27 Mar 2017

Basic of DL : XOR and DNN

ISL lab Seminar

Han-Sol Kang

Contents

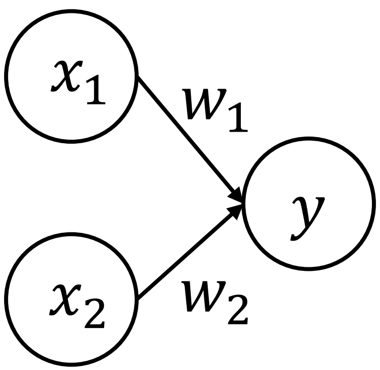
- Introduction
- XOR Problem
- Deep NN
- Implementation

Introduction

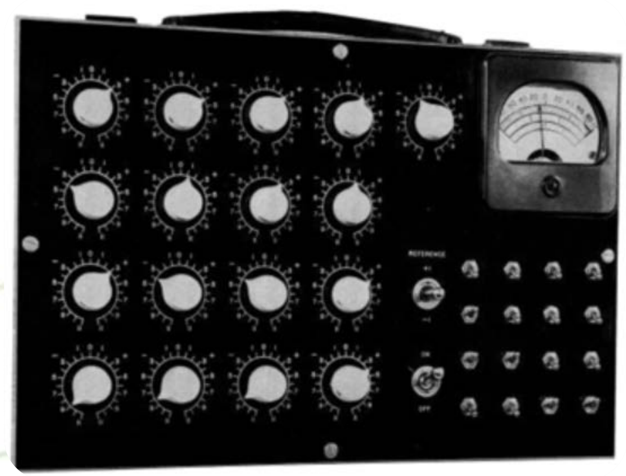
- Perceptron

Frank Rosenblatt(1957)

Frank Rosenblatt, ~1957: Perceptron



Widrow and Hoff, ~1960: Adaline/Madaline

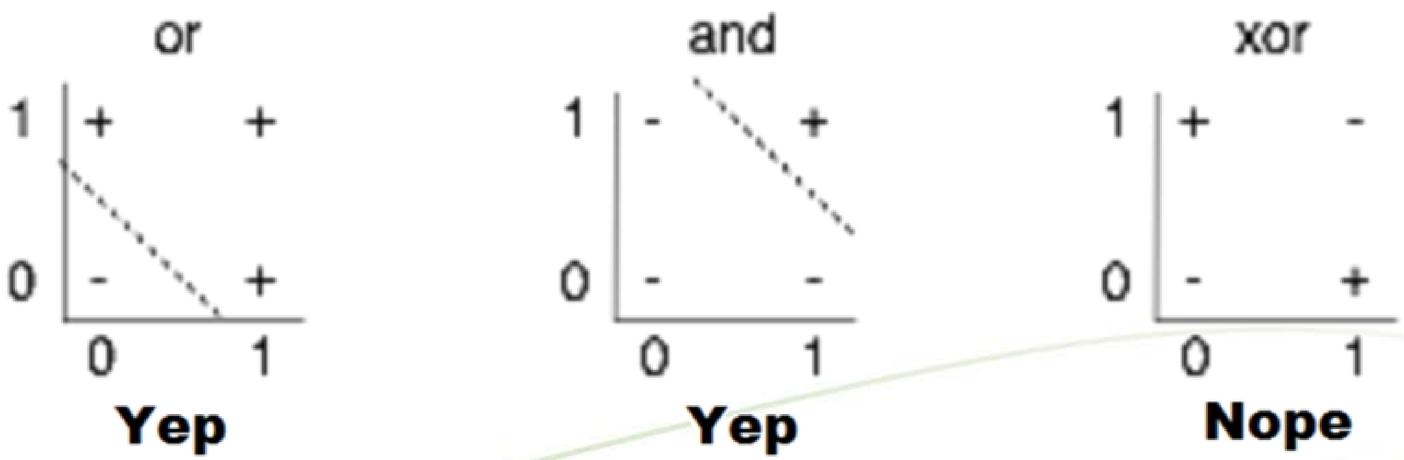


Introduction

- Perceptron

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.**

The New York Times July 08, 1958



Introduction

✫ Perceptron

Perceptrons (1969) by Marvin Minsky, founder of the MIT AI Lab

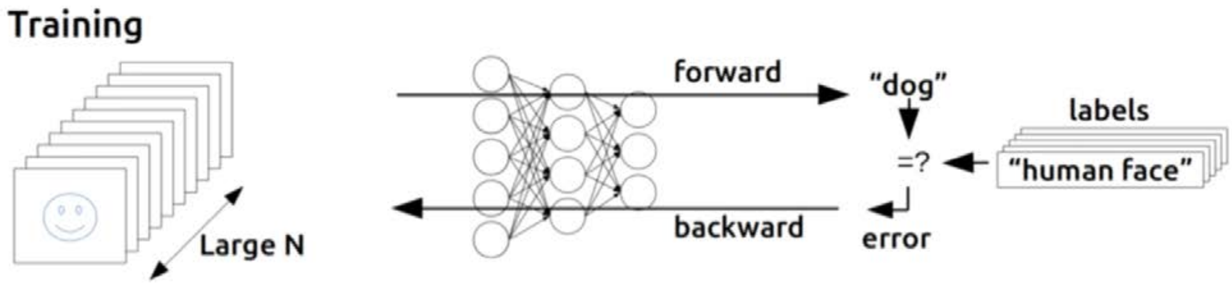
- We need to use MLP, multilayer perceptrons (multilayer neural nets)
- No one on earth had found a viable way to train MLPs good enough to learn such simple functions.

“No one on earth had found a viable way to train”

Introduction

- Backpropagation

1974, 1982 by Paul Werbos, 1986 by Hinton



Terminator 2 (1991)

JOHN: Can you learn? So you can be... you know. More human. Not such a dork all the time.

TERMINATOR: My CPU is a neural-net processor... a learning computer. But **Skynet** presets the switch to "read-only" when we are sent out alone.

