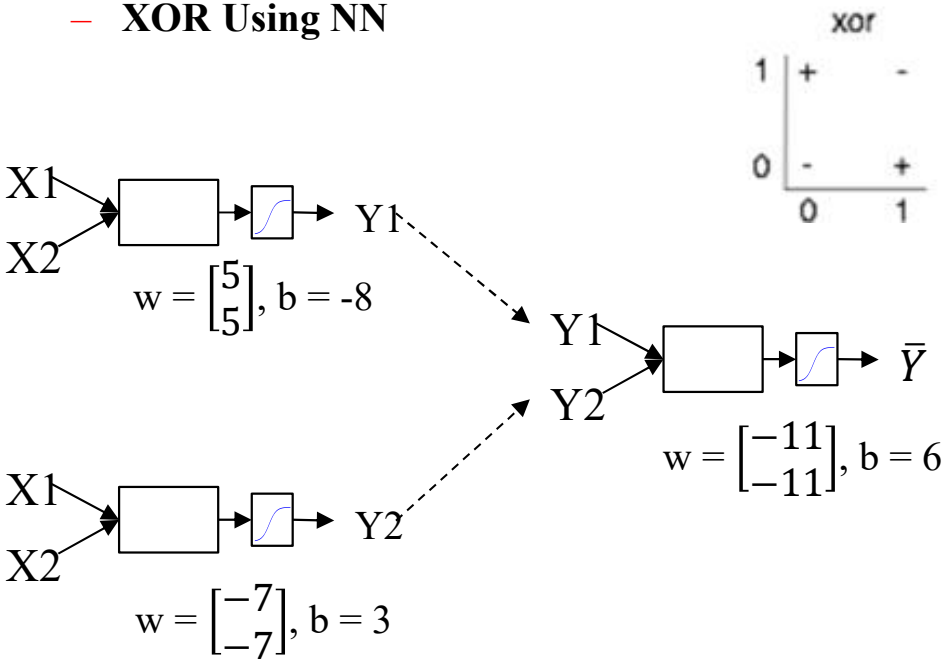
http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-4/

- **Neural Network (NN)**
  - **XOR 문제: 하나의 Logistic regression unit으로는 XOR 문제 해결 불가 (수학적으로 증명: Dr.Minsky)**
    - » 여러 개의 Logistic regression unit을 조합하여 XOR 문제는 해결 가능
    - » 각각의 복잡한 네트워크에 포함되어 있는 Weight와 bias를 학습하는 것은 불가능

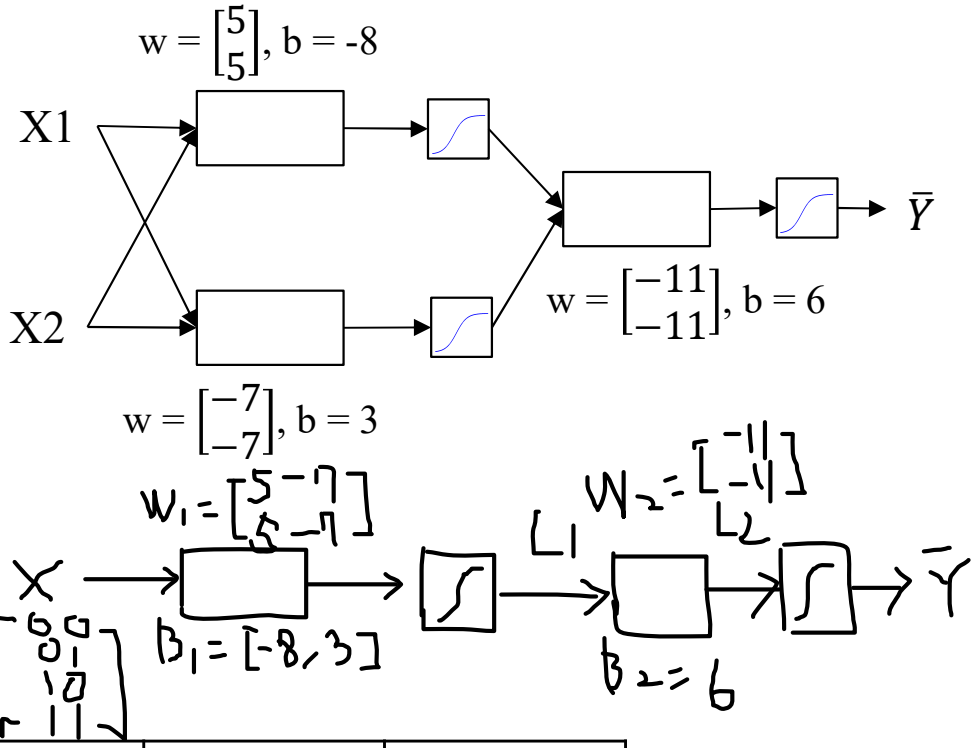




- **Neural Network (NN)**
  - **XOR Using NN**



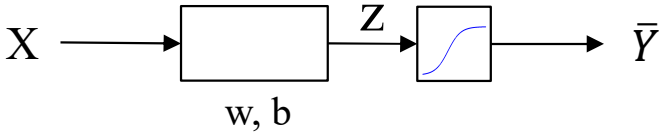
Simple Neural Network Forward Propagation



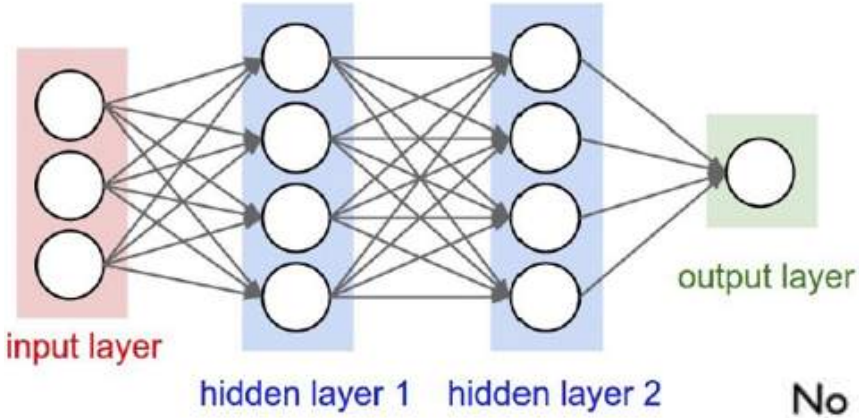
$$g(z) = \frac{1}{(1+e^{-z})}$$

x1	x2	y1	y2	$\bar{Y}$	XOR
0	0	Sigmoid(-8) = 0 (0.000335)	Sigmoid(3) = 1 (0.9525)	Sigmoid(-5) = 0 (0.006693)	0
0	1	Sigmoid(-3) = 0 (0.0474)	Sigmoid(-4) = 0 (0.017)	Sigmoid(6) = 1 (0.9975)	1
1	0	Sigmoid(-3) = 0 (0.0474)	Sigmoid(-4) = 0 (0.017)	Sigmoid(6) = 1 (0.9975)	1
1	1	Sigmoid(2) = 1 (0.88)	Sigmoid(-11) = 0 (1.67E-05)	Sigmoid(-5) = 0 (0.006693)	0

- **Neural Network (NN)**
  - 어떻게 학습을 시킬 수 있을까? => Gradient Descent Algorithm



Logistic Regression, Multinomial Classification

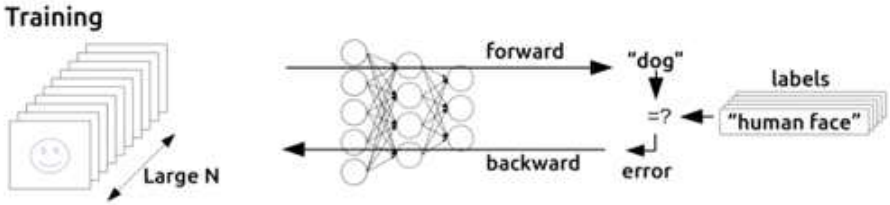


Gradient Descent
=> 미분이 복잡하고 어려워 짐

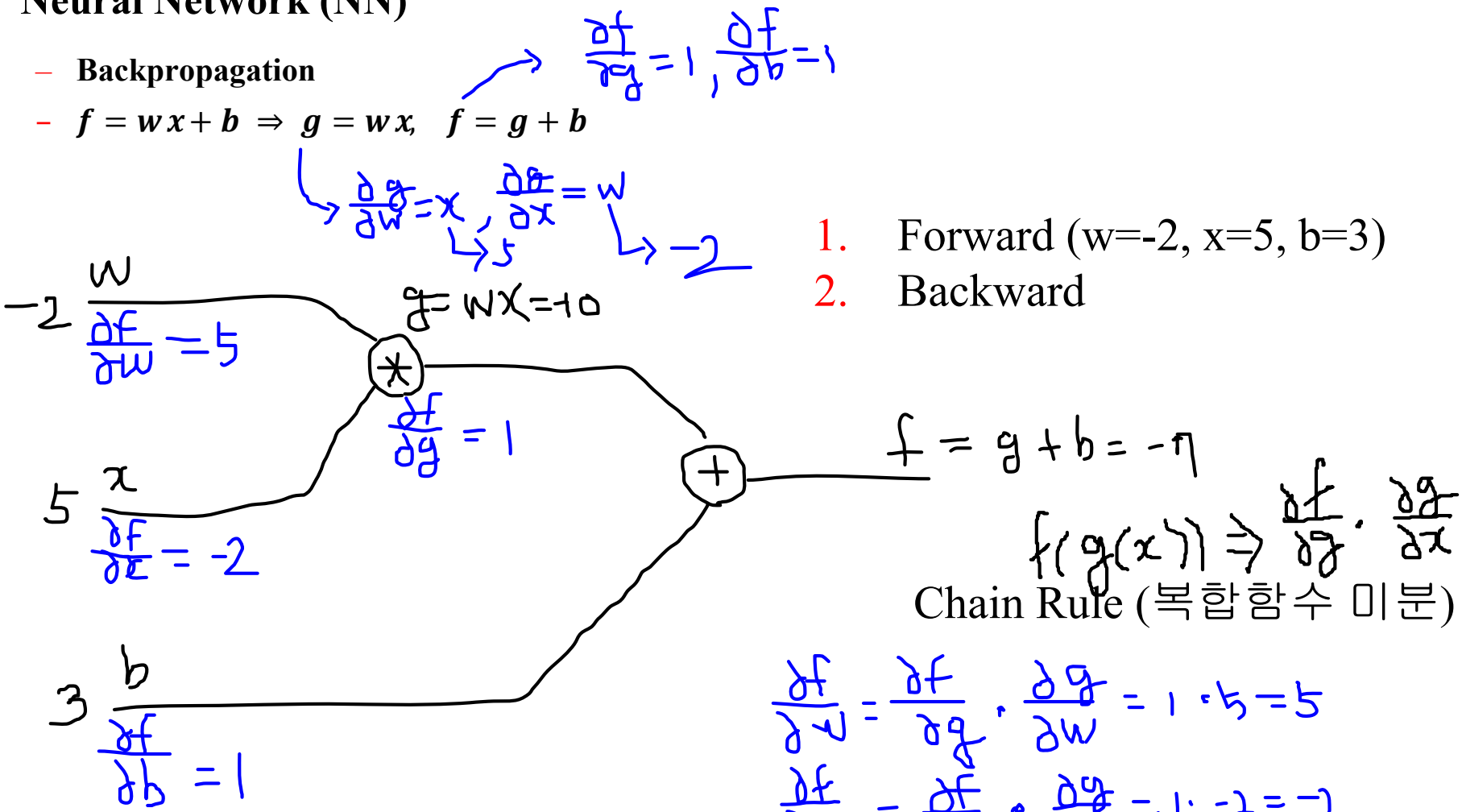
No one on earth had found a viable way to train MLPs good enough to learn such simple functions.

(1974, 1982 by Paul Werbos, 1986 by Hinton)

  - **Backpropagation**



- **Neural Network (NN)**
  - **Backpropagation**
  - $f = wx + b \Rightarrow g = wx, \quad f = g + b$



1. Forward (w=-2, x=5, b=3)
2. Backward

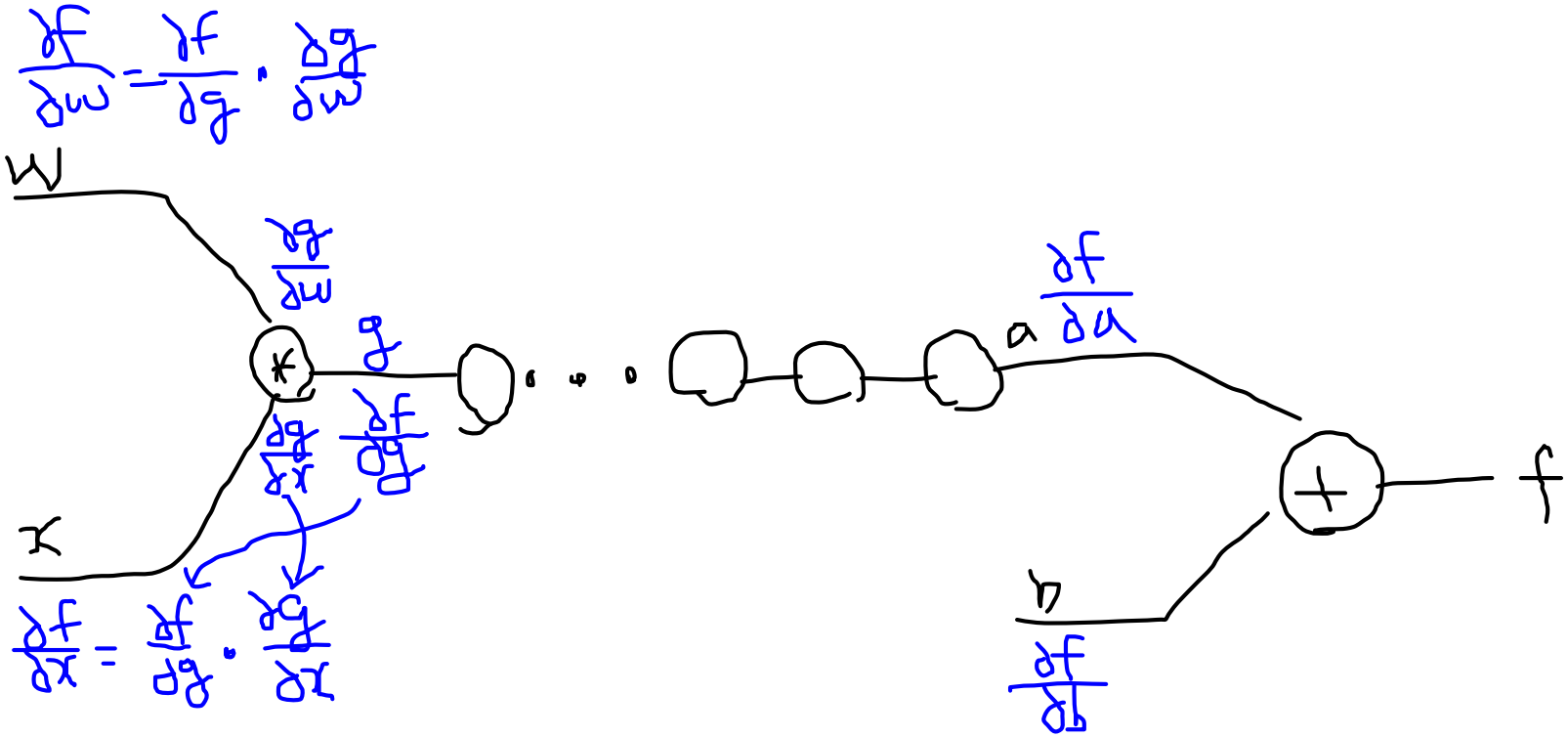
$f(g(x)) \Rightarrow \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$

Chain Rule (복합함수 미분)

$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial w} = 1 \cdot 5 = 5$

$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} = 1 \cdot -2 = -2$

- **Neural Network (NN)**
  - **Backpropagation (Chain Rule)**



◆ Neural Network (NN)

