

과제명	머신러닝 12주차 과제
-----	--------------



- 과 목 명 머신러닝
- 담당교수 김 용 운
- 학부(과) 컴퓨터소프트웨어공학과
- 학 년 4
- 분반번호 01
- 학 번 20173147
- 이 름 이 명 진
- 연 락 처 010-2999-7748
- 제 출 일 2020 년 07 월 03 일




원광대학교
WONKWANG UNIVERSITY

12-01 Lab_딥러닝 (1)

sigmoid는 layer를 3장,4장 까지 쌓는 데는 문제가 없지만 그 이상(100개,200개)으로 증가하면 ~~ gradient 현상이 발생하게된다.

- XOR data set

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \oplus B$		<table> <thead> <tr> <th>A</th><th>B</th><th>X</th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

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

xor => A= x1 , B=x2

xor은 array 형태로 들어가게 될 것임.

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

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
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=0.1).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)
```

XOR with
logistic regression?

But
it doesn't work!

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

예측값도 0.5정도로 나옴.

- Wide NN for XOR => 네트워크를 깊게 쌓는 것.

```
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, 1]), name='weight2')
```

```
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
```

```
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)
```

[2,10], [10,1]	[2,2], [2,1]
Hypothesis:	Hypothesis:
[[0.00358802]	[[0.01338218]
[0.99366933]	[0.98166394]
[0.99204296]	[0.98809403]
[0.0095663]]	[0.01135799]]
Correct:	Correct:
[[0.]	[[0.]
[1.]	[1.]
[1.]	[1.]
[0.]]	[0.]]
Accuracy: 1.0	Accuracy: 1.0

10개 펼치기만 했는데 미묘하게 그래도 정확도가 좋아짐.

Deep 또한 마찬가지로

- Deep NN for XOR

```

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)

```

4 layers	2 layers
Hypothesis:	Hypothesis:
[[7.80e-04]	[[0.01338218]
[9.99e-01]	[0.98166394]
[9.98e-01]	[0.98809403]
[1.55e-03]]	[0.01135799]]
Correct:	Correct:
[[0.]	[[0.]
[1.]	[1.]
[1.]	[1.]
[0.]]	[0.]]
Accuracy: 1.0	Accuracy: 1.0

네트워크를 어떠한 모양으로 쌓느냐에 따라 달라지는 것이 딥러닝의 핵심!!

=====

12-02 Lab_딥러닝 (2)

Tensorboard 사용법을 실습해 본다.

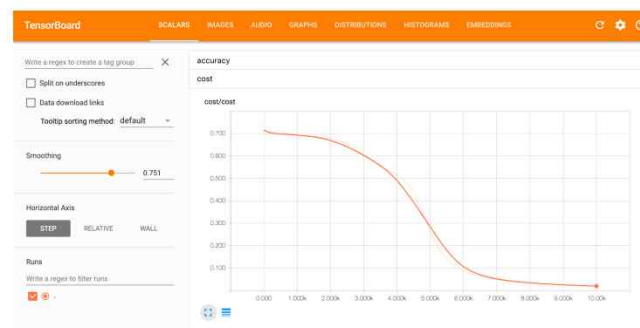
- Old fashion : print, print, print => 가시성이 너무 안좋다. tensorboard를 이용하자

- TensorBoard: TF logging/debugging tool

- ✓Visualize your TF graph

- ✓Plot quantitative metrics

- ✓Show additional data



- 5 steps of using TensorBoard

5가지 방법으로 진행된다. ctrl c,v 말고 슬래쉬 주의하자!

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("C:/MachineLearning/logs")
```

```
writer.add_graph(sess.graph)
```

4. Run summary merge and add_summary

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

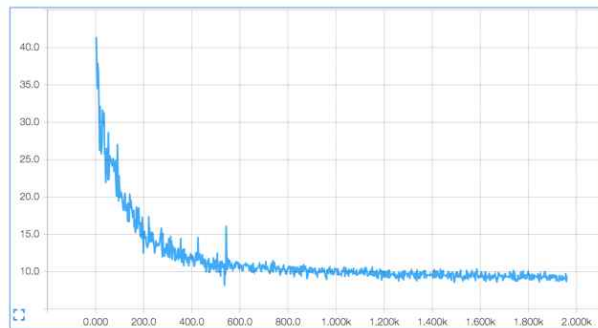
```
writer.add_summary(s, global_step=global_step)
```

5. Launch TensorBoard

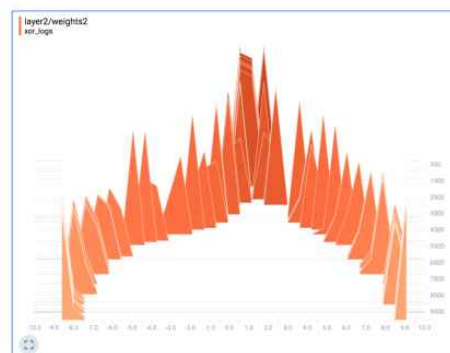
```
tensorboard --logdir=C:/MachineLearning/logs
```