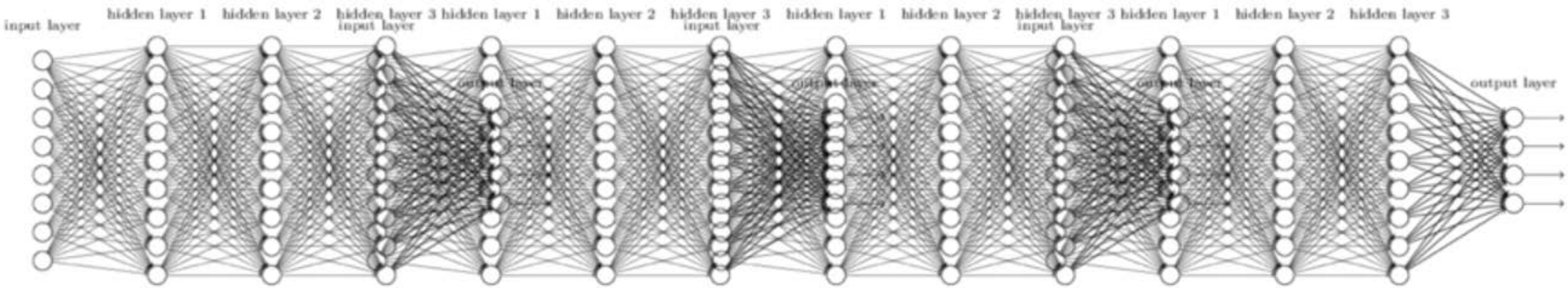
…

We'll learn how to **set** the neural net

Introduction

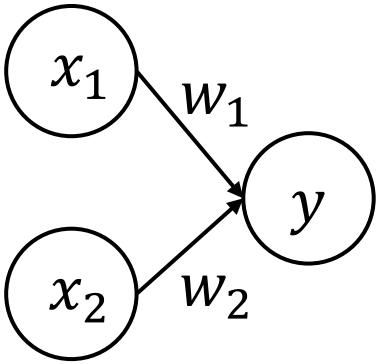
A BIG problem

- 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



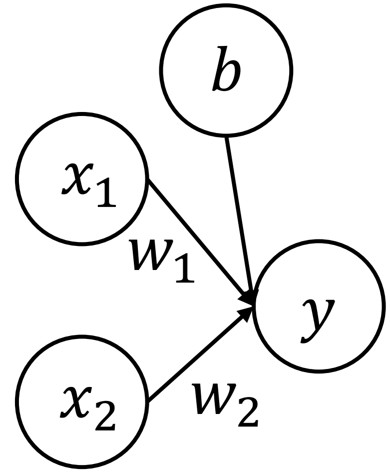
XOR Problem

- AND, OR, XOR using NN



$$y = \begin{cases} 0 \ (w_1 x_1 + w_2 x_2 \le \theta) \\ 1 \ (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

$$y = \begin{cases} 0 \ (b + w_1 x_1 + w_2 x_2 \le 0) \\ 1 \ (b + w_1 x_1 + w_2 x_2 > 0) \end{cases}$$



x1	x2	y
0	0	0
1	0	0
0	1	0
1	1	1

$$(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$$

x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	1

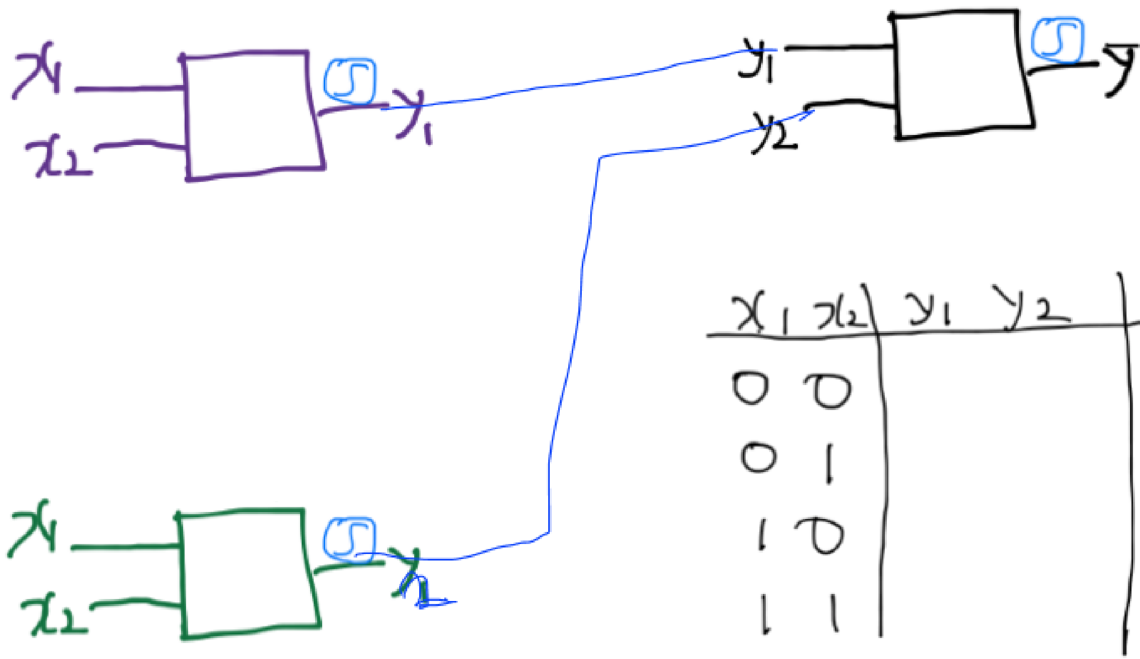
$$(w_1, w_2, \theta) = (0.5, 0.5, 0.2)$$

x1	x2	y
0	0	1
1	0	1
0	1	1
1	1	0

$$(w_1, w_2, \theta) = (-0.5, -0.5, -0.7)$$

XOR Problem

- AND, OR, XOR using NN



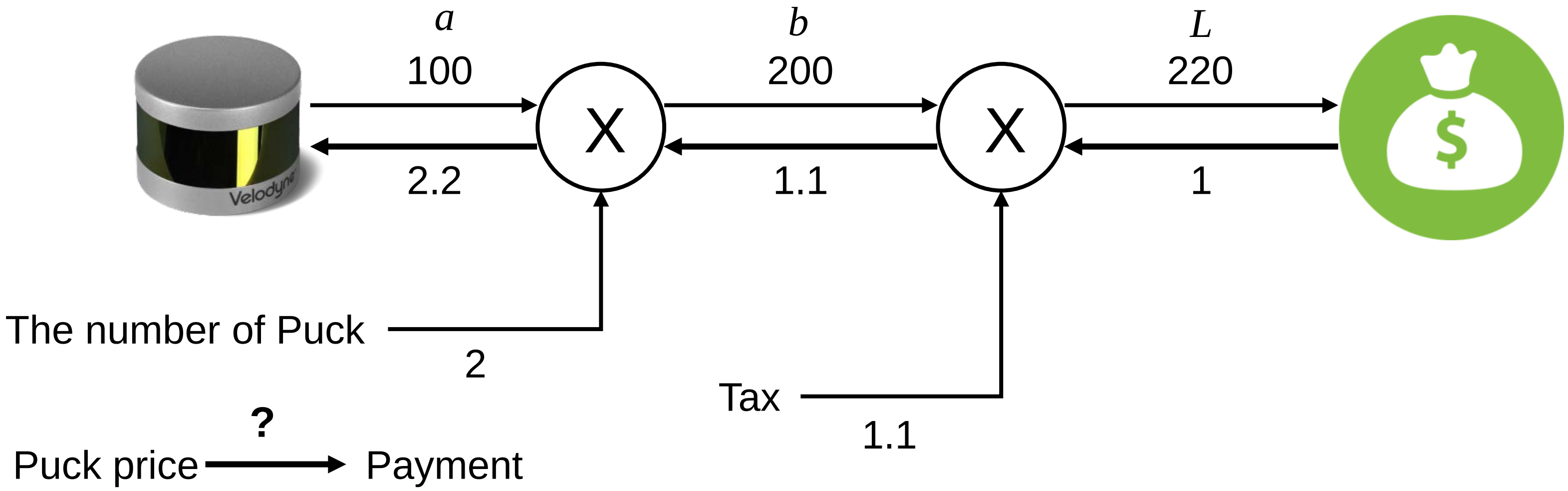
x_1	x_2	y_1	y_2	$\bar{y}$	XOR
0	0				0
0	1				1
1	0				1
1	1				0