– XOR with Logistic Regression

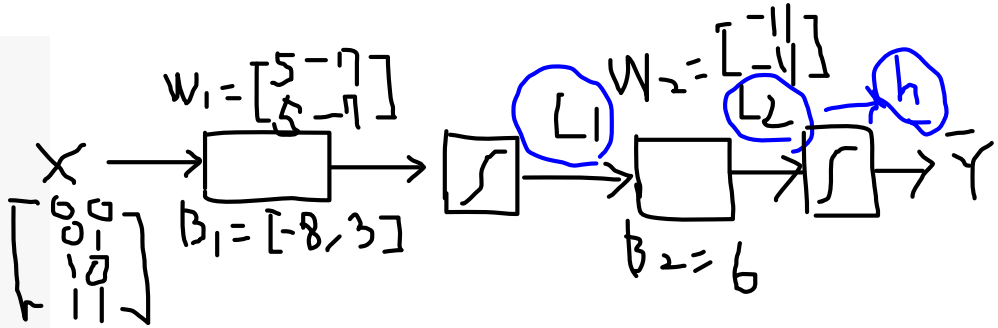
```
import tensorflow as tf
import numpy as np

learning_rate = 0.1

x_data = [[0, 0], [0, 1], [1, 0], [1, 1]]
y_data = [[0], [1], [1], [0]]
x_data = np.array(x_data, dtype=np.float32)
y_data = np.array(y_data, dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run(W))

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
    print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)
```

```
Hypothesis:  [[ 0.5]
 [ 0.5]
 [ 0.5]
 [ 0.5]]
Correct:  [[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
Accuracy:  0.5
```

◆ Neural Network (NN)

– XOR with NN

```
import tensorflow as tf
import numpy as np

learning_rate = 0.1

x_data = [[0, 0], [0, 1], [1, 0], [1, 1]]
y_data = [[0], [1], [1], [0]]
x_data = np.array(x_data, dtype=np.float32)
y_data = np.array(y_data, dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
Layer1 = tf.sigmoid(tf.matmul(X, W1)+b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')

hypothesis = tf.sigmoid(tf.matmul(Layer1, W2) + b2)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run([W1, W2]))

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
    print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)
```



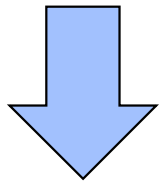

```
Hypothesis:  [[ 0.02242479]
 [ 0.97097147]
 [ 0.9712109 ]
 [ 0.01954121]]
Correct:  [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
Accuracy:  1.0
```

◆ Neural Network (NN)

– XOR with Wide & Deep NN

```
learning_rate = 0.1

x_data = [[0, 0], [0, 1], [1, 0], [1, 1]]
y_data = [[0], [1], [1], [0]]
x_data = np.array(x_data, dtype=np.float32)
y_data = np.array(y_data, dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
Layer1 = tf.sigmoid(tf.matmul(X, W1)+b1)

W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
Layer2 = tf.sigmoid(tf.matmul(Layer1, W2)+b2)

W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
Layer3 = tf.sigmoid(tf.matmul(Layer2, W3)+b3)

W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
b4 = tf.Variable(tf.random_normal([1]), name='bias4')

hypothesis = tf.sigmoid(tf.matmul(Layer3, W4) + b4)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run([W1, W2, W3, W4]))
```

```
Hypothesis:  [[ 0.02242479]
 [ 0.97097147]
 [ 0.9712109 ]
 [ 0.01954121]]
Correct:  [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
Accuracy:  1.0
```



```
Hypothesis:  [[  7.98452587e-04]
 [  9.98807430e-01]
 [  9.98901725e-01]
 [  2.08205916e-03]]
Correct:  [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
Accuracy:  1.0
```

Neural Network (NN)

Tensorboard for XOR NN

- Tensorboard: TF logging/debugging

- Visualize your TF graph
- Plot quantitative metrics
- Show additional data

```
9400 0.0151413 [array([[ 6.21692038,  6.05913448],
       [-6.33773184, -5.75189114]], dtype=float32), array([[ 9.93581772],
       [-9.43034935]], dtype=float32)]
9500 0.014909 [array([[ 6.22498751,  6.07049847],
       [-6.34637976, -5.76352596]], dtype=float32), array([[ 9.96414757],
       [-9.45942593]], dtype=float32)]
9600 0.0146836 [array([[ 6.23292685,  6.08166742],
       [-6.35489035, -5.77496052]], dtype=float32), array([[ 9.99207973],
       [-9.48807526]], dtype=float32)]
9700 0.0144647 [array([[ 6.24074268,  6.09264851],
       [-6.36326933, -5.78619957]], dtype=float32), array([[ 10.01962471],
       [ -9.51631165]], dtype=float32)]
9800 0.0142521 [array([[ 6.24843407,  6.10344648],
       [-6.37151814, -5.79724932]], dtype=float32), array([[ 10.04679298],
       [ -9.54414845]], dtype=float32)]
9900 0.0140456 [array([[ 6.25601053,  6.11406422],
       [-6.3796401 , -5.80811596]], dtype=float32), array([[ 10.07359505],
       [ -9.57159519]], dtype=float32)]
10000 0.0138448 [array([[ 6.26347113,  6.12451124],
       [-6.38764334, -5.81880617]], dtype=float32), array([[ 10.10004139],
       [ -9.59866238]], dtype=float32)]
```

Neural Network (NN)

Tensorboard for XOR NN

5 steps of using Tensorboard

1. From TF graph, decide which tensors you want to log

```
w2_hist = tf.summary.histogram("weights2", W2)
cost_summ = tf.summary.scalar("cost", cost)
```

2. Merge all summaries

```
summary = tf.summary.merge_all()
```

3. Create writer and add graph

```
# Create summary writer
writer = tf.summary.FileWriter('./logs')
writer.add_graph(sess.graph)
```

4. Run summary merge and add_summary

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
writer.add_summary(s, global_step=global_step)
```

5. Launch TensorBoard

```
tensorboard --logdir=./logs
```

```
with tf.name_scope("layer1"):
    W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
    b1 = tf.Variable(tf.random_normal([10]), name='bias1')
    Layer1 = tf.sigmoid(tf.matmul(X, W1)+b1)

    w1_hist = tf.summary.histogram("weights1", W1)
    b1_hist = tf.summary.histogram("biases1", b1)
    layer1_hist = tf.summary.histogram("layer1", Layer1)
```

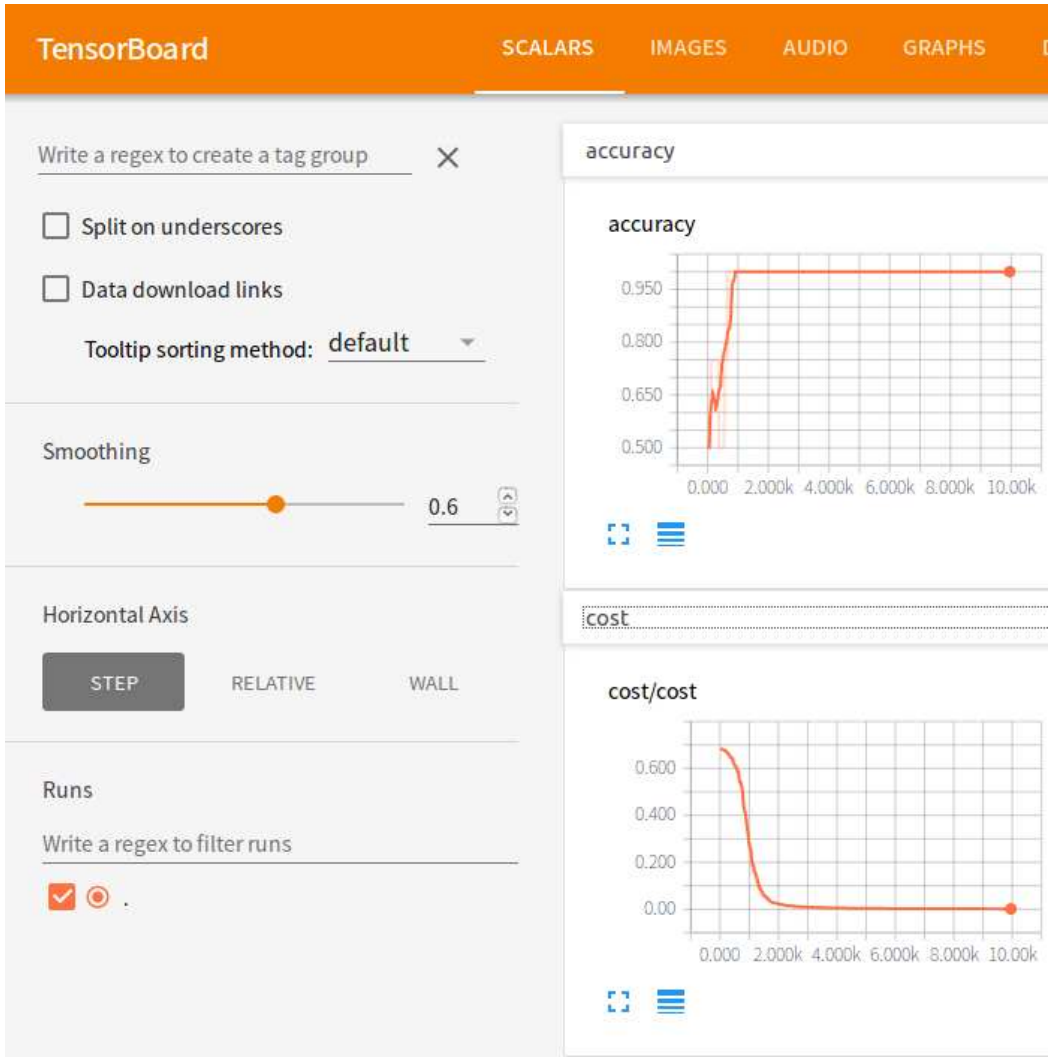
```
with tf.name_scope("cost"):
    cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
    cost_summary = tf.summary.scalar("cost", cost)
```

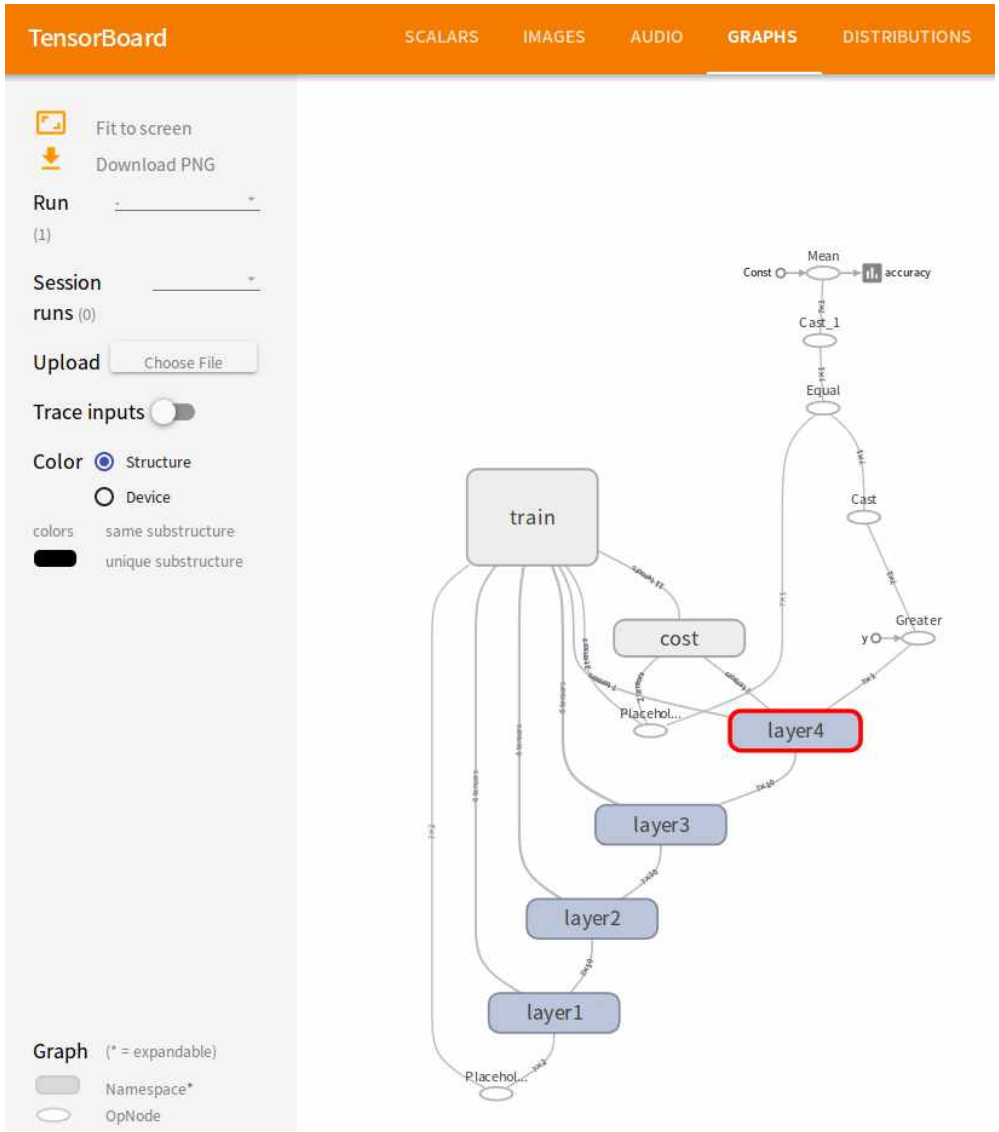
```
with tf.Session() as sess:

    merged_summary = tf.summary.merge_all()
    writer = tf.summary.FileWriter("./logs/xor_logs")
    writer.add_graph(sess.graph)   #show the graph
```

```
for step in range(10001):
    summary, _ = sess.run([merged_summary, train], feed_dict={X: x_data, Y: y_data})
    writer.add_summary(summary, global_step=step)
```

```
(ml-class) james@james-VirtualBox:~/ml-class$ tensorboard --logdir=./logs/xor_lo
gs
Starting TensorBoard b'39' on port 6006
(You can navigate to http://127.0.1.1:6006)
```

- **Neural Network (NN)**
  - **Tensorboard for XOR NN**





- **Neural Network (NN)**
  - **ReLu (Rectified Linear Unit)**

Neural Network: 넓고 깊을 수록 성능이 좋아 지는가?

Layer 1

```
Hypothesis:  [[ 0.02242479]
 [ 0.97097147]
 [ 0.9712109 ]
 [ 0.01954121]]
Correct:  [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
Accuracy:  1.0
```

Layer 4

```
Hypothesis:  [[  7.98452587e-04]
 [  9.98807430e-01]
 [  9.98901725e-01]
 [  2.08205916e-03]]
Correct:  [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
Accuracy:  1.0
```

Layer 9

Deep Network, Deep Learning

?