- Scalar tensors

```
cost_summ = tf.summary.scalar("cost", cost)
```



- Histogram (multi-dimensional tensors)



- Add scope for better graph hierarchy

- Multiple runs

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

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

```
...
```

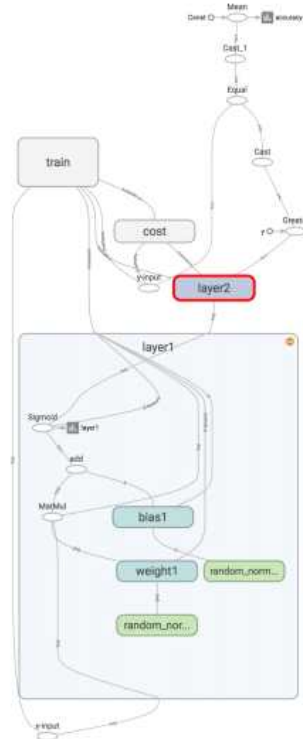
```
writer = tf.summary.FileWriter("./logs/xor_logs")
```

```
tensorboard --logdir=./logs/xor_logs_r0_01
```

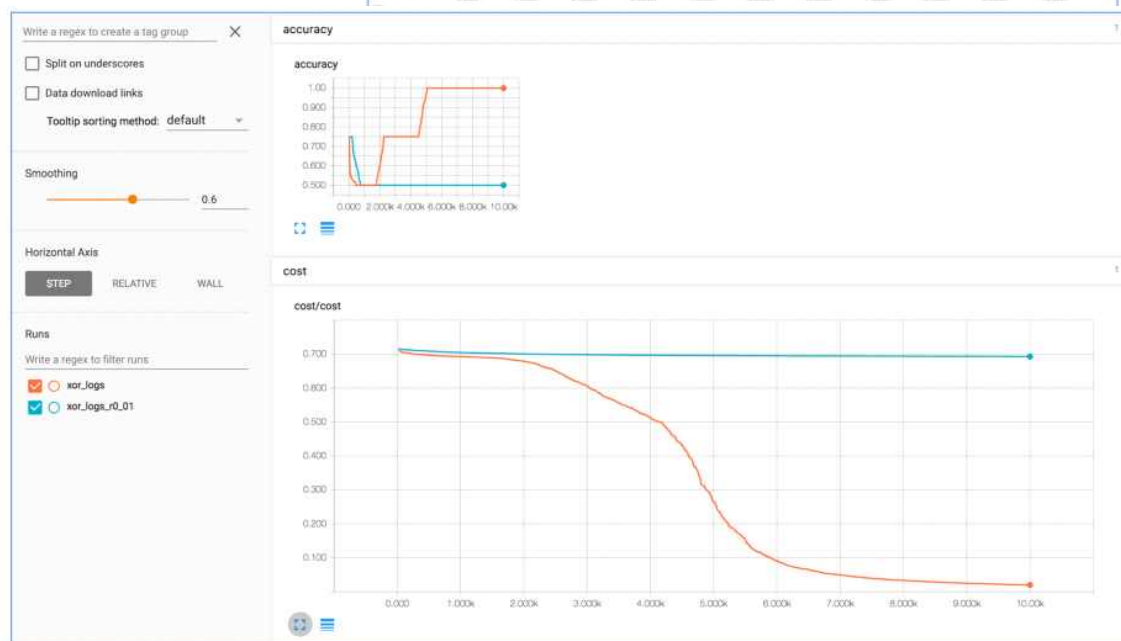
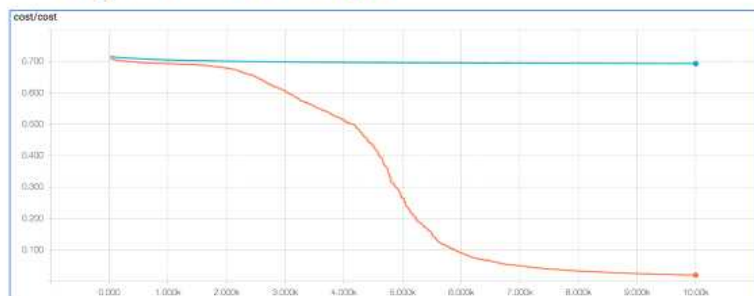
```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