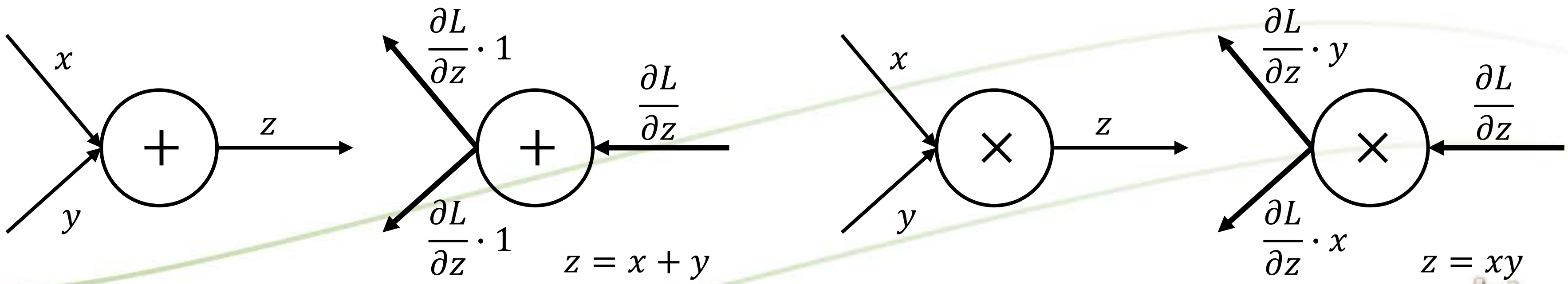
"No one on earth had found a viable way to train"

XOR Problem

Backpropagation

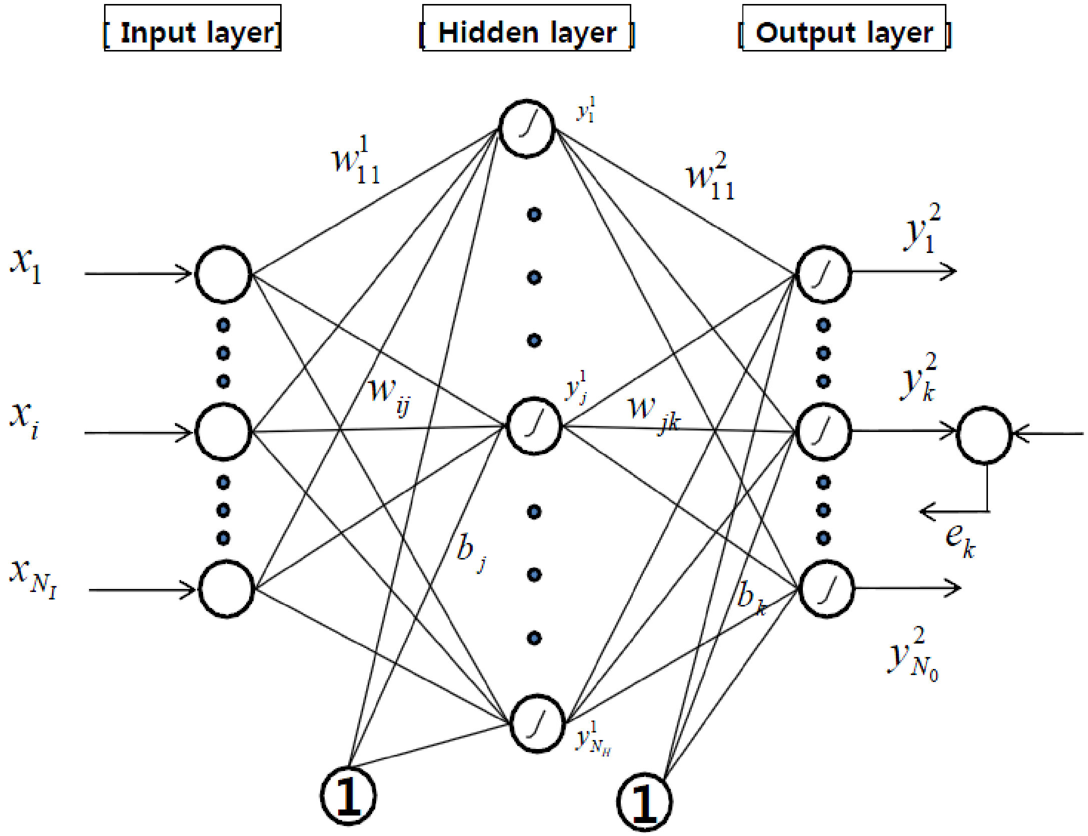
Kurt bought two pucks, one for 100. Get the payment amount. However, consumption tax is charged at 10%.





XOR Problem

- Backpropagation



$$E = \frac{1}{2}\sum_{k}^{NO} e_k^{\ 2}$$

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$$

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

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

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

XOR Problem

- Backpropagation

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial w_{jk}} \quad \Rightarrow \quad \boxed{\Delta w_{jk}(t) = -\eta \frac{\partial E}{\partial w_{jk}} = \eta e_k f'(s_k) y_j}$$

$$= e_k \frac{\partial e_k}{\partial w_{jk}} \quad \Leftarrow \quad \frac{\partial E}{\partial e_k} = \frac{1}{2}\frac{\partial e_k^2}{\partial e_k} = e_k$$

$$= e_k \frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial w_{jk}}$$

$$= -e_k \frac{\partial y_k}{\partial w_{jk}} \quad \Leftarrow \quad \frac{\partial e_k}{\partial y_k} = \frac{\partial (yd_k - y_k)}{\partial y_k} = -1$$

$$= -e_k \frac{\partial y_k}{\partial s_k}\frac{\partial s_k}{\partial w_{jk}}$$

$$= -e_k f'(s_k)\frac{\partial s_k}{\partial w_{jk}} \quad \Leftarrow \quad \frac{\partial y_k}{\partial s_k} = \frac{\partial f(s_k)}{\partial s_k} = f'(s_k)$$

$$= -e_k f'(s_k) y_j \quad \Leftarrow \quad \frac{\partial s_k}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}\sum_{j=1}^{NH} w_{jk}y_j + b_k = y_j$$