```
Hypothesis:  [[ 0.50006747]
 [ 0.49990967]
 [ 0.50005692]
 [ 0.49995124]]
Correct:  [[ 1.]
 [ 0.]
 [ 1.]
 [ 0.]]
Accuracy:  0.5
```

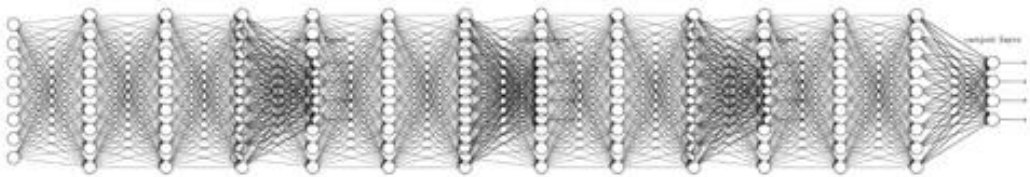
```
W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
Layer1 = tf.sigmoid(tf.matmul(X, W1)+b1)

W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
Layer2 = tf.sigmoid(tf.matmul(Layer1, W2)+b2)

W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
Layer3 = tf.sigmoid(tf.matmul(Layer2, W3)+b3)

W4 = tf.Variable(tf.random_normal([10, 5]), name='weight4')
b4 = tf.Variable(tf.random_normal([5]), name='bias4')
Layer4 = tf.sigmoid(tf.matmul(Layer3, W4)+b4)
```

```
W5 = tf.Variable(tf.random_normal([5, 5]), name='weight5')
b5 = tf.Variable(tf.random_normal([5]), name='bias5')
Layer5 = tf.sigmoid(tf.matmul(Layer4, W5)+b5)

W6 = tf.Variable(tf.random_normal([5, 5]), name='weight6')
b6 = tf.Variable(tf.random_normal([5]), name='bias6')
Layer6 = tf.sigmoid(tf.matmul(Layer5, W6)+b6)

W7 = tf.Variable(tf.random_normal([5, 5]), name='weight7')
b7 = tf.Variable(tf.random_normal([5]), name='bias7')
Layer7 = tf.sigmoid(tf.matmul(Layer6, W7)+b7)

W8 = tf.Variable(tf.random_normal([5, 5]), name='weight8')
b8 = tf.Variable(tf.random_normal([5]), name='bias8')
Layer8 = tf.sigmoid(tf.matmul(Layer7, W8)+b8)

W9 = tf.Variable(tf.random_normal([5, 1]), name='weight9')
b9 = tf.Variable(tf.random_normal([1]), name='bias9')
hypothesis = tf.sigmoid(tf.matmul(Layer8, W9) + b9)
```

- **Neural Network (NN)**
  - **ReLu (Rectified Linear Unit)**

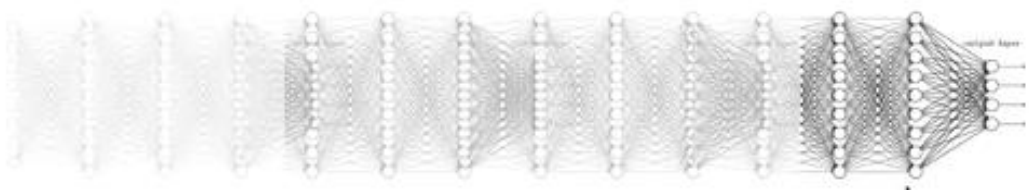
Backpropagation

Layer 1~3일 경우, 학습이 잘 되는데, 깊어지면 결과가 좋지 않음



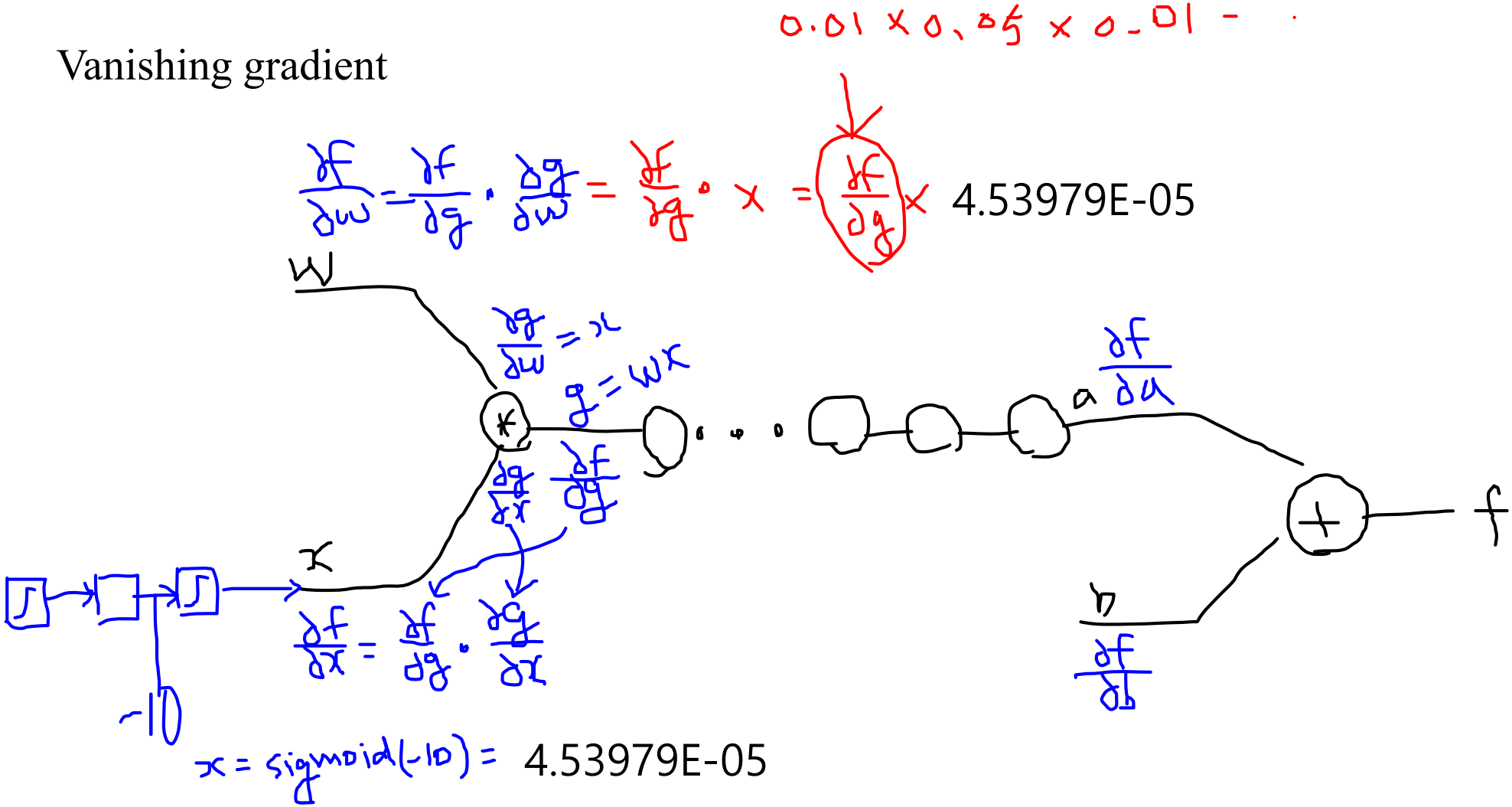
Why??

Vanishing gradient



- **Neural Network (NN)**
  - **ReLu (Rectified Linear Unit)**

Vanishing gradient

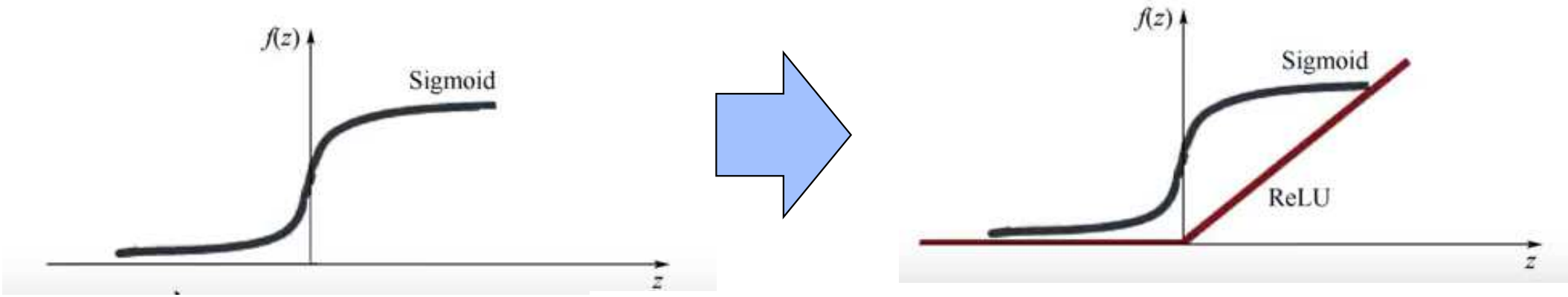


Neural Network winter season 2: 1986~2006

- **Deep Neural Network**
  - **ReLu (Rectified Linear Unit)**
    - » Geoffrey Hinton's summary of finding up to today

• Our labeled datasets were thousands of times too small.
• Our computers were millions of times too slow.
• We initialized the weights in a stupid way.
• We used the wrong type of non-linearity.

Sigmoid!




```
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
```

```
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
```

max(0, x)

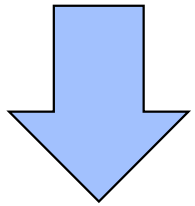
http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-4/

◆ Deep Neural Network

– XOR with Wide & Deep NN & ReLu

```
W1 = tf.Variable(tf.random_normal([2, 5]), name='weight1')
b1 = tf.Variable(tf.random_normal([5]), name='bias1')
#Layer1 = tf.sigmoid(tf.matmul(X, W1)+b1)
Layer1 = tf.nn.relu(tf.matmul(X, W1)+b1)

W2 = tf.Variable(tf.random_normal([5, 5]), name='weight2')
b2 = tf.Variable(tf.random_normal([5]), name='bias2')
#Layer2 = tf.sigmoid(tf.matmul(Layer1, W2)+b2)
Layer2 = tf.nn.relu(tf.matmul(Layer1, W2)+b2)

W3 = tf.Variable(tf.random_normal([5, 5]), name='weight3')
b3 = tf.Variable(tf.random_normal([5]), name='bias3')
#Layer3 = tf.sigmoid(tf.matmul(Layer2, W3)+b3)
Layer3 = tf.nn.relu(tf.matmul(Layer2, W3)+b3)

W4 = tf.Variable(tf.random_normal([5, 5]), name='weight4')
b4 = tf.Variable(tf.random_normal([5]), name='bias4')
#Layer4 = tf.sigmoid(tf.matmul(Layer3, W4)+b4)
Layer4 = tf.nn.relu(tf.matmul(Layer3, W4)+b4)

W5 = tf.Variable(tf.random_normal([5, 5]), name='weight5')
b5 = tf.Variable(tf.random_normal([5]), name='bias5')
#Layer5 = tf.sigmoid(tf.matmul(Layer4, W5)+b5)
Layer5 = tf.nn.relu(tf.matmul(Layer4, W5)+b5)

W6 = tf.Variable(tf.random_normal([5, 5]), name='weight6')
b6 = tf.Variable(tf.random_normal([5]), name='bias6')
#Layer6 = tf.sigmoid(tf.matmul(Layer5, W6)+b6)
Layer6 = tf.nn.relu(tf.matmul(Layer5, W6)+b6)

W7 = tf.Variable(tf.random_normal([5, 5]), name='weight7')
b7 = tf.Variable(tf.random_normal([5]), name='bias7')
#Layer7 = tf.sigmoid(tf.matmul(Layer6, W7)+b7)
Layer7 = tf.nn.relu(tf.matmul(Layer6, W7)+b7)

W8 = tf.Variable(tf.random_normal([5, 5]), name='weight8')
b8 = tf.Variable(tf.random_normal([5]), name='bias8')
#Layer8 = tf.sigmoid(tf.matmul(Layer7, W8)+b8)
Layer8 = tf.nn.relu(tf.matmul(Layer7, W8)+b8)

W9 = tf.Variable(tf.random_normal([5, 1]), name='weight9')
b9 = tf.Variable(tf.random_normal([1]), name='bias9')
hypothesis = tf.sigmoid(tf.matmul(Layer8, W9) + b9)
```

```
Hypothesis:  [[ 0.50006747]
 [ 0.49990967]
 [ 0.50005692]
 [ 0.49995124]]
Correct:  [[ 1.]
 [ 0.]
 [ 1.]
 [ 0.]]
Accuracy:  0.5
```

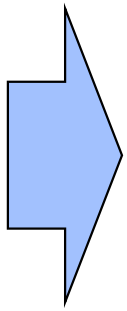


```
Hypothesis:  [[  9.13933691e-05]
 [  9.99976277e-01]
 [  9.99976277e-01]
 [  1.61922744e-05]]
Correct:  [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
Accuracy:  1.0
```

- **Deep Neural Network**
  - **MNIST Deep NN & ReLu**

```
#Layer 1
W1 = tf.Variable(tf.random_normal([784, 256]), name='weight1')
b1 = tf.Variable(tf.random_normal([256]), name='bias1')
Layer1 = tf.nn.relu(tf.matmul(X, W1) + b1)

#Layer 2
W2 = tf.Variable(tf.random_normal([256, 128]), name='weight2')
b2 = tf.Variable(tf.random_normal([128]), name='bias2')
Layer2 = tf.nn.relu(tf.matmul(Layer1, W2) + b2)

#Layer 3
W3 = tf.Variable(tf.random_normal([128, 64]), name='weight3')
b3 = tf.Variable(tf.random_normal([64]), name='bias3')
Layer3 = tf.nn.relu(tf.matmul(Layer2, W3) + b3)

#Layer 4
W4 = tf.Variable(tf.random_normal([64, classes]), name='weight4')
b4 = tf.Variable(tf.random_normal([classes]), name='bias4')

hypothesis = tf.matmul(Layer3, W4) + b4

# adjust learning_rate
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
opt = tf.train.AdamOptimizer(learning_rate=0.01).minimize(cost)
```