```
...
```

```
writer = tf.summary.FileWriter("./logs/xor_logs_r0_01")
```

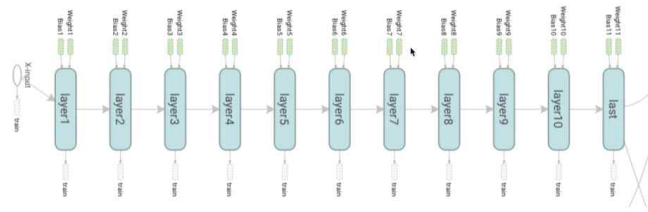


tensorboard --logdir=./logs

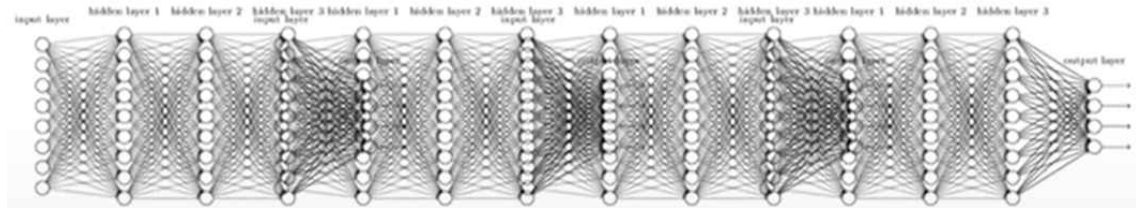


ReLU

Tensorboard visualization

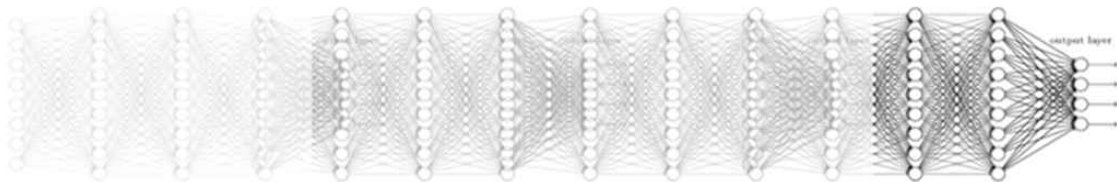


layer에 구성된 부분이 답하게 ㄱㄱ



back propagation이 등장하면서 암흑기에서 벗어날 수 있었다.

Vanishing gradient (NN winter2: 1986-2006) => 20년동안 또 암흑기 시작
경사 기울기가 사라진다.



Geoffrey Hinton's summary of findings 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.

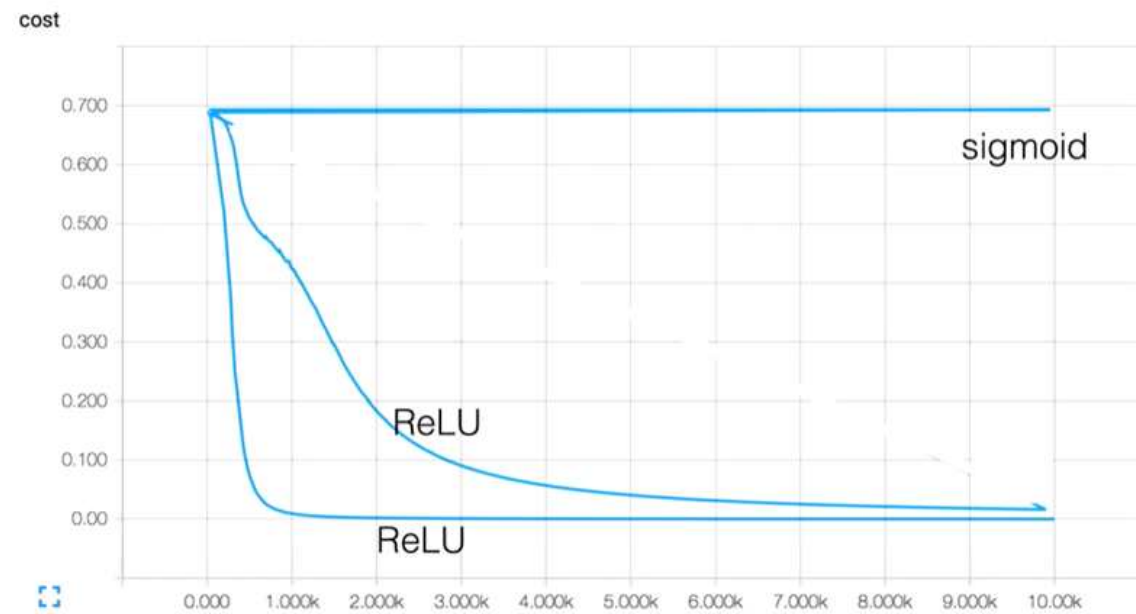


곱하고곱하고 곱하면서 vanishing gradient 현상이 발생하게된다.

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

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

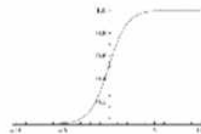
결국 시그모이드에서 렐루 함수로 바뀌게 됨. 그러나 맨 마지막에는 시그모이드를 사용해야한다!



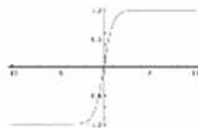
activation function

Sigmoid

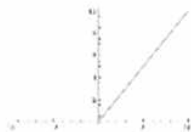
$$\sigma(x) = 1/(1 + e^{-x})$$



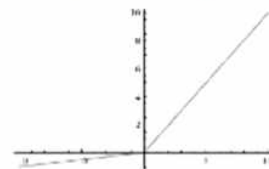
tanh $\tanh(x)$



ReLU $\max(0, x)$



Leaky ReLU
 $\max(0.1x, x)$



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