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial b_k} \quad \Rightarrow \quad \boxed{\Delta b_k(t) = -\eta \frac{\partial E}{\partial b_k} = \eta e_k f'(s_k)}$$

$$= e_k \frac{\partial e_k}{\partial b_k} \quad \Leftarrow \quad \frac{\partial E}{\partial e_k} = \frac{1}{2}\frac{\partial e_k^2}{\partial e_k} = e_k$$

$$= e_k \frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial b_k}$$

$$= -e_k \frac{\partial y_k}{\partial b_k} \quad \Leftarrow \quad \frac{\partial e_k}{\partial y_k} = \frac{\partial (yd_k - y_k)}{\partial y_k} = -1$$

$$= -e_k \frac{\partial y_k}{\partial s_k}\frac{\partial s_k}{\partial b_k}$$

$$= -e_k f'(s_k)\frac{\partial s_k}{\partial b_k} \quad \Leftarrow \quad \frac{\partial y_k}{\partial s_k} = \frac{\partial f(s_k)}{\partial s_k} = f'(s_k)$$

$$= -e_k f'(s_k) y_j \quad \Leftarrow \quad \frac{\partial s_k}{\partial b_k} = \frac{\partial}{\partial b_k}\sum_{j=1}^{NH} w_{jk}y_j + b_k = 1$$

XOR Problem

- Backpropagation

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}} = \eta f'(s_j) x_i \sum_{k=1}^{No} e_k f'(s_k) w_{jk}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial w_{ij}}$$

$$= e_k \frac{\partial e_k}{\partial w_{ij}} \qquad \frac{\partial E}{\partial e_k} = \frac{1}{2}\frac{\partial e_k^2}{\partial e_k} = e_k$$

$$= e_k \frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial w_{ij}}$$

$$= -e_k \frac{\partial y_k}{\partial w_{ij}} \qquad \frac{\partial e_k}{\partial y_k} = \frac{\partial (yd_k - y_k)}{\partial y_k} = -1$$

$$= -e_k \frac{\partial y_k}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}}$$

$$= -e_k \sum_{k=1}^{No} \frac{\partial y_k}{\partial s_k}\frac{\partial s_k}{\partial w_{ij}} \qquad \frac{\partial y_k}{\partial y_j} = \sum_{k=1}^{No} \frac{\partial y_k}{\partial s_k}\frac{\partial s_k}{\partial y_j}$$

$$= -e_k \sum_{k=1}^{No} f'(s_k)\frac{\partial s_k}{\partial w_{ij}} \qquad \frac{\partial y_k}{\partial s_k} = f'(s_k)$$

$$= -e_k \sum_{k=1}^{No} f'(s_k)\frac{\partial s_k}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}}$$

$$= -e_k \sum_{k=1}^{No} f'(s_k) w_{jk}\frac{\partial y_j}{\partial w_{ij}} \qquad \frac{\partial s_k}{\partial y_j} = \frac{\partial (\sum_{j=1}^{NH} w_{jk} y_j + b_k)}{\partial y_j} = w_{jk}$$

$$= -e_k \sum_{k=1}^{No} f'(s_k) w_{jk}\frac{\partial y_j}{\partial s_j}\frac{\partial s_j}{\partial w_{ij}}$$

$$= -e_k \sum_{k=1}^{No} f'(s_k) w_{jk} f'(s_j)\frac{\partial s_j}{\partial w_{ij}} \qquad \frac{\partial y_j}{\partial s_j} = f'(s_j)$$

$$= -e_k \sum_{k=1}^{No} f'(s_k) w_{jk} f'(s_j) x_i \qquad \frac{\partial s_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{i=1}^{NI} w_{ij} x_i + b_j = x_i$$

XOR Problem

- Backpropagation

$$\Delta b_j(t) = -\eta \frac{\partial E}{\partial b_j} = \eta f'(s_j) \sum_{k=1}^{No} e_k f'(s_k) w_{jk}$$

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial b_j}$$

$$= e_k \frac{\partial e_k}{\partial b_j} \quad \Leftarrow \quad \frac{\partial E}{\partial e_k} = \frac{1}{2}\frac{\partial e_k^2}{\partial e_k} = e_k$$

$$= e_k \frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial b_j}$$

$$= -e_k \frac{\partial y_k}{\partial b_j} \quad \Leftarrow \quad \frac{\partial e_k}{\partial y_k} = \frac{\partial (yd_k - y_k)}{\partial y_k} = -1$$

$$= -e_k \frac{\partial y_k}{\partial y_j}\frac{\partial y_j}{\partial b_j}$$

$$= -e_k \sum_{k=1}^{No} \frac{\partial y_k}{\partial s_k}\frac{\partial s_k}{\partial b_j} \quad \Leftarrow \quad \frac{\partial y_k}{\partial y_j} = \sum_{k=1}^{No} \frac{\partial y_k}{\partial s_k}\frac{\partial s_k}{\partial y_j}$$

$$= -e_k \sum_{k=1}^{No} f'(s_k)\frac{\partial s_k}{\partial b_j} \quad \Leftarrow \quad \frac{\partial y_k}{\partial s_k} = f'(s_k)$$

$$= -e_k \sum_{k=1}^{No} f'(s_k)\frac{\partial s_k}{\partial y_j}\frac{\partial y_j}{\partial b_j}$$

$$= -e_k \sum_{k=1}^{No} f'(s_k) w_{jk}\frac{\partial y_j}{\partial b_j} \quad \Leftarrow \quad \frac{\partial s_k}{\partial y_j} = \frac{\partial (\sum_{j=1}^{NH} w_{jk} y_j + b_k)}{\partial y_j} = w_{jk}$$