```
Epoch: 0001 cost = 2.898222886
Epoch: 0002 cost = 1.108921999
Epoch: 0003 cost = 0.874881318
Epoch: 0004 cost = 0.763935795
Epoch: 0005 cost = 0.694653324
Epoch: 0006 cost = 0.645577858
Epoch: 0007 cost = 0.608289677
Epoch: 0008 cost = 0.579229297
Epoch: 0009 cost = 0.555318202
Epoch: 0010 cost = 0.535004765
Epoch: 0011 cost = 0.517436929
Epoch: 0012 cost = 0.502804135
Epoch: 0013 cost = 0.489608879
Epoch: 0014 cost = 0.477709723
Epoch: 0015 cost = 0.467347786
Epoch: 0016 cost = 0.457904843
Training Done!!!!
Accuracy:  0.8895
Label:  [6]
Prediction:  [6]
```



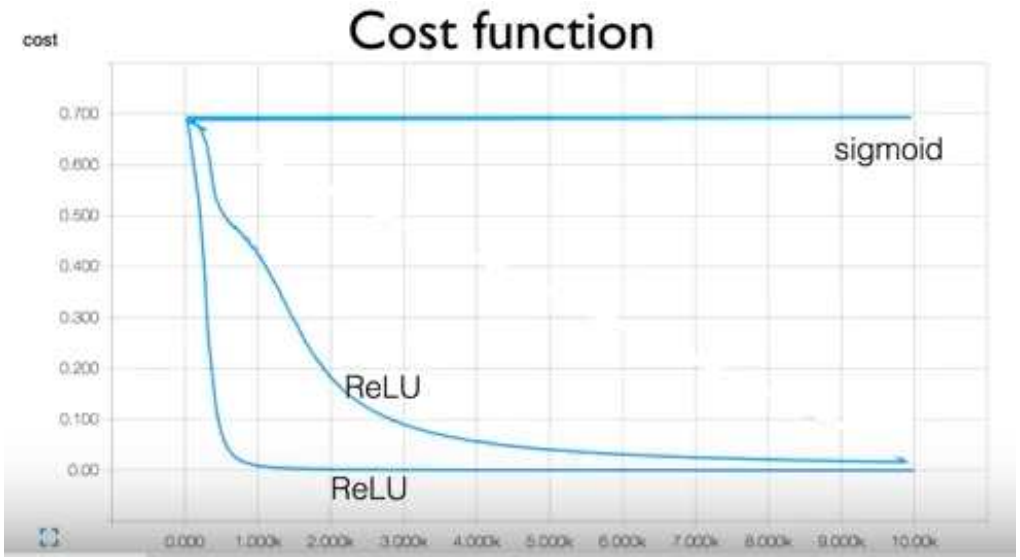
```
Epoch: 0001 cost = 121.618425740
Epoch: 0002 cost = 16.247775788
Epoch: 0003 cost = 7.806664266
Epoch: 0004 cost = 4.454607668
Epoch: 0005 cost = 3.198214591
Epoch: 0006 cost = 2.333993622
Epoch: 0007 cost = 1.801443546
Epoch: 0008 cost = 1.454116968
Epoch: 0009 cost = 1.079358814
Epoch: 0010 cost = 1.011013666
Epoch: 0011 cost = 0.772059956
Epoch: 0012 cost = 0.509782461
Epoch: 0013 cost = 0.422369142
Epoch: 0014 cost = 0.396618821
Epoch: 0015 cost = 0.206073985
Epoch: 0016 cost = 0.169676772
Training Done!!!!
Accuracy:  0.951
Label:  [7]
Prediction:  [7]
```

◆ Deep Neural Network

– Initialization

» Geoffrey Hinton's summary of finding up to today

- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- **We initialized the weights in a stupid way.**
- We used the wrong type of non-linearity.



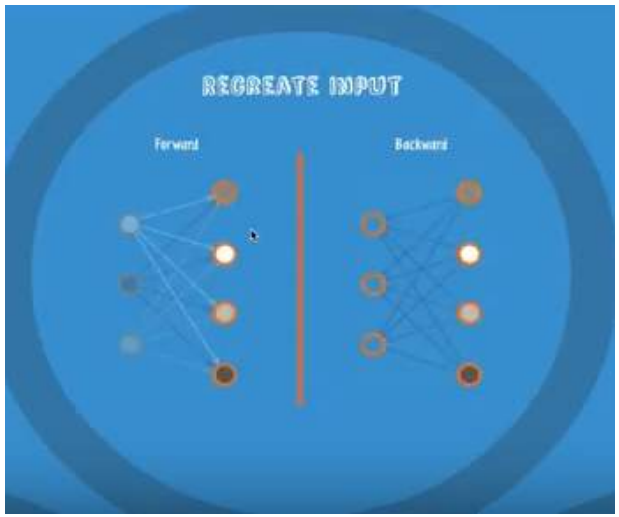
◆ Deep Neural Network

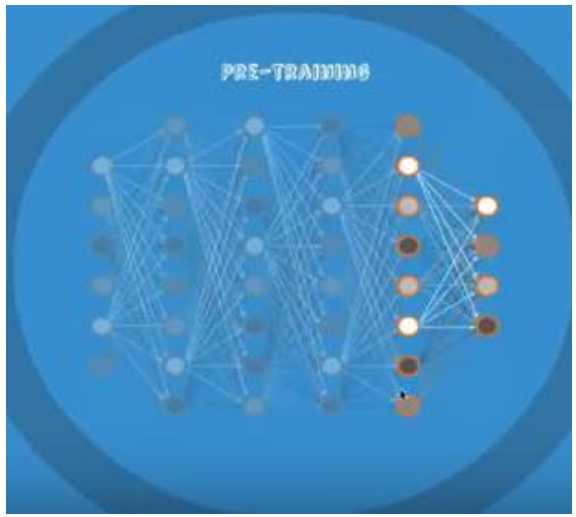
– Initialization

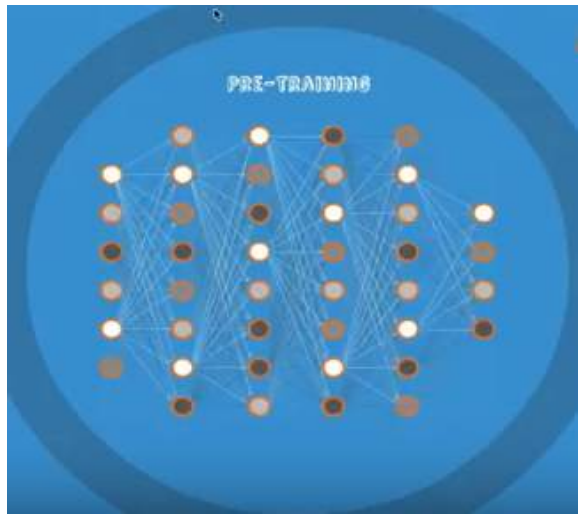
» Need to set the initial weight values wisely

- Not all 0's
- Challenging issue
- Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets"
  - Restricted Boatman Machine (RBM)

restricted boltzmann machine







◆ Deep Neural Network

– Initialization

» **No need to use complicated RBM for weight initializations**

- Simple methods are OK
  - **Xavier initialization**: X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in International conference on artificial intelligence and statistics, 2010
  - **He's initialization**: K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015

- Makes sure the weights are 'just right', not too small, not too big
- Using number of input (fan_in) and output (fan_out)

```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

ResNet (ImageNet: Error 3%이하)

Init method	maxout	ReLU	VLReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	92.97	89.28	n/c
OrthoNorm	93.78	91.74	92.40	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	**93.09**	–	n/c
Xavier	91.75	90.63	92.27	**89.82**	n/c
MSRA	n/c†	90.91	92.43	89.54	n/c

Deep Neural Network

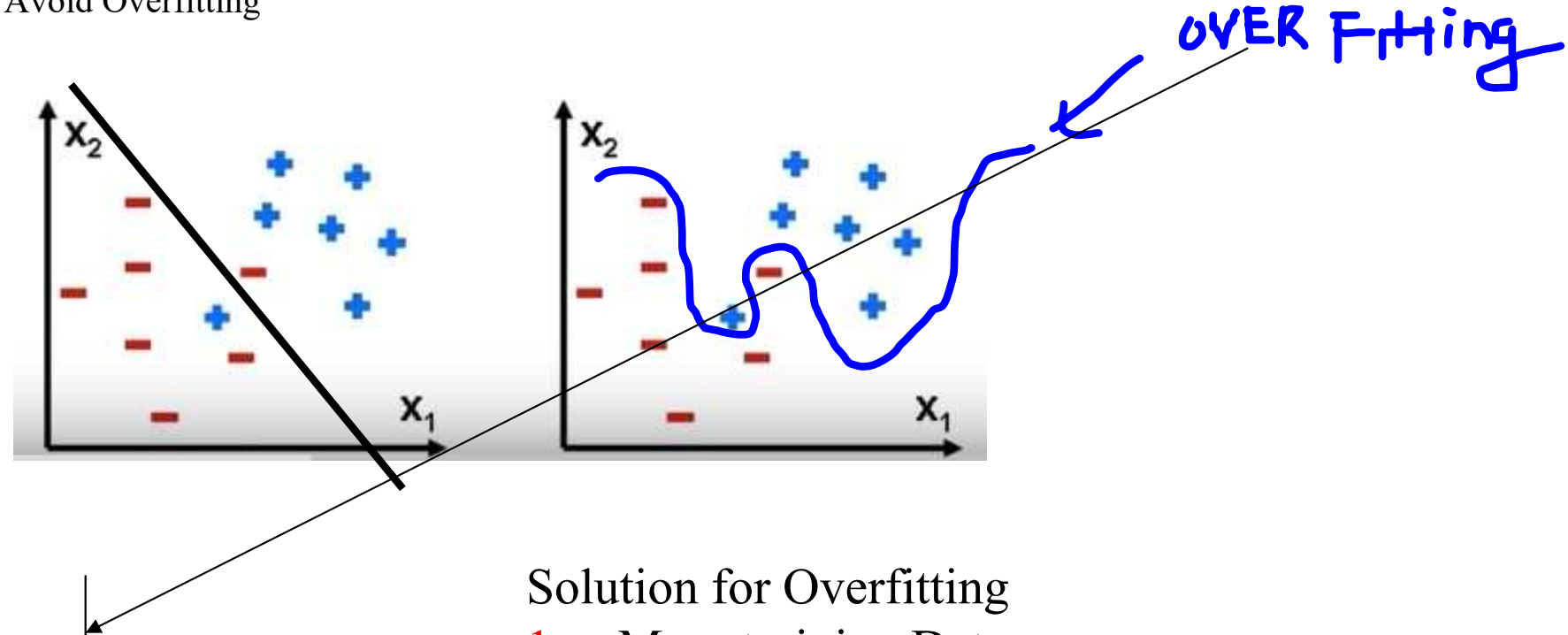
MNIST Deep NN & ReLu & Xavier Initialization

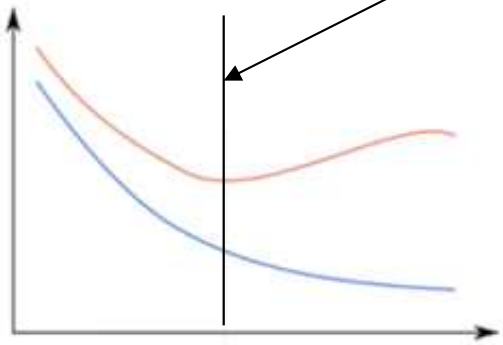
```
#Layer 1
W1 = tf.get_variable("W1", shape=[784,256], initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256]))
Layer1 = tf.nn.relu(tf.matmul(X, W1) + b1)

#Layer 2
W2 = tf.get_variable("W2", shape=[256,128], initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([128]))
Layer2 = tf.nn.relu(tf.matmul(Layer1, W2) + b2)

#Layer 3
W3 = tf.get_variable("W3", shape=[128,64], initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([64]))
Layer3 = tf.nn.relu(tf.matmul(Layer2, W3) + b3)

#Layer 4
W4 = tf.get_variable("W4", shape=[64,classes], initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([classes]))

hypothesis = tf.matmul(Layer3, W4) + b4

# adjust learning_rate
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
opt = tf.train.AdamOptimizer(learning_rate=0.01).minimize(cost)
```

```
Epoch: 0001 cost = 121.618425740
Epoch: 0002 cost = 16.247775788
Epoch: 0003 cost = 7.806664266
Epoch: 0004 cost = 4.454607668
Epoch: 0005 cost = 3.198214591
Epoch: 0006 cost = 2.333993622
Epoch: 0007 cost = 1.801443546
Epoch: 0008 cost = 1.454116968
Epoch: 0009 cost = 1.079358814
Epoch: 0010 cost = 1.011013666
Epoch: 0011 cost = 0.772059956
Epoch: 0012 cost = 0.509782461
Epoch: 0013 cost = 0.422369142
Epoch: 0014 cost = 0.396618821
Epoch: 0015 cost = 0.206073985
Epoch: 0016 cost = 0.169676772
Training Done!!!!
Accuracy:  0.951
Label:  [7]
Prediction:  [7]
```

```
Epoch: 0001 cost = 0.314719560
Epoch: 0002 cost = 0.136928339
Epoch: 0003 cost = 0.114664485
Epoch: 0004 cost = 0.105699805
Epoch: 0005 cost = 0.093989542
Epoch: 0006 cost = 0.091876498
Epoch: 0007 cost = 0.079226325
Epoch: 0008 cost = 0.076840967
Epoch: 0009 cost = 0.074611597
Epoch: 0010 cost = 0.064871844
Epoch: 0011 cost = 0.064510580
Epoch: 0012 cost = 0.072228441
Epoch: 0013 cost = 0.067988544
Epoch: 0014 cost = 0.050129582
Epoch: 0015 cost = 0.062997886
Epoch: 0016 cost = 0.054448708
Training Done!!!!
Accuracy:  0.9769
Label:  [2]
Prediction:  [2]
```

- **Deep Neural Network**
  - **Drop out**
    - » Avoid Overfitting





- Very high accuracy on the training dataset (eg: 0.99)
- Poor accuracy on the test data set (0.85)

Solution for Overfitting

1. More training Data
2. **Regularization**
   - Not have too big numbers in the weight

$$\text{cost} + \lambda \sum W^2$$

```
l2reg = 0.001 * tf.reduce_sum(tf.square(W))
```

Deep Neural Network

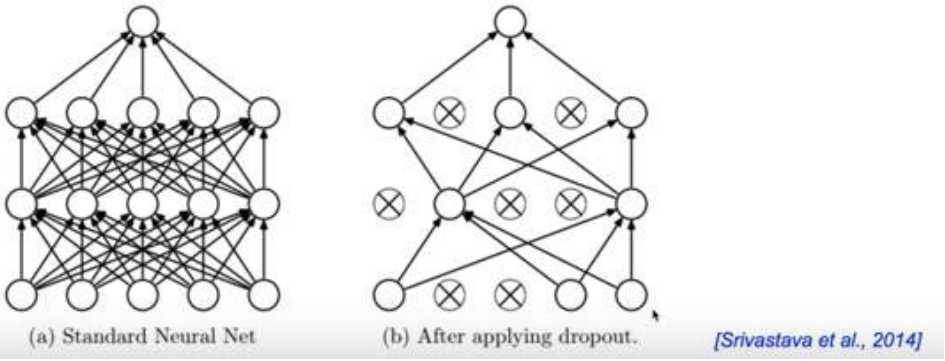
Dropout

» Avoid Overfitting

Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]

Regularization: **Dropout**

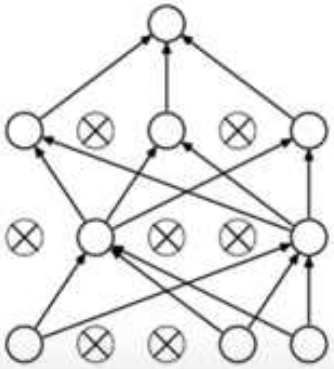
"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net (b) After applying dropout. [Srivastava et al., 2014]

```
dropout_rate = tf.placeholder("float")
_L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
L1 = tf.nn.dropout(_L1, dropout_rate)
```

```
TRAIN:
  sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys,
  dropout_rate: 0.7})

EVALUATION:
  print "Accuracy:", accuracy.eval({X: mnist.test.images, Y:
  mnist.test.labels, dropout_rate: 1})
```



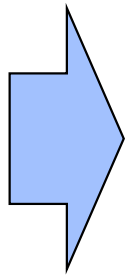
Forces the network to have a redundant representation.

Deep Neural Network

MNIST Deep NN & ReLu & Xavier Initialization & Dropout

```
W1 = tf.get_variable("W1", shape=[784, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

W2 = tf.get_variable("W2", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)

W3 = tf.get_variable("W3", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([512]))
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
L3 = tf.nn.dropout(L3, keep_prob=keep_prob)

W4 = tf.get_variable("W4", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([512]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

W5 = tf.get_variable("W5", shape=[512, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5
```

```
Epoch: 0001 cost = 0.314719560
Epoch: 0002 cost = 0.136928339
Epoch: 0003 cost = 0.114664485
Epoch: 0004 cost = 0.105699805
Epoch: 0005 cost = 0.093989542
Epoch: 0006 cost = 0.091876498
Epoch: 0007 cost = 0.079226325
Epoch: 0008 cost = 0.076840967
Epoch: 0009 cost = 0.074611597
Epoch: 0010 cost = 0.064871844
Epoch: 0011 cost = 0.064510580
Epoch: 0012 cost = 0.072228441
Epoch: 0013 cost = 0.067988544
Epoch: 0014 cost = 0.050129582
Epoch: 0015 cost = 0.062997886
Epoch: 0016 cost = 0.054448708
Training Done!!!!
Accuracy:  0.9769
Label:  [2]
Prediction:  [2]
```



```
Epoch: 0001 cost = 0.477981920
Epoch: 0002 cost = 0.174209262
Epoch: 0003 cost = 0.133356226
Epoch: 0004 cost = 0.107630954
Epoch: 0005 cost = 0.097190227
Epoch: 0006 cost = 0.082390315
Epoch: 0007 cost = 0.075062841
Epoch: 0008 cost = 0.069322145
Epoch: 0009 cost = 0.064841361
Epoch: 0010 cost = 0.058239024
Epoch: 0011 cost = 0.056864930
Epoch: 0012 cost = 0.054271612
Epoch: 0013 cost = 0.052183560
Epoch: 0014 cost = 0.046791719
Epoch: 0015 cost = 0.044676678
Learning Finished!
Accuracy: 0.9818
```

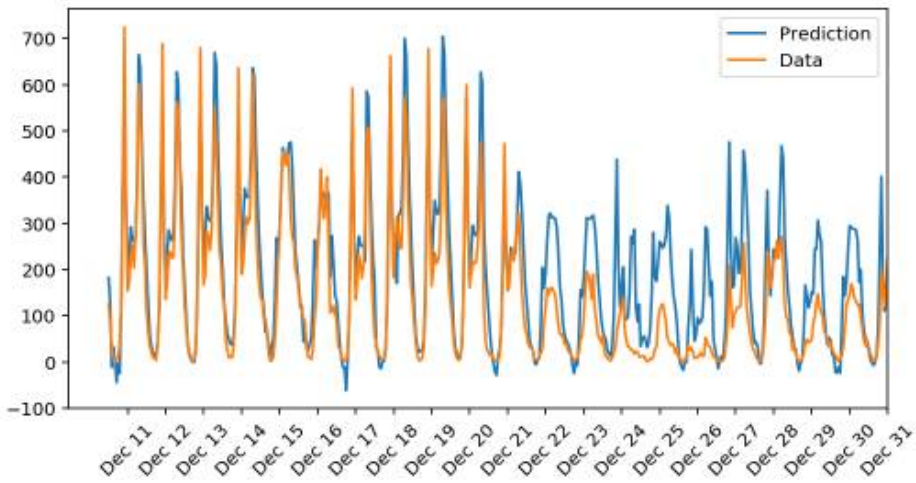
```
for i in range(total_batch):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)
    feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
    c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
    avg_cost += c / total_batch
```

```
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))
```