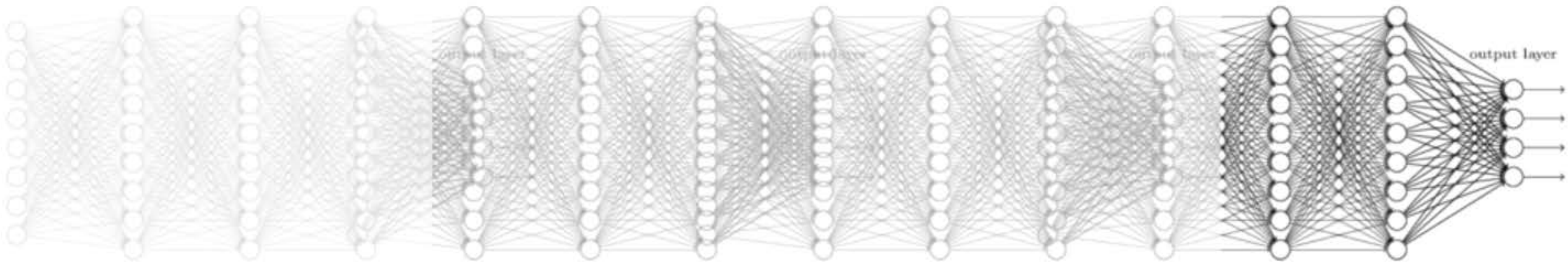
$$= -e_k \sum_{k=1}^{No} f'(s_k) w_{jk}\frac{\partial y_j}{\partial s_j}\frac{\partial s_j}{\partial b_j}$$

$$= -e_k \sum_{k=1}^{No} f'(s_k) w_{jk} f'(s_j)\frac{\partial s_j}{\partial b_j} \quad \Leftarrow \quad \frac{\partial y_j}{\partial s_j} = f'(s_j)$$

$$= -e_k \sum_{k=1}^{No} f'(s_k) w_{jk} f'(s_j) x_i \quad \Leftarrow \quad \frac{\partial s_j}{\partial b_j} = \frac{\partial}{\partial b_j}\sum_{i=1}^{NI} w_{ij} x_i + b_j = 1$$

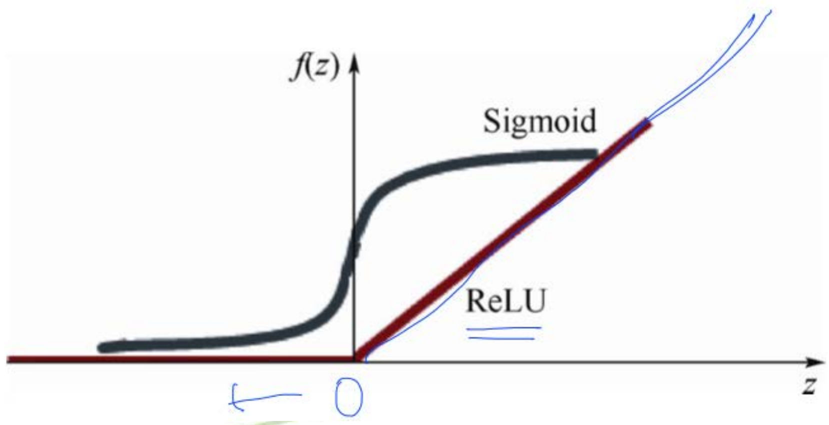
Deep NN

- Vanishing Gradient



“We used the wrong type of non-linearity”

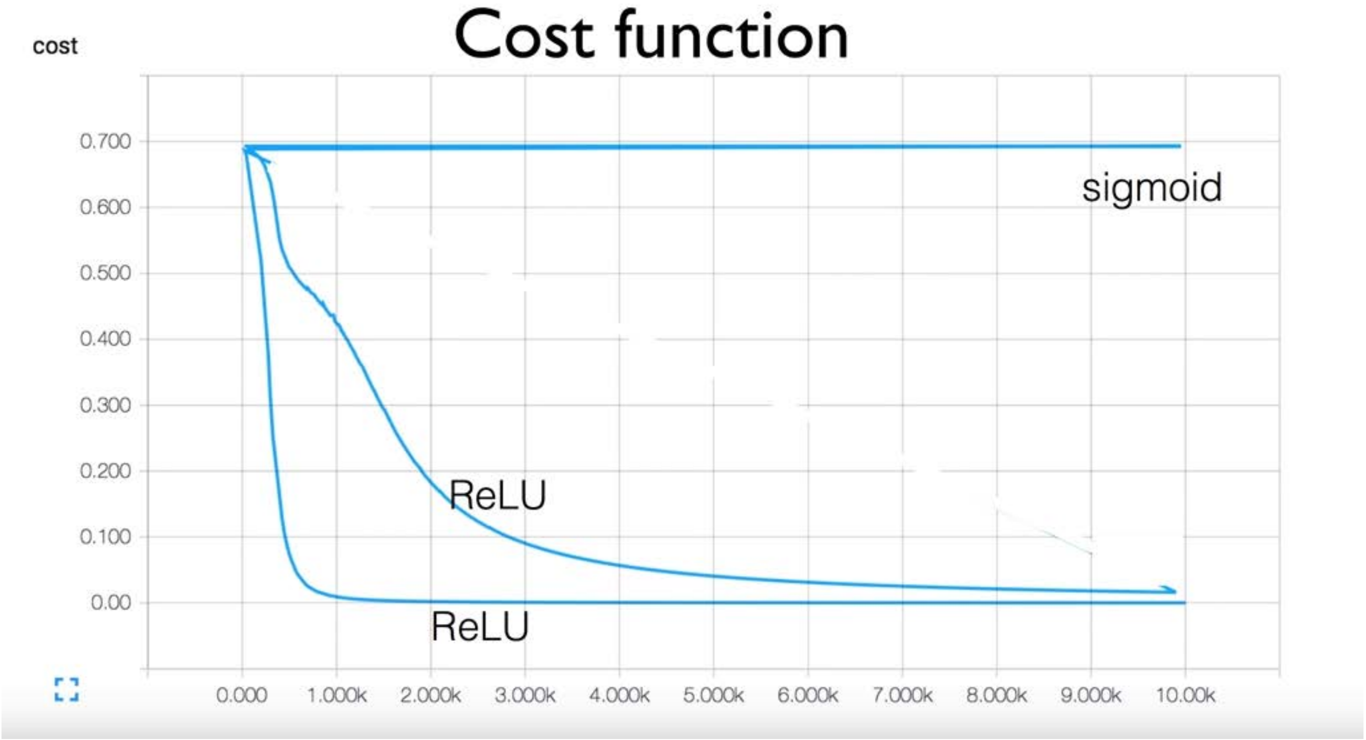
Geoffrey Hinton



ReLU(Rectified Linear Unit)

Deep NN

- Initialize weights



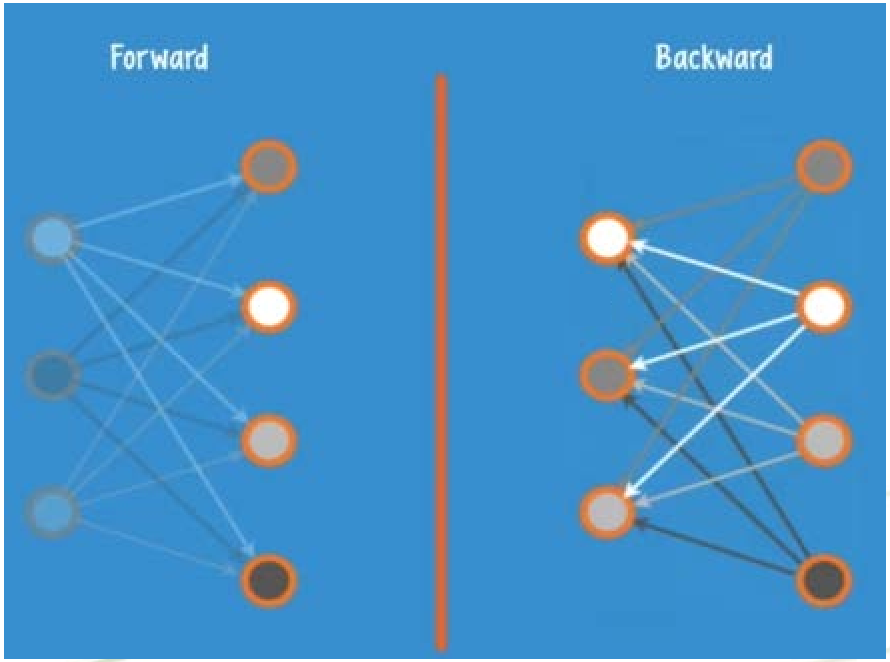
“We initialized the weights in a stupid way”

Geoffrey Hinton

Deep NN

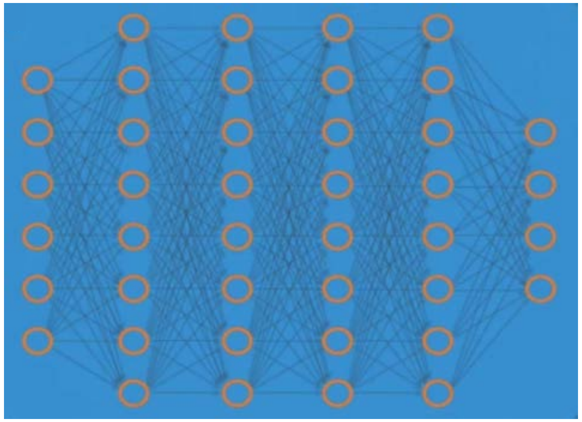
- Initialize weights

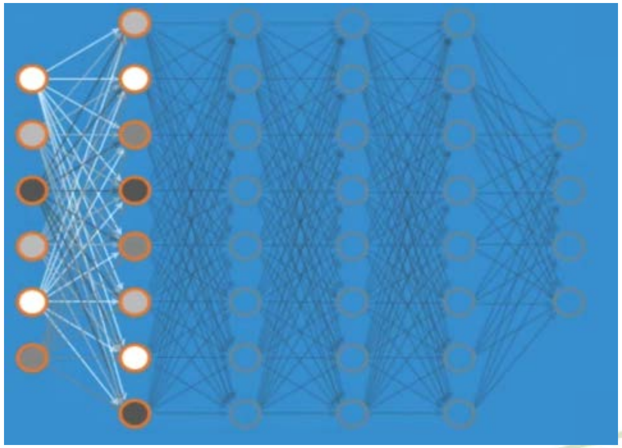
Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets”
- Restricted Boltzmann Machine (RBM)

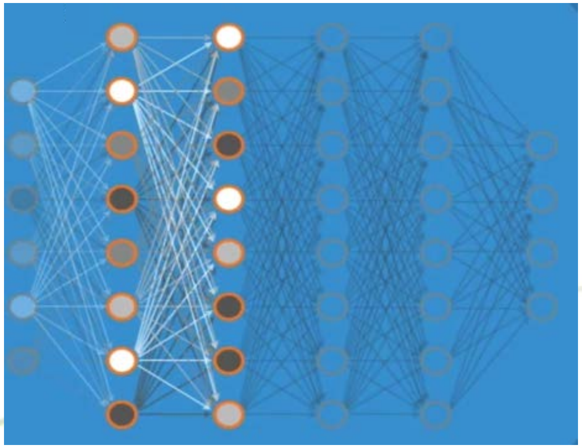


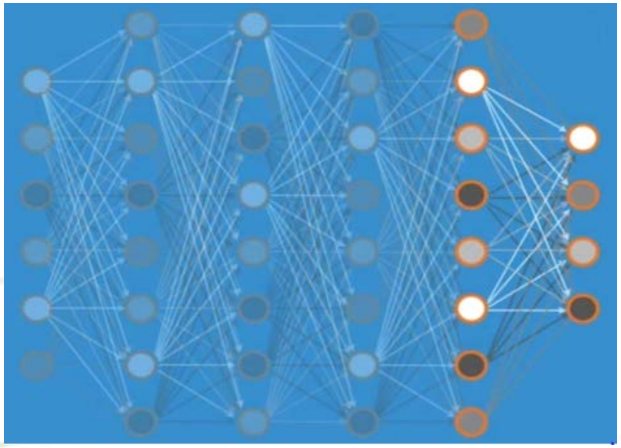
Deep NN

- Initialize weights









Deep NN

- Initialize weights(Xavier/He initialization)

  - Makes sure the weights are ‘just right’, not too small, not too big

  - Using number of input (fan_in) and output (fan_out)

Xavier

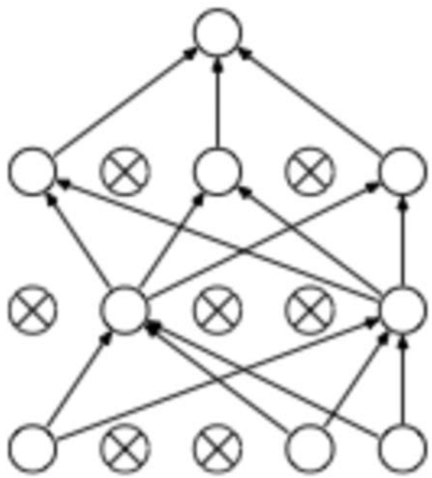
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

He

W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)

Deep NN

- Dropout



(b) After applying dropout.