Deep Neural Network

- Network를 보다 깊게 쌓아 MNIST Test
- 속도 표지판 인식
- 차량 번호판 인식

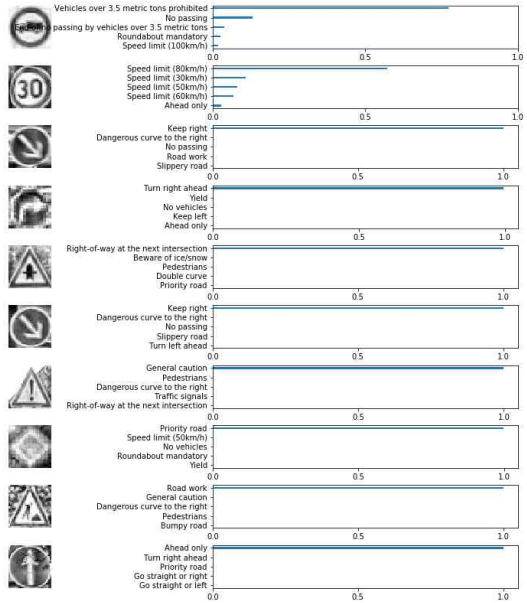
기존 Data를 학습하여 자전거 대여 대수 예측하기 (using CNN)



NEXT

Convolution Neural Network
Recurrent Neural Network

Traffic Sign Recognition using CNN

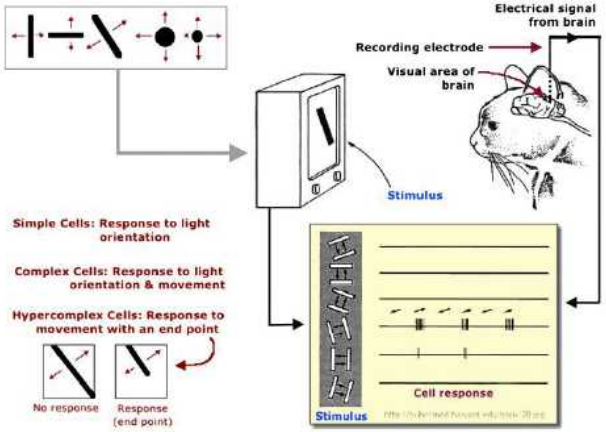


Steering value prediction using CNN

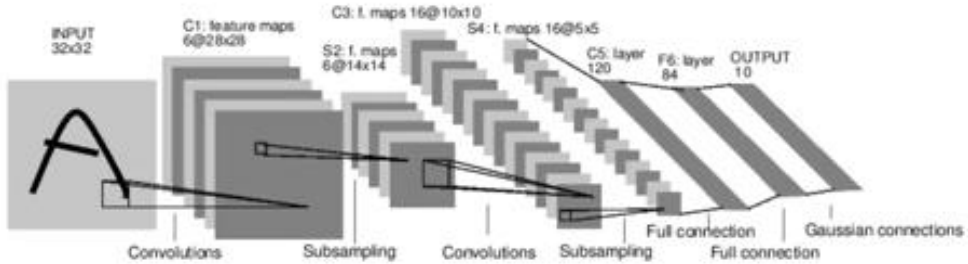
Language Translation using RNN

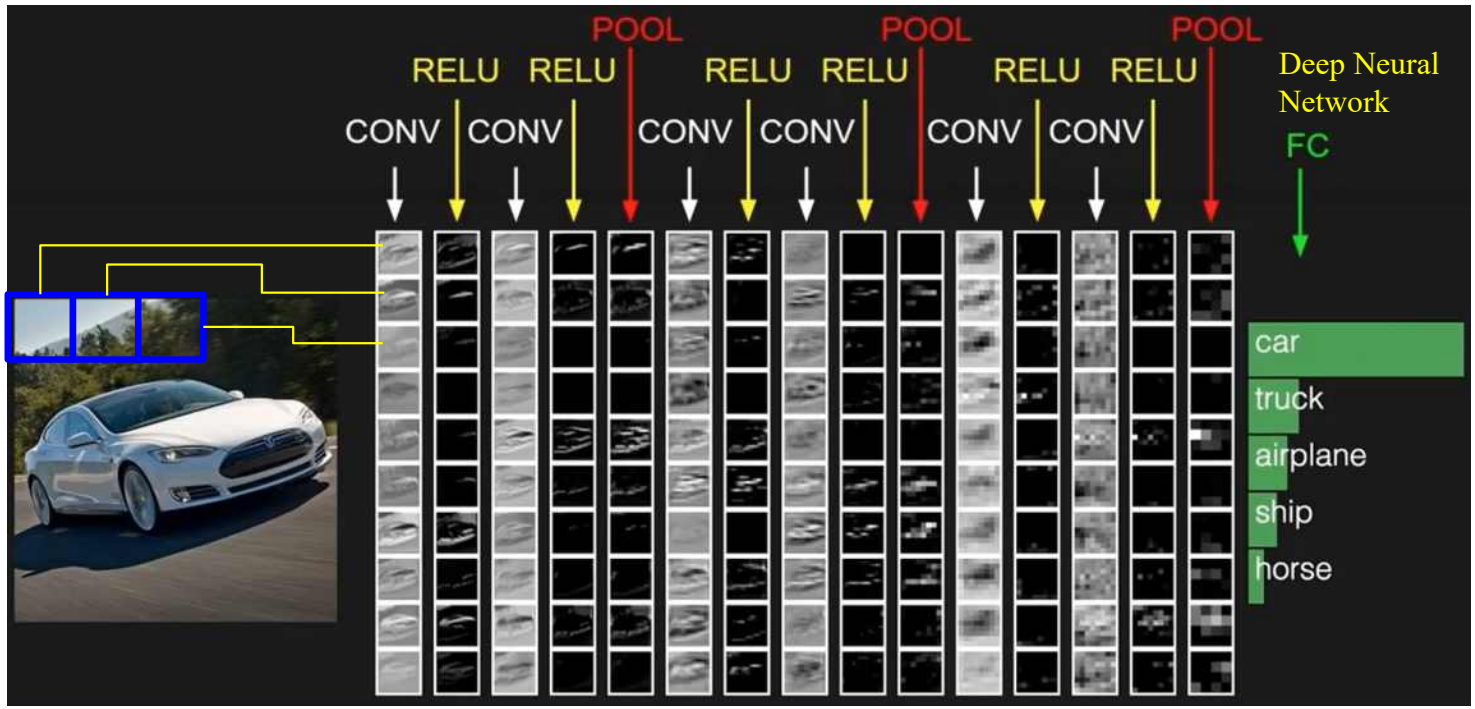
```
Input
  Word Ids:      [208, 68, 203, 32, 9, 95, 129]
  English Words: ['he', 'saw', 'a', 'old', 'yellow', 'truck', '.']

Prediction
  Word Ids:      [288, 175, 27, 141, 209, 293, 10, 325, 1]
  French Words: il a vu un vieux camion jaune . <EOS>
```

- **Convolution Neural Network**



CNN: Convolutional Neural Network



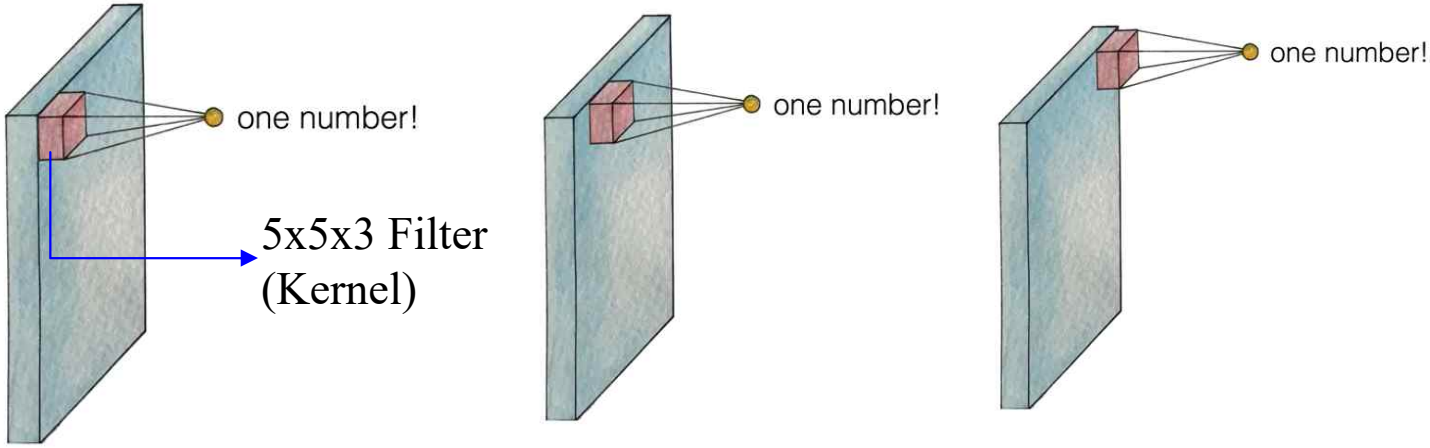


SoftMax

◆ Convolution Neural Network

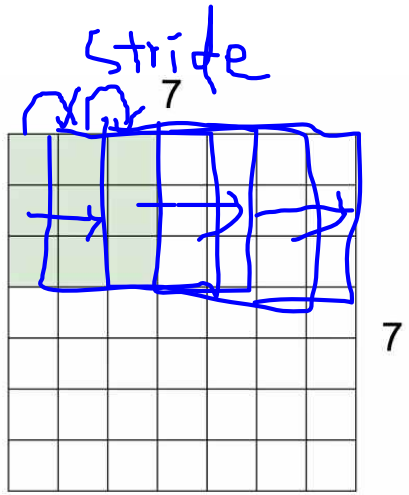
– Conv Layer

동일한 필터(w)를 가지고 이동



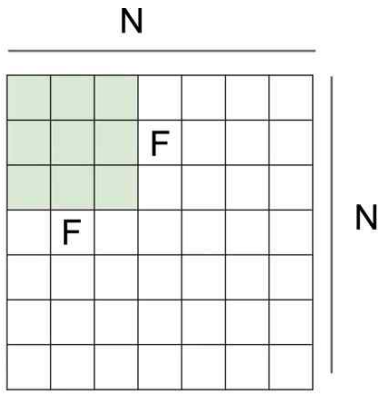
32x32x3 image

How many numbers can we get?



7x7 input (spatially)
assume 3x3 filter

⇒ 5 x 5 (out)

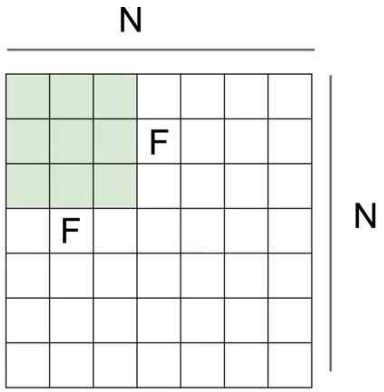


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

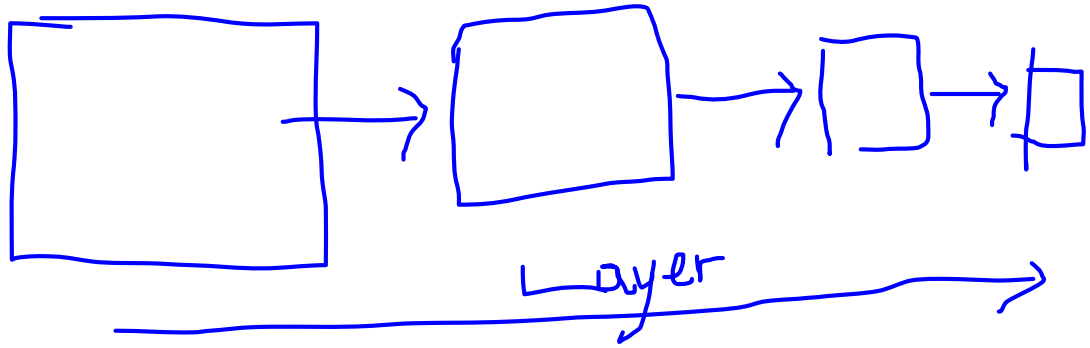
Convolution Neural Network

Conv Layer

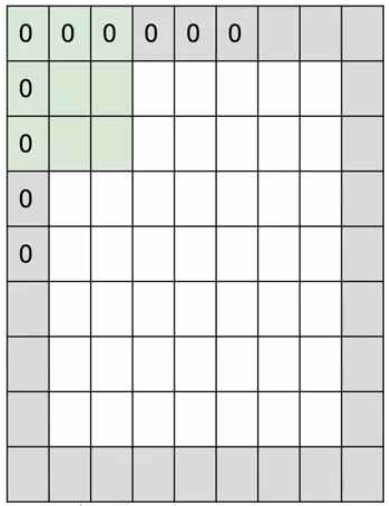


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33



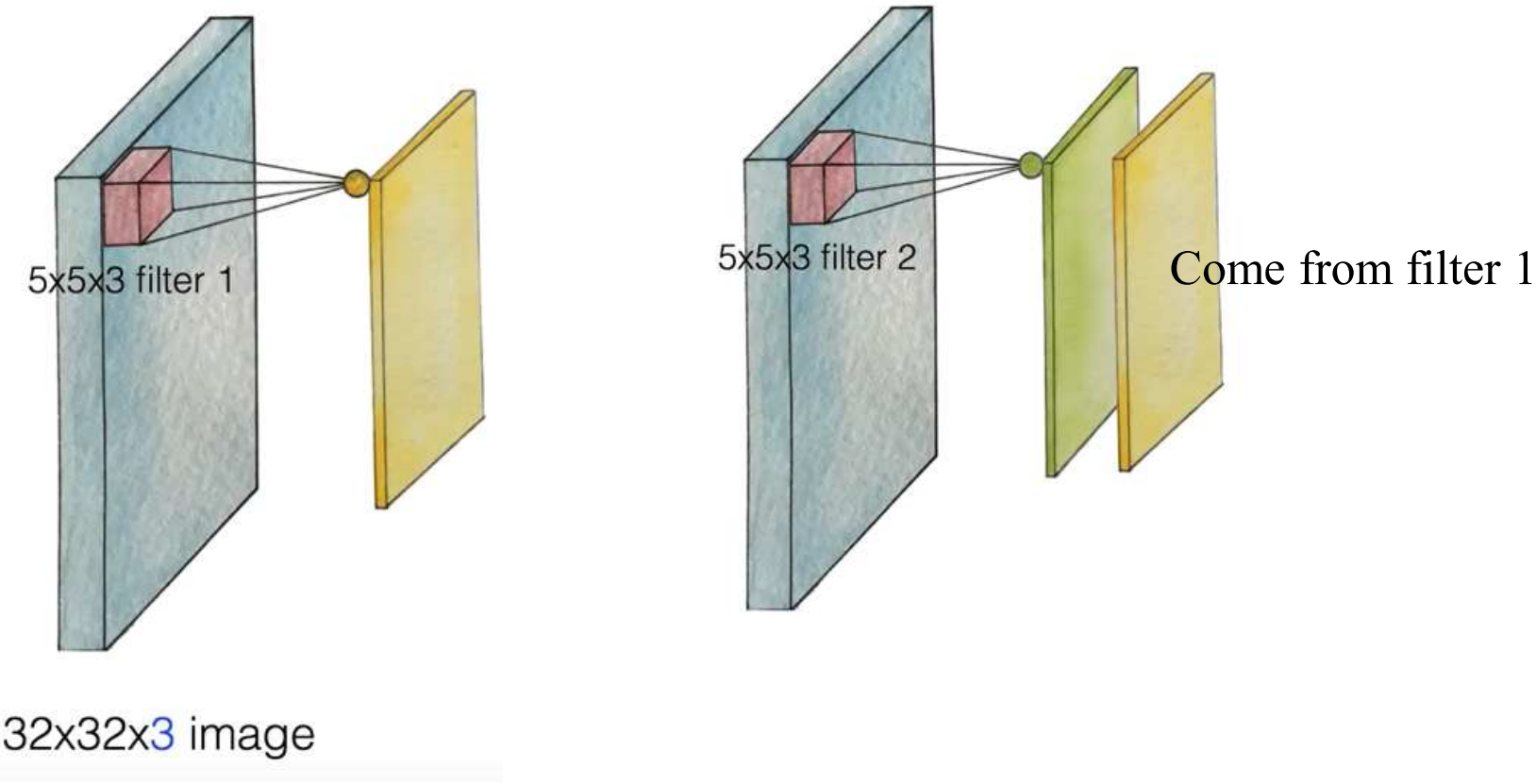
In practice: Common to zero pad the border