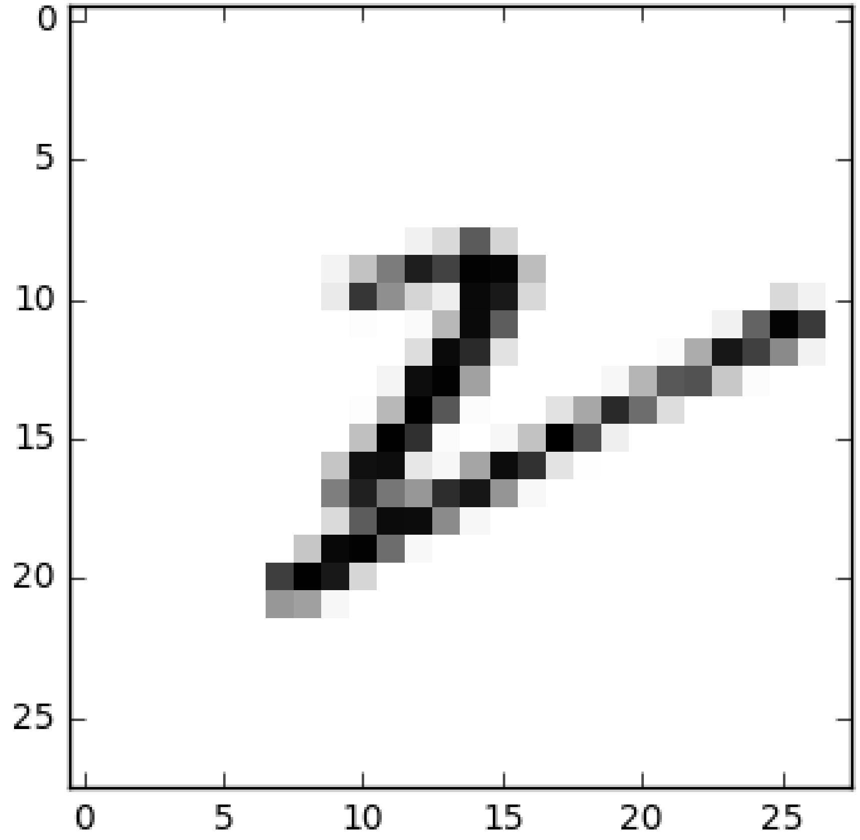
Implementation

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)


# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W = tf.Variable(tf.random_normal([784, 10]))
b = tf.Variable(tf.random_normal([10]))

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

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

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

```
# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
print('Learning Finished!')

# Test model and check accuracy
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}))

# 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), cmap='Greys',
interpolation='nearest')
plt.show()
```

Implementation

```
('Epoch:', '0001', 'cost =', '5.916487252')
('Epoch:', '0002', 'cost =', '1.863573338')
('Epoch:', '0003', 'cost =', '1.162345760')
('Epoch:', '0004', 'cost =', '0.894605613')
('Epoch:', '0005', 'cost =', '0.753347107')
('Epoch:', '0006', 'cost =', '0.665160576')
('Epoch:', '0007', 'cost =', '0.604045915')
('Epoch:', '0008', 'cost =', '0.558101759')
('Epoch:', '0009', 'cost =', '0.523281238')
('Epoch:', '0010', 'cost =', '0.495043325')
('Epoch:', '0011', 'cost =', '0.471873087')
('Epoch:', '0012', 'cost =', '0.452187982')
('Epoch:', '0013', 'cost =', '0.435230404')
('Epoch:', '0014', 'cost =', '0.420703621')
('Epoch:', '0015', 'cost =', '0.407859434')
Learning Finished!
('Accuracy:', 0.90359998)
('Label: ', array([2]))
('Prediction: ', array([1]))
```



Implementation

```python
import tensorflow as tf
import random

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3
# 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)
```

```python
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

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

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

# Test model and check accuracy
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}))

# 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]}))
```

Implementation

Softmax

```
('Epoch:', '0001', 'cost =', '5.916487252')
('Epoch:', '0002', 'cost =', '1.863573338')
('Epoch:', '0003', 'cost =', '1.162345760')
('Epoch:', '0004', 'cost =', '0.894605613')
('Epoch:', '0005', 'cost =', '0.753347107')
('Epoch:', '0006', 'cost =', '0.665160576')
('Epoch:', '0007', 'cost =', '0.604045915')
('Epoch:', '0008', 'cost =', '0.558101759')
('Epoch:', '0009', 'cost =', '0.523281238')
('Epoch:', '0010', 'cost =', '0.495043325')
('Epoch:', '0011', 'cost =', '0.471873087')
('Epoch:', '0012', 'cost =', '0.452187982')
('Epoch:', '0013', 'cost =', '0.435230404')
('Epoch:', '0014', 'cost =', '0.420703621')
('Epoch:', '0015', 'cost =', '0.407859434')
Learning Finished!

('Accuracy:',0.90359998)
```

NN

```
('Epoch:', '0001', 'cost =', '164.116649972')
('Epoch:', '0002', 'cost =', '41.866736450')
('Epoch:', '0003', 'cost =', '26.609068727')
('Epoch:', '0004', 'cost =', '18.717741623')
('Epoch:', '0005', 'cost =', '13.838593242')
('Epoch:', '0006', 'cost =', '10.368780142')
('Epoch:', '0007', 'cost =', '7.660989459')
('Epoch:', '0008', 'cost =', '5.893673751')
('Epoch:', '0009', 'cost =', '4.475466314')
('Epoch:', '0010', 'cost =', '3.376285574')
('Epoch:', '0011', 'cost =', '2.614971533')
('Epoch:', '0012', 'cost =', '1.986375339')
('Epoch:', '0013', 'cost =', '1.538742549')
('Epoch:', '0014', 'cost =', '1.246197118')
('Epoch:', '0015', 'cost =', '0.954491639')
Learning Finished!

('Accuracy:', 0.95029998)
```

Implementation

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

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# dropout (keep_prob) rate  0.7 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