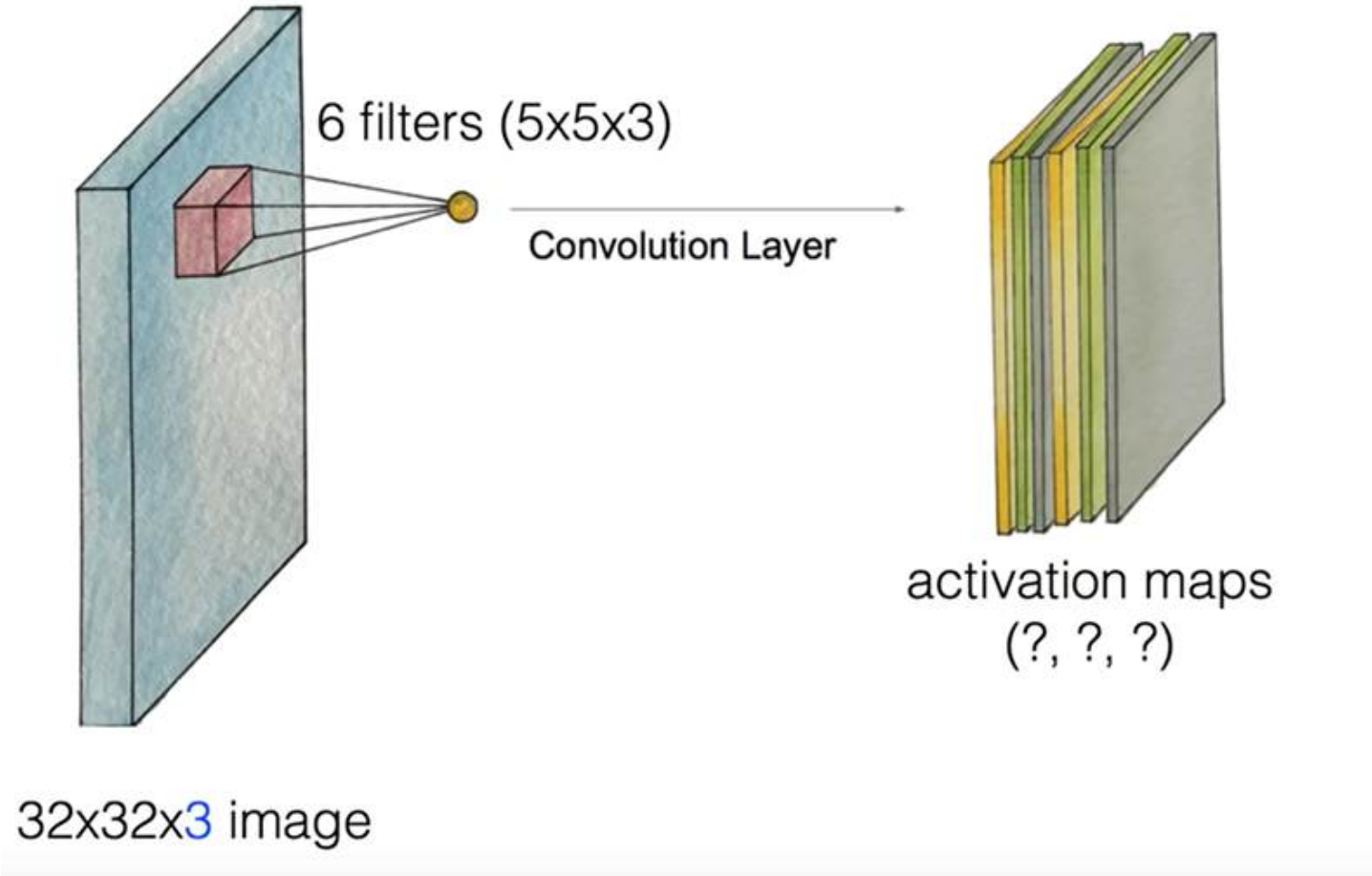
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

- **Convolution Neural Network**
  - **Conv Layer**

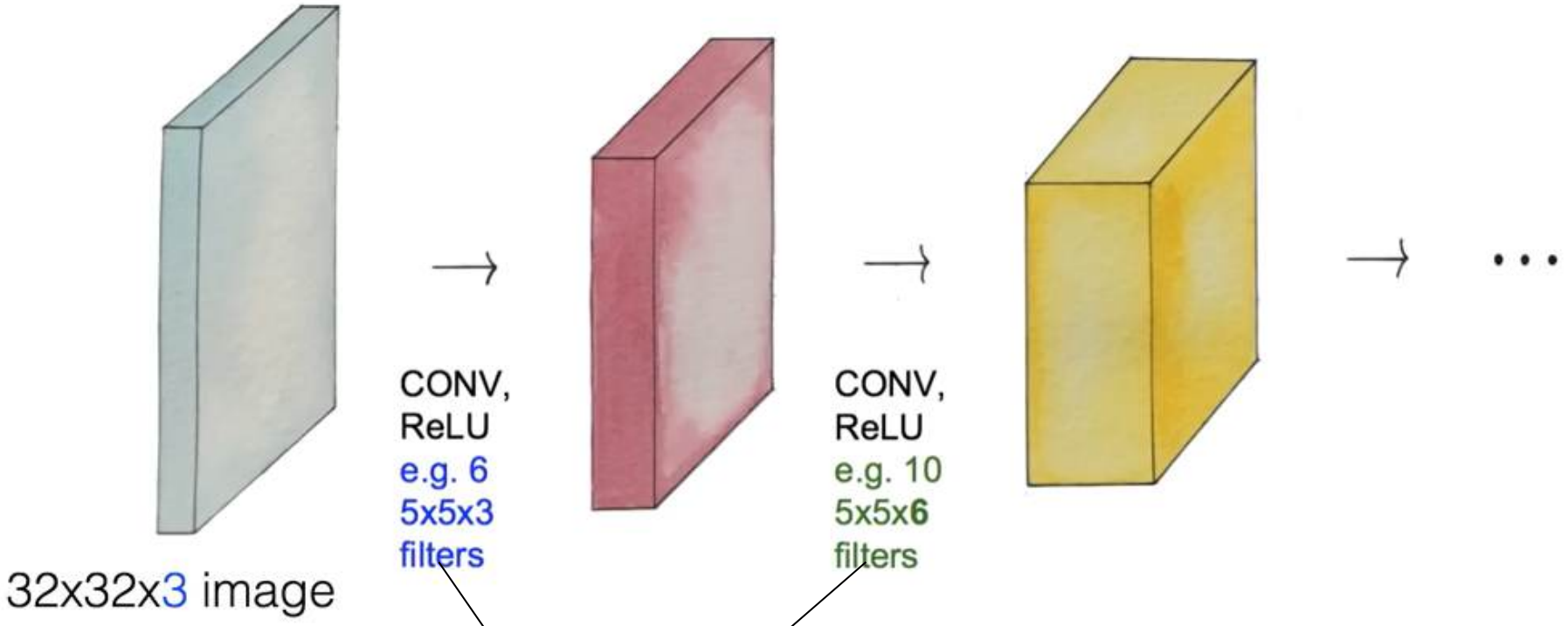


- **Convolution Neural Network**
  - **Conv Layer**

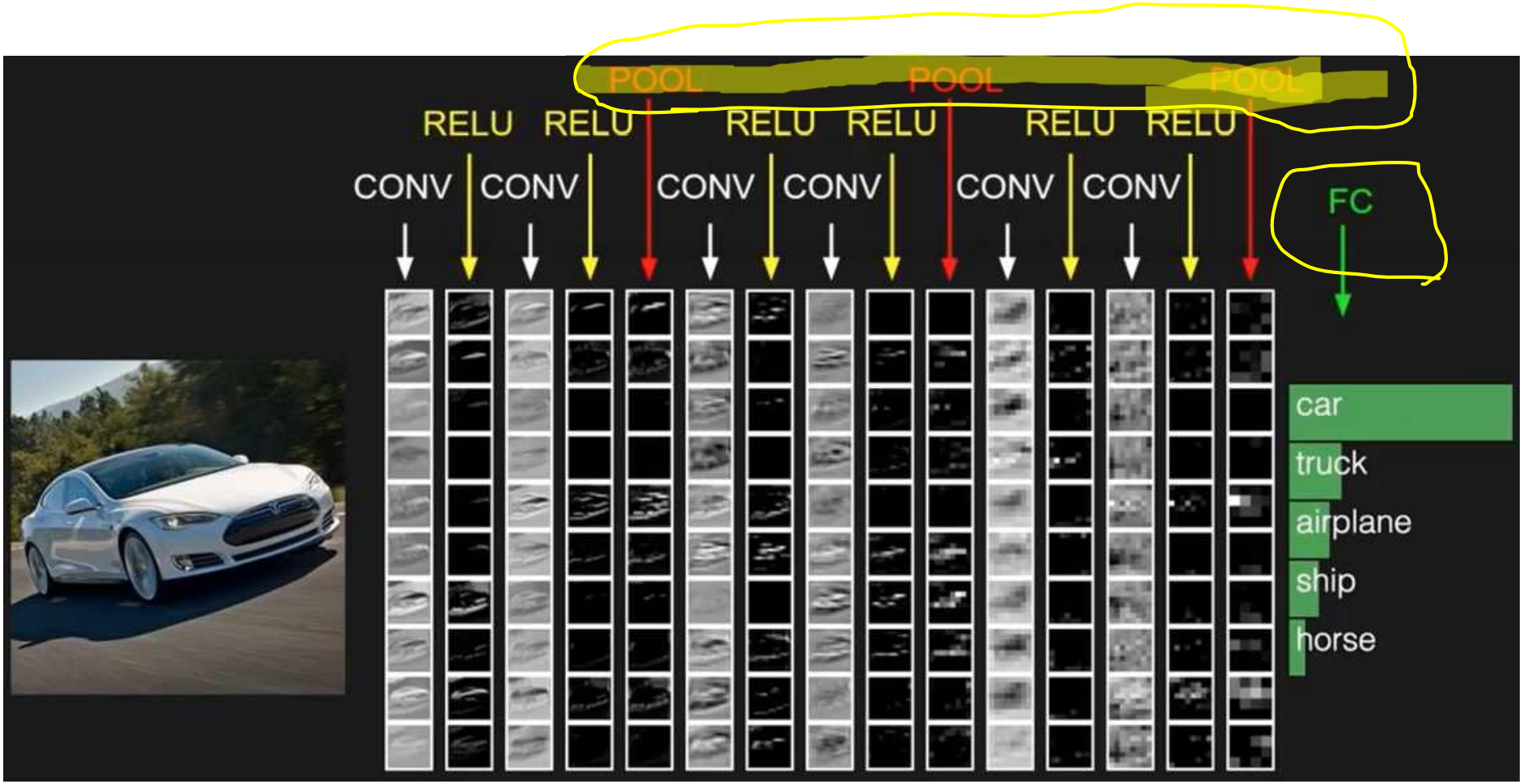


◆ Convolution Neural Network

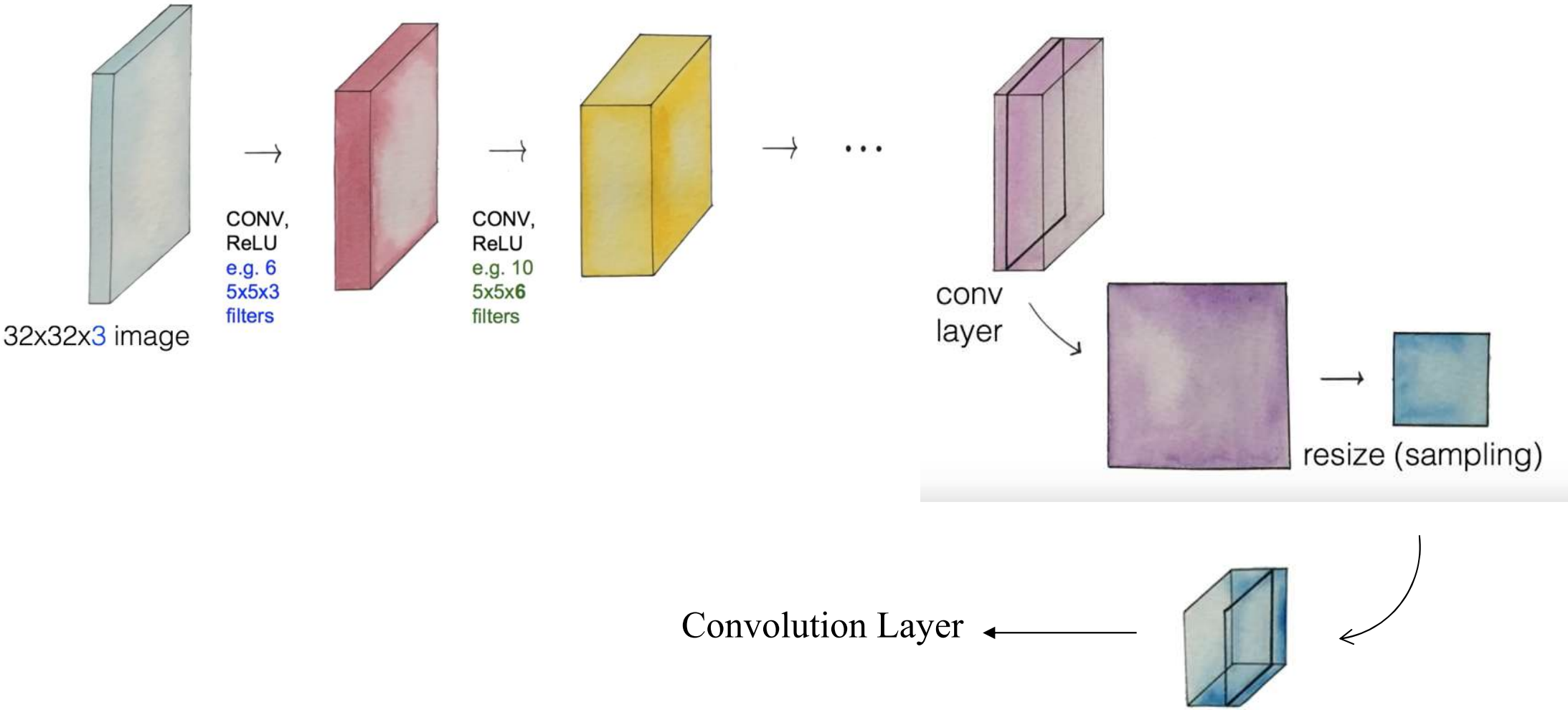
– Conv Layer



- **Convolution Neural Network**
  - **Max pooling and others**



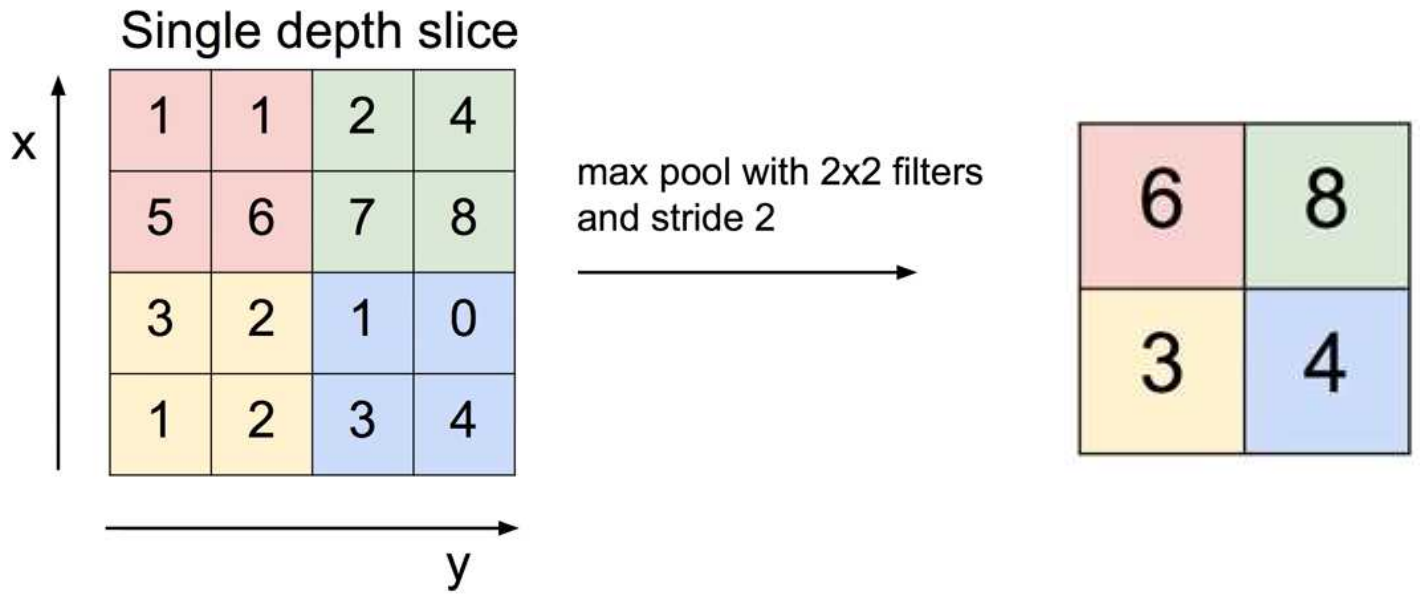
- **Convolution Neural Network**
  - **Pooling Layer (Sampling과 유사)**