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
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)
```

▪

▪

▪

Implementation

NN

('Epoch:', '0001', 'cost =', '164.116649972')
('Epoch:', '0002', 'cost =', '41.866736450')
('Epoch:', '0003', 'cost =', '26.609068727')
('Epoch:', '0004', 'cost =', '18.717741623')
('Epoch:', '0005', 'cost =', '13.838593242')
('Epoch:', '0006', 'cost =', '10.368780142')
('Epoch:', '0007', 'cost =', '7.660989459')
('Epoch:', '0008', 'cost =', '5.893673751')
('Epoch:', '0009', 'cost =', '4.475466314')
('Epoch:', '0010', 'cost =', '3.376285574')
('Epoch:', '0011', 'cost =', '2.614971533')
('Epoch:', '0012', 'cost =', '1.986375339')
('Epoch:', '0013', 'cost =', '1.538742549')
('Epoch:', '0014', 'cost =', '1.246197118')
('Epoch:', '0015', 'cost =', '0.954491639')
Learning Finished!

('Accuracy:', 0.95029998)

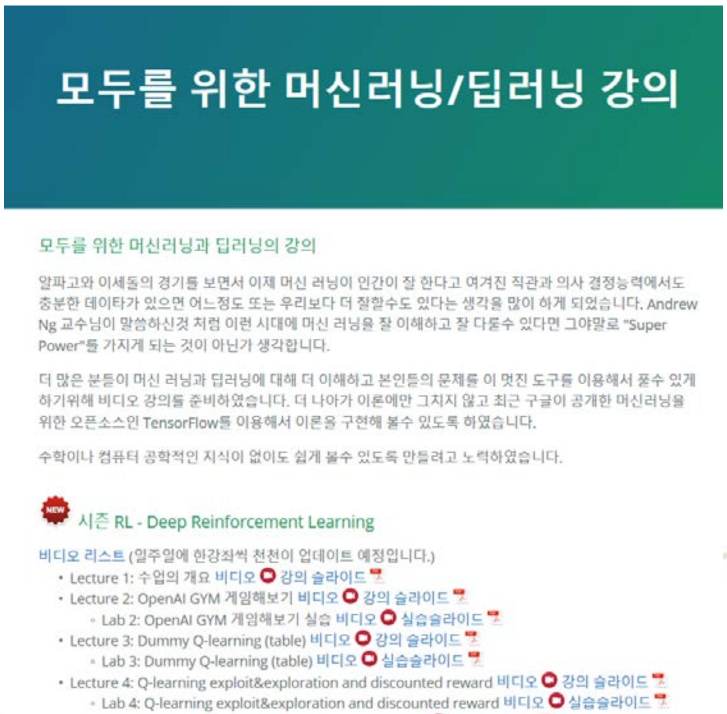
NN(Xavier,Dropout)

('Epoch:', '0001', 'cost =', '0.475521204')
('Epoch:', '0002', 'cost =', '0.174723941')
('Epoch:', '0003', 'cost =', '0.132422534')
('Epoch:', '0004', 'cost =', '0.110649394')
('Epoch:', '0005', 'cost =', '0.094175926')
('Epoch:', '0006', 'cost =', '0.082326408')
('Epoch:', '0007', 'cost =', '0.078204827')
('Epoch:', '0008', 'cost =', '0.067890784')
('Epoch:', '0009', 'cost =', '0.065861956')
('Epoch:', '0010', 'cost =', '0.059872363')
('Epoch:', '0011', 'cost =', '0.056675084')
('Epoch:', '0012', 'cost =', '0.053590286')
('Epoch:', '0013', 'cost =', '0.049909270')
('Epoch:', '0014', 'cost =', '0.049200659')
('Epoch:', '0015', 'cost =', '0.048159967')
Learning Finished!

('Accuracy:', 0.98250002)

Appendix

- Sung Hun Kim & Deep Learning from Scratch



모두를 위한 머신러닝/딥러닝 강의

모두를 위한 머신러닝과 딥러닝의 강의

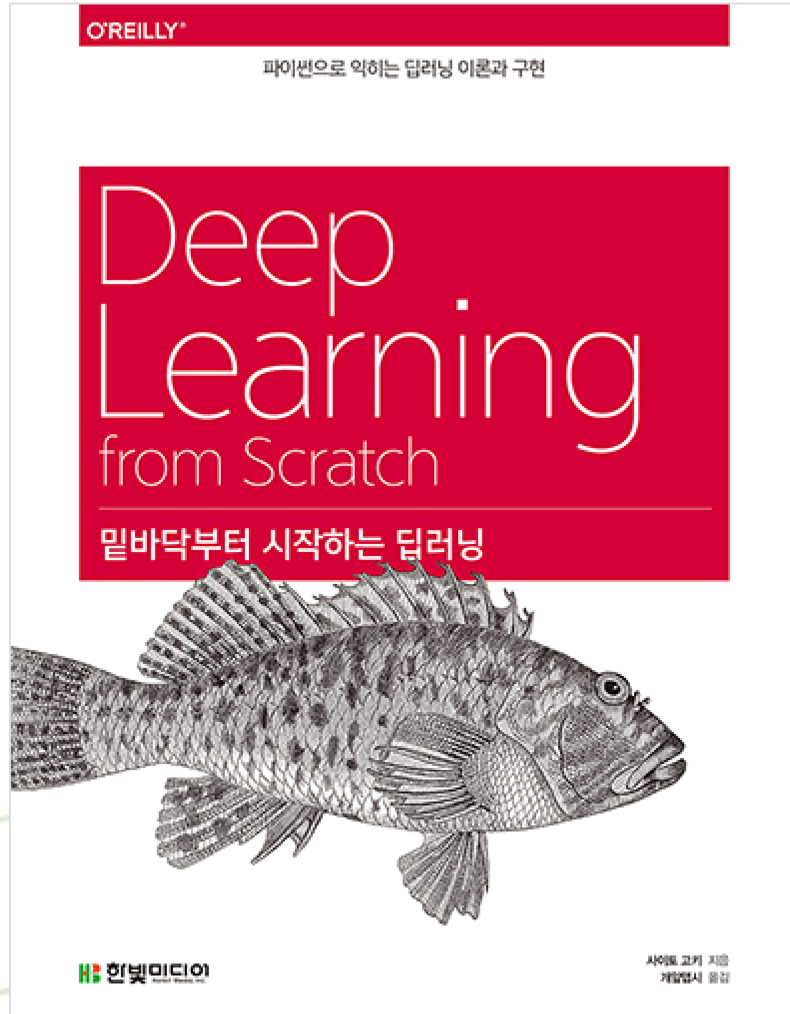
알파고와 이세돌의 경기를 보면서 이제 머신 러닝이 인간이 잘 한다고 여겨진 직관과 의사 결정능력에서도 충분한 데이타가 있으면 어느정도 또는 우리보다 더 잘할수도 있다는 생각을 많이 하게 되었습니다. Andrew Ng 교수님이 말씀하신것 처럼 이런 시대에 머신 러닝을 잘 이해하고 잘 다룰수 있다면 그야말로 "Super Power"를 가지게 되는 것이 아닌가 생각합니다.

더 많은 분들이 머신 러닝과 딥러닝에 대해 더 이해하고 본인들의 문제를 이 멋진 도구를 이용해서 풀수 있게 하기위해 비디오 강의를 준비하였습니다. 더 나아가 이론에만 그치지 않고 최근 구글이 공개한 머신러닝을 위한 오픈소스인 TensorFlow를 이용해서 이론을 구현해 볼수 있도록 하였습니다.

수학이나 컴퓨터 공학적인 지식이 없이도 쉽게 볼수 있도록 만들려고 노력하였습니다.

NEW 시즌 RL - Deep Reinforcement Learning

비디오 리스트 (일주일에 한강좌씩 천천이 업데이트 예정입니다.)
- Lecture 1: 수업의 개요 비디오 강의 슬라이드
- Lecture 2: OpenAI GYM 게임해보기 비디오 강의 슬라이드
  - Lab 2: OpenAI GYM 게임해보기 실습 비디오 실습슬라이드
- Lecture 3: Dummy Q-learning (table) 비디오 강의 슬라이드
  - Lab 3: Dummy Q-learning (table) 비디오 실습슬라이드
- Lecture 4: Q-learning exploit&exploration and discounted reward 비디오 강의 슬라이드
  - Lab 4: Q-learning exploit&exploration and discounted reward 비디오 실습슬라이드



Q & A

Thank You!!!