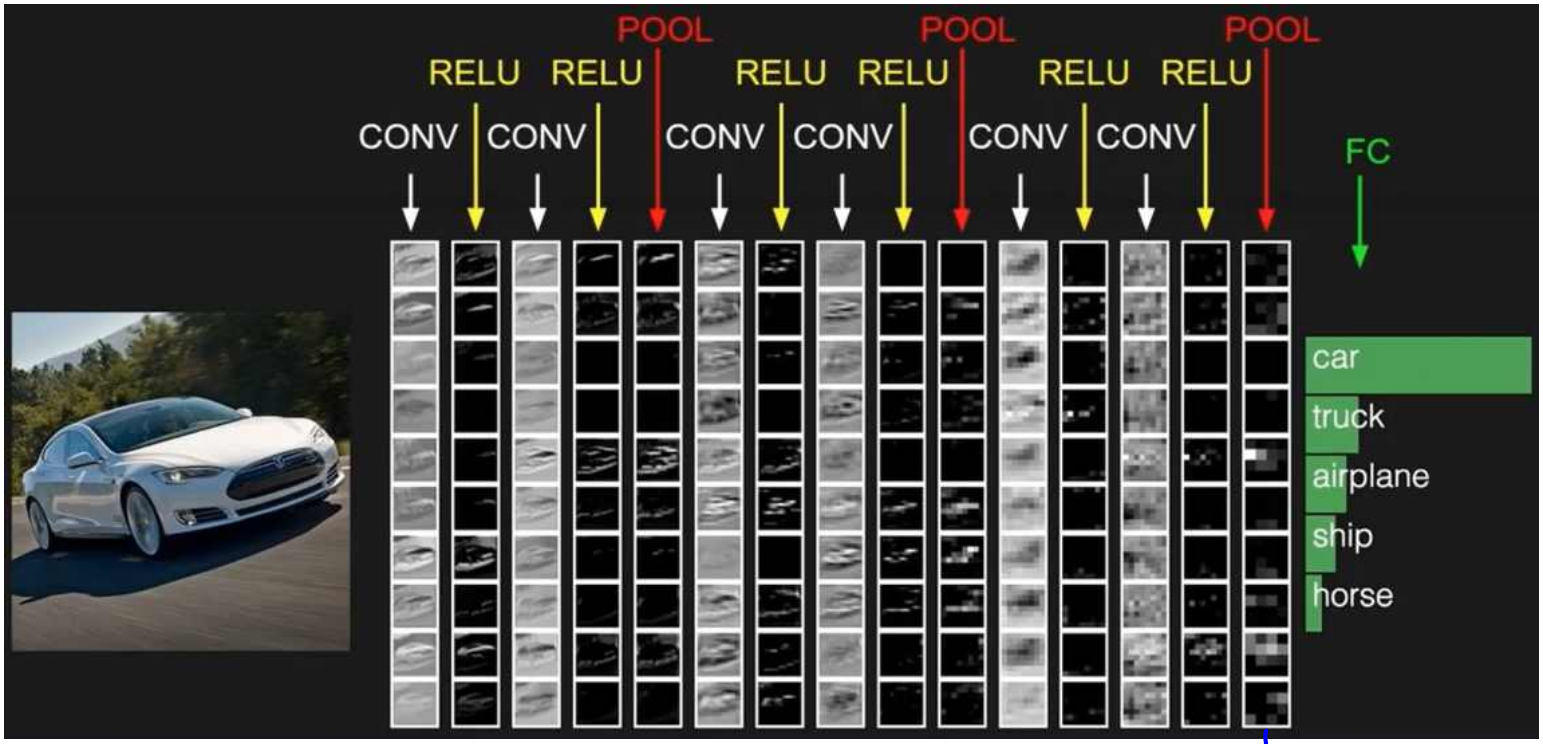
Convolution Neural Network

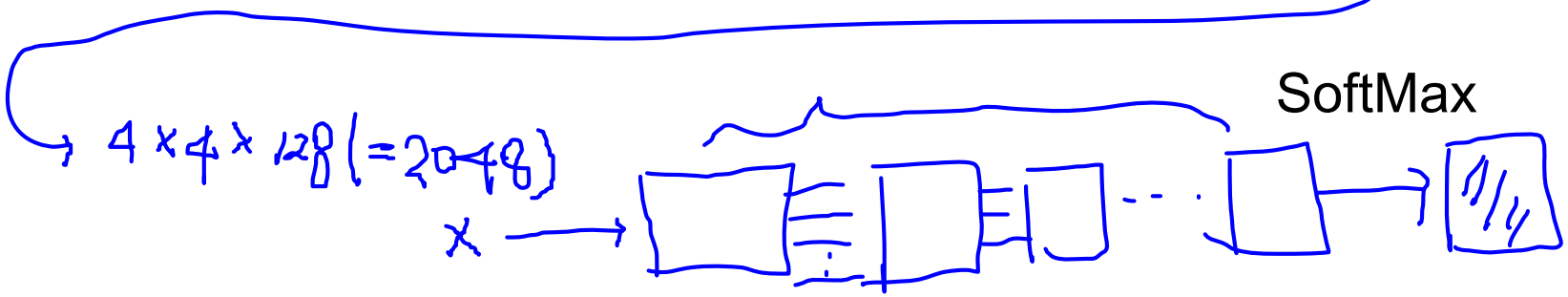
Pooling Layer

- Max Pooling



- **Convolution Neural Network**
  - **FC (Fully Connected) Layer**

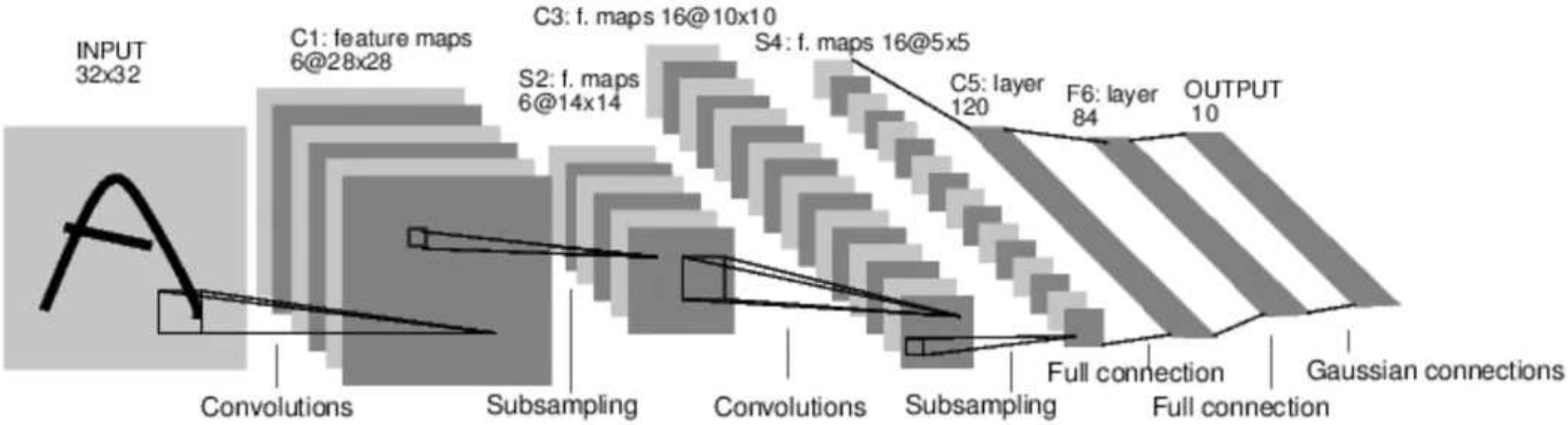




Deep Neural Network

- **Convolution Neural Network**
  - **CNN Case Study**
    - » LeNet-5

[LeCun et al., 1998]

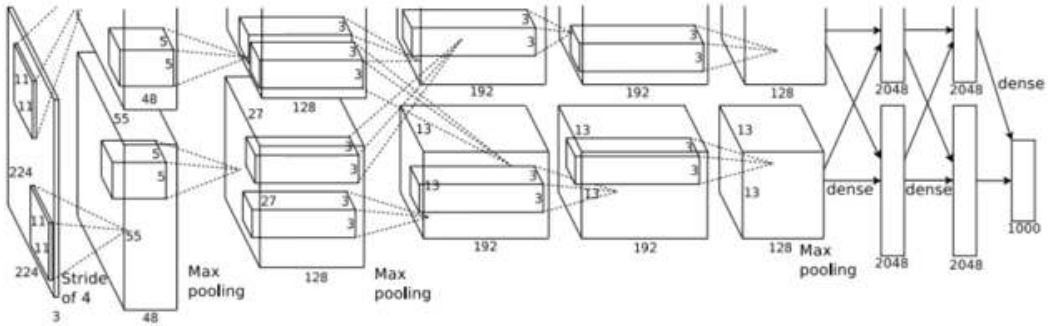


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Convolution Neural Network

CNN Case Study

» AlexNet [Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

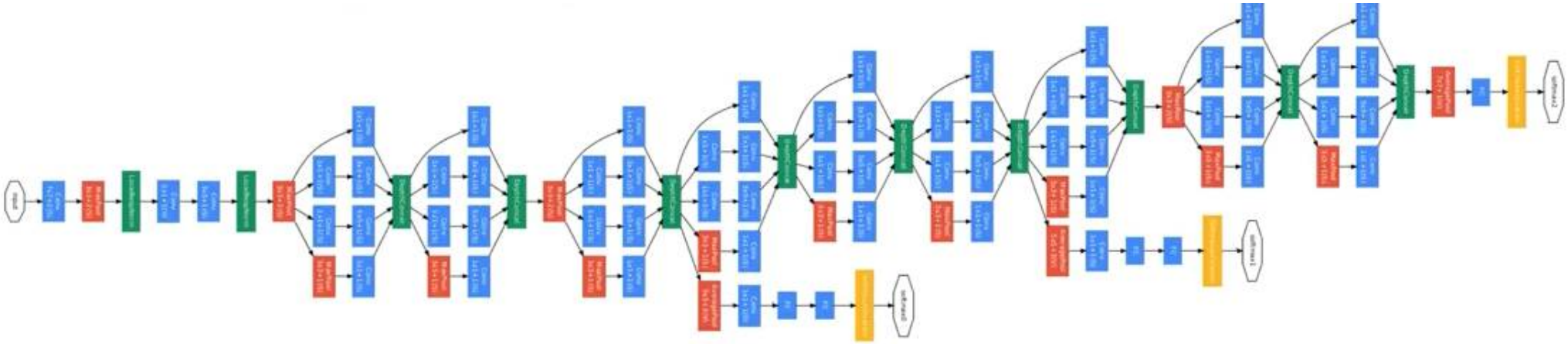
Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

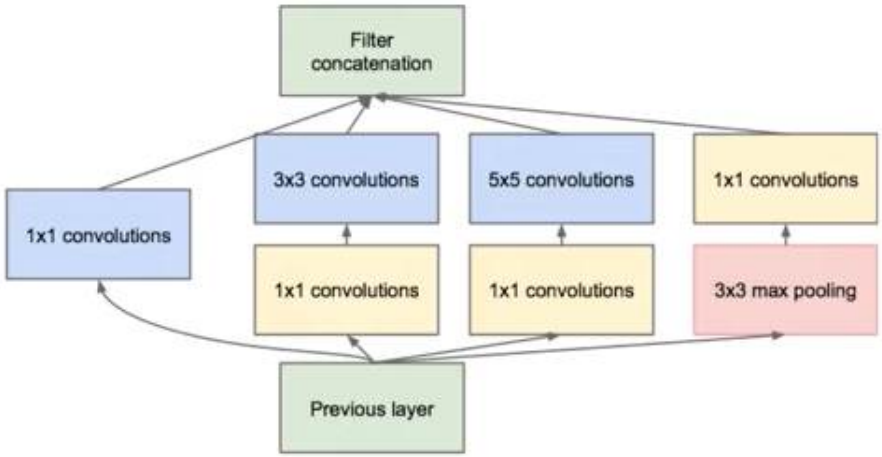
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Convolution Neural Network

CNN Case Study

- GoogLeNet [Szegedy et al. 2014]



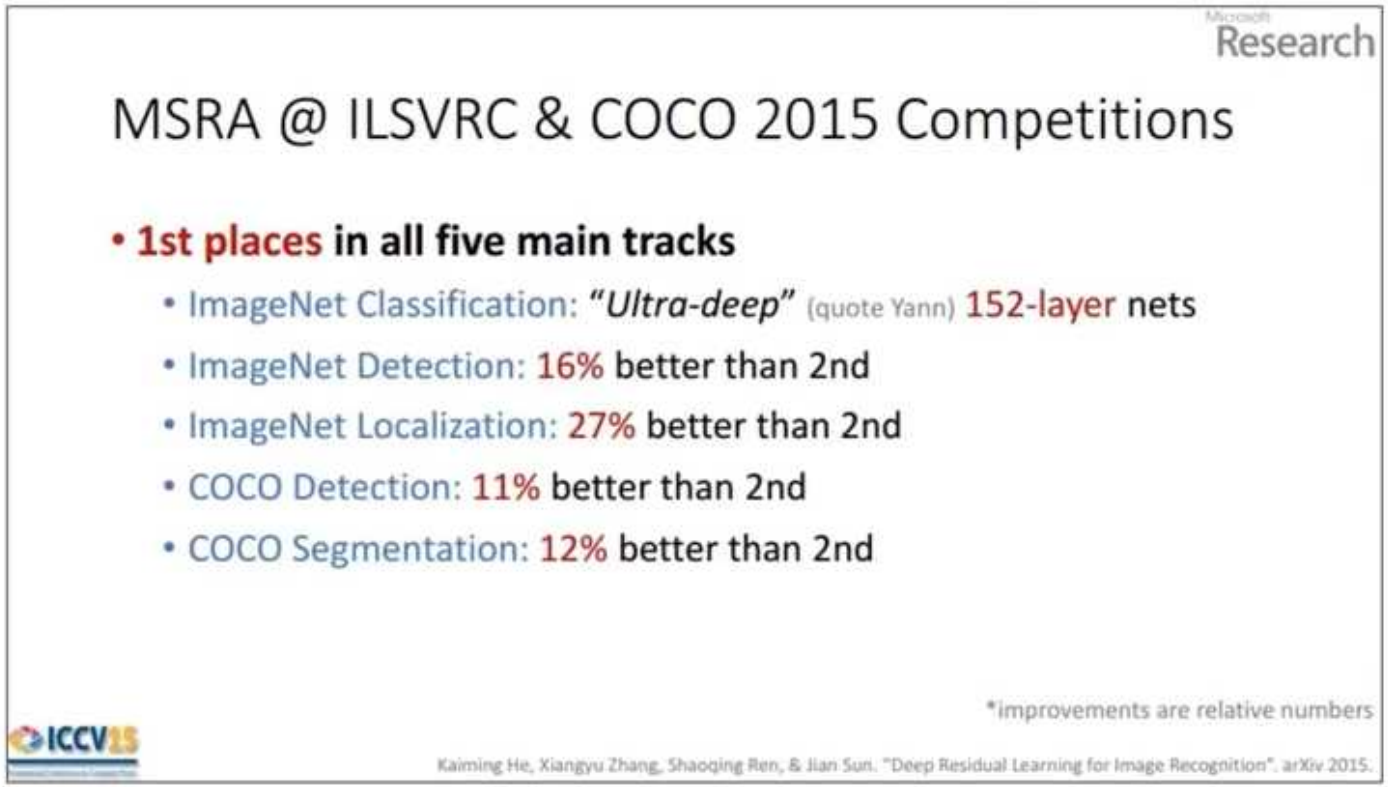


Inception module

ILSVRC 2014 winner (6.7% top 5 error)

- **Convolution Neural Network**
  - **CNN Case Study**
    - » ResNet [He et al. 2015]

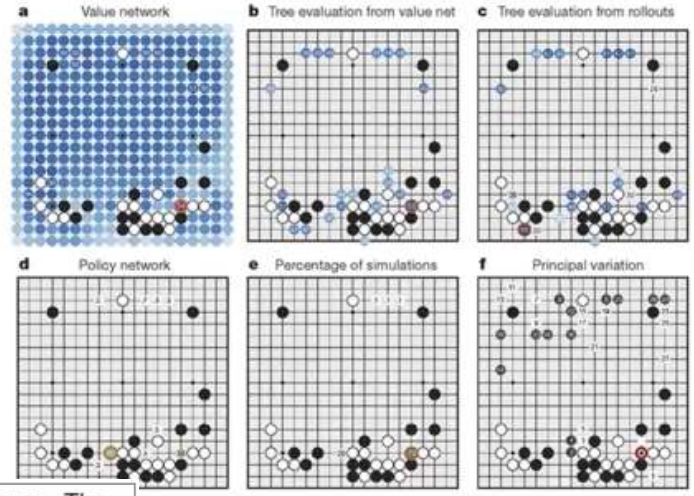
ILSVRC 2015 winner (3.6% top 5 error)



Convolution Neural Network

CNN Case Study

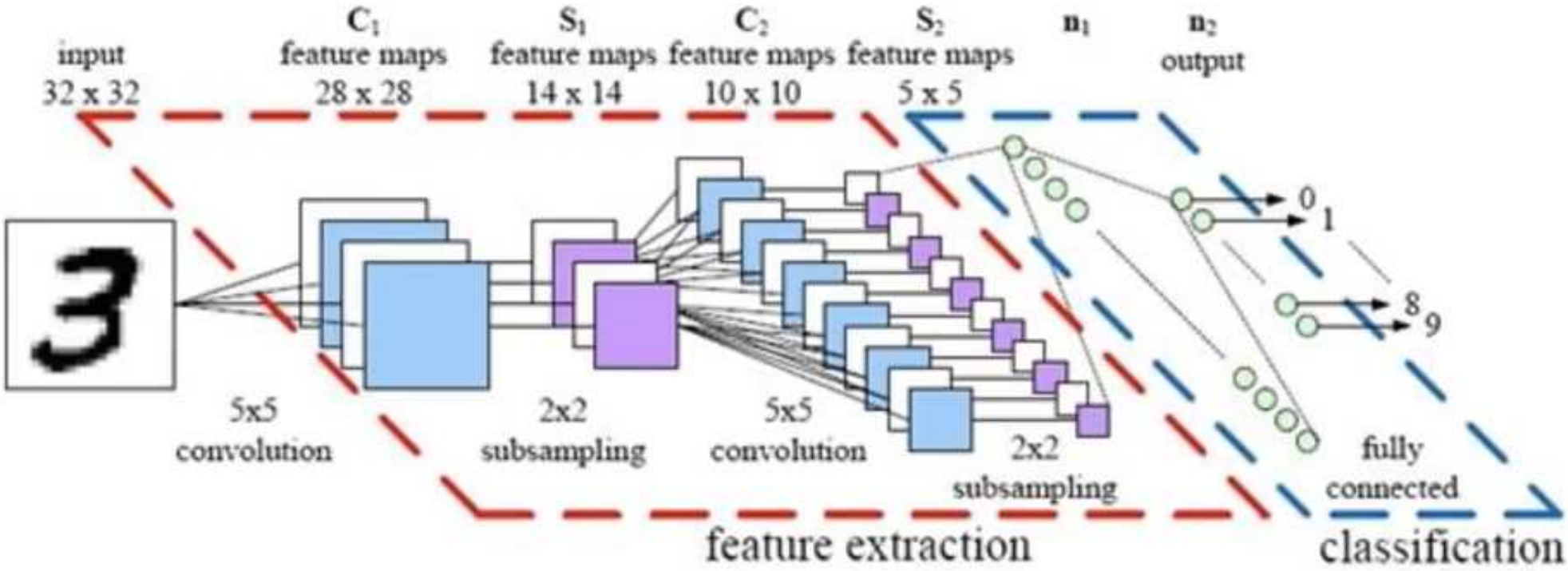
Deep Mind's AlphaGo



The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

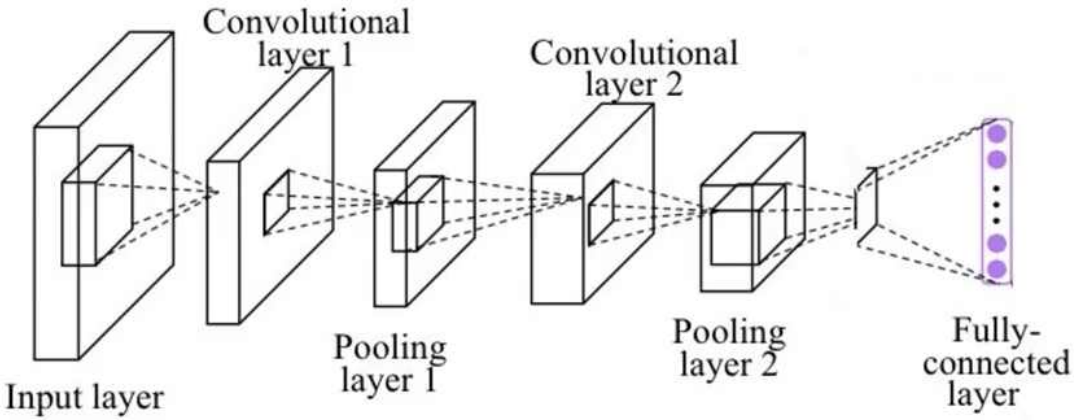
- **Convolution Neural Network**
  - **MNIST 99% using CNN**



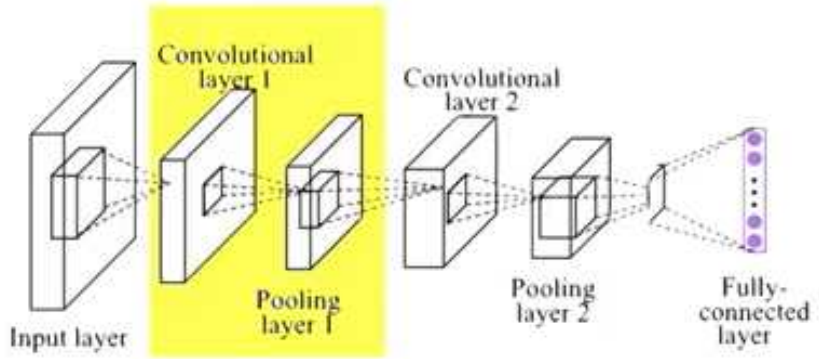
◆ Convolution Neural Network

– 실습: MNIST 99% using CNN

» Simple CNN: Layer1




```
# input placeholders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 10])

# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv     -> (?, 28, 28, 32)
#    Pool     -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
        strides=[1, 2, 2, 1], padding='SAME')
```

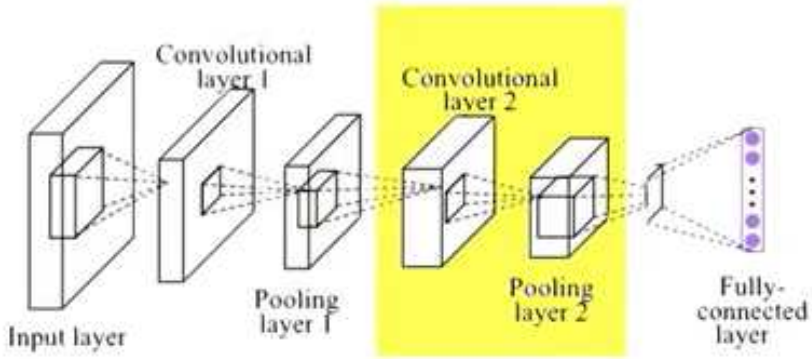


◆ Convolution Neural Network

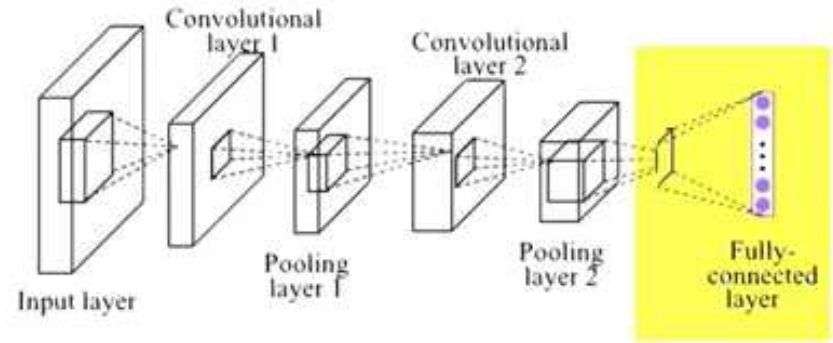
– 실습: MNIST 99% using CNN

» Simple CNN : Layer 3

```
# L2 ImgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.reshape(L2, [-1, 7 * 7 * 64])
```




```
L2 = tf.reshape(L2, [-1, 7 * 7 * 64])

# Final FC 7x7x64 inputs -> 10 outputs
W3 = tf.get_variable("W2", shape=[7 * 7 * 64, 10],
initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b
```

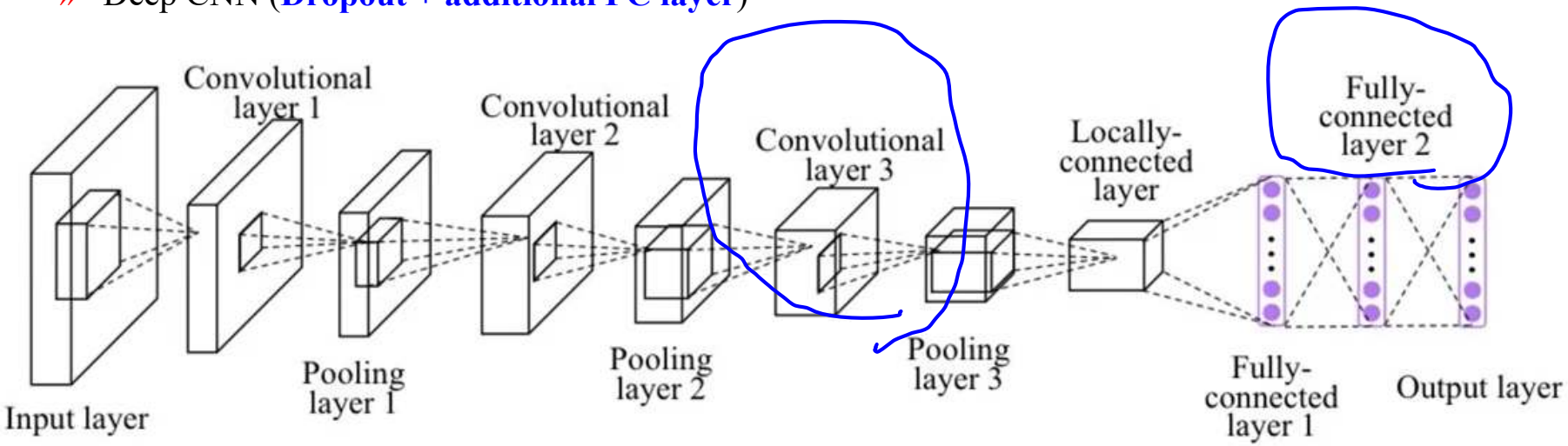



```
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

Accuracy: 98.85%

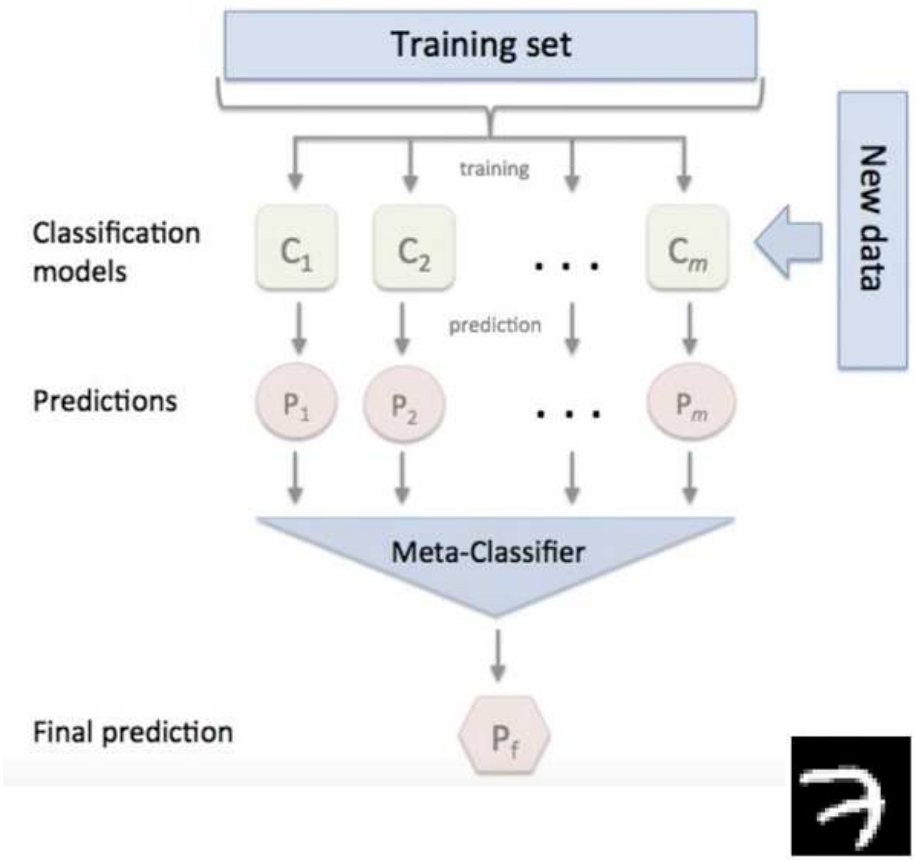
◆ Convolution Neural Network

- **실습: MNIST 99% using CNN**
  - » Deep CNN (**Dropout + additional FC layer**)



Accuracy: 99.38%

- **Convolution Neural Network**
  - **Ensemble**



Accuracy: 99.52%

	0	1	2	3	4	5	6	7	8	9
C_1 →	0.1	0.01	0.02	0.8	...	...	...	...	...	...
C_2 →	0.01	0.5	0.02	0.4	...	...	...	...	...	...
.										
.										
.										
C_m →	0.01	0.01	0.1	0.7	...	...	...	...	...	...
Sum	0.12	0.52	0.14	**1.9**	...	...	...	...	...	...

Thank you & Good luck !