시즌 1 - 딥러닝의 기본 - Lecture 10

노트북:	모두를 위한 머신러닝		
만든 날짜:	2019-01-09 오후 2:13	**수정한 날짜:**	2019-01-09 오후 4:14
작성자:	rr		
태그:	#모두를 위한, .Lecture		

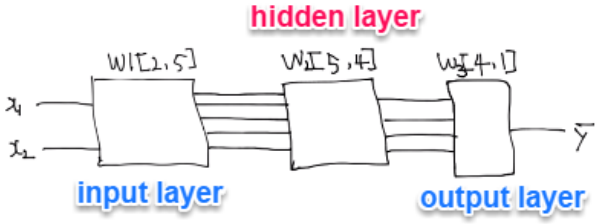
Lecture 10

= deep & wide NN

```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)
hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```



- poor results

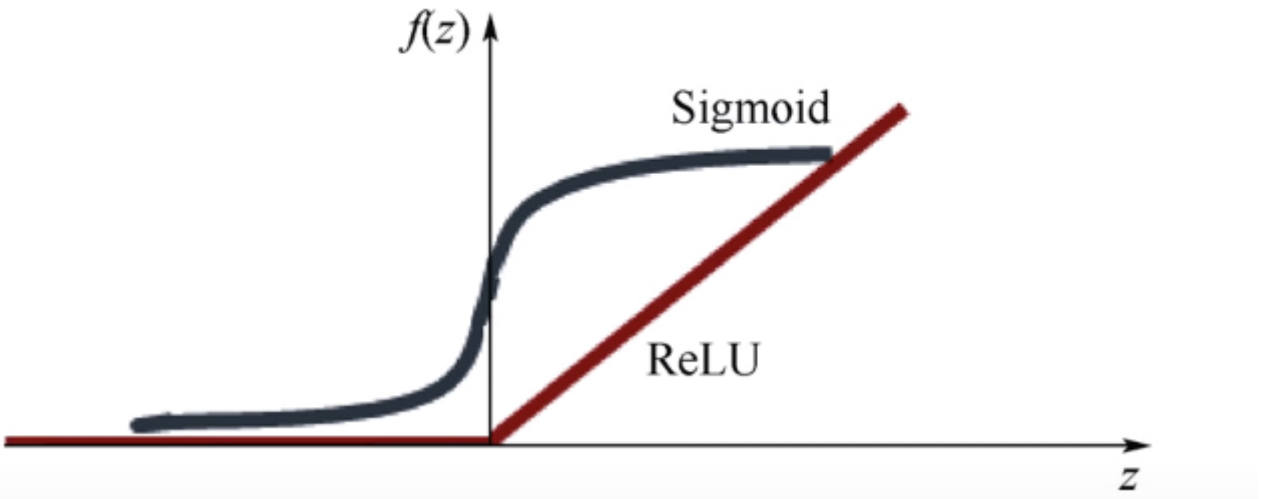
hidden layer를 9개 만들어서 실행시켰는데 Accuracy가 0.5로 나옴

-> backpropagation의 문제!
적은 레이어에서는 잘 동작하지만 9~10 이상의 레이어에서 잘 동작하지 않음
chain rule을 적용하기 때문에 1보다 작은 (0.01, 0.03, ...) 값이 계속 곱해지게 됨
점점 더 작아지게 되고 최종적인 결과에서 0에 가까운 값이 됨

-> Vanishing gradient(기울기가 사라지는 문제)가 발생함
학습하기가 어려고, 입력 값이 최종 값에 영향이 없어지기 때문에 예측이 제대로 되지 않음

-> non-linearity
sigmoid를 잘못 써서 이런 문제를 발생시킨 것 같다.

-> sigmoid 대신에 **ReLU**: Rectified Linear Unit 사용하자!



max(0, x)로 간단하게 구현 가능

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

마지막은 0~1 사이의 값이어야 하기 때문에 sigmoid를 사용함

```
with tf.name_scope("Layer1") as scope:
    L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
with tf.name_scope("Layer2") as scope:
    L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
...
hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
```

Accuracy가 1.0로 나옴

- 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

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

- sigmoid

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

- tanh

tanh(x)

= Weight 초기화

- Xavier/He initialization

input(fan_in), output(fan_out)이 몇 개인가에 따라서 랜덤하게 줌

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

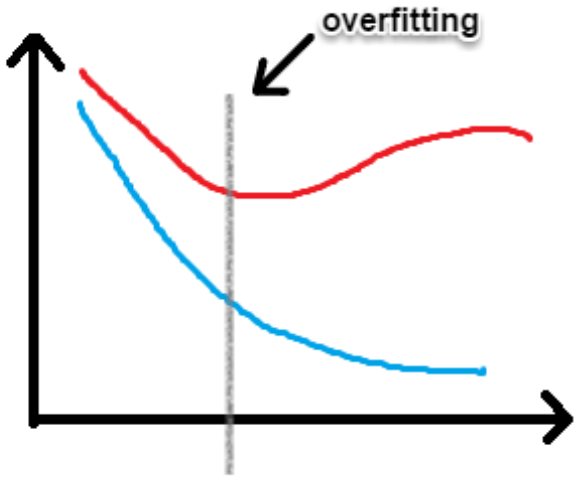
```
# He initialization
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

= Dropout과 앙상블

- overfitting 확인

training data set 에서는 acc: 0.99
test data set에서는 acc: 0.85

layer가 많을 수록 overfitting 문제가 더 심해짐



- overfitting 방지

-training data 더 많이 사용하기
-**Regularization**

- Regularization

각각 엘리먼트에 있는 값을 곱한 것이 최소화가 되게 하는 것
(lec 07 참고)

- **Dropout**

Neural Networks 에서 overfitting 방지 방법

복잡하게 만든 것을 몇 개 줄이자, 어떤 노드들을 버리자고 하는 것
랜덤하게 쉬게 해서 훈련을 시키는 것
실전에서는 총 동원해서 예측해야 함

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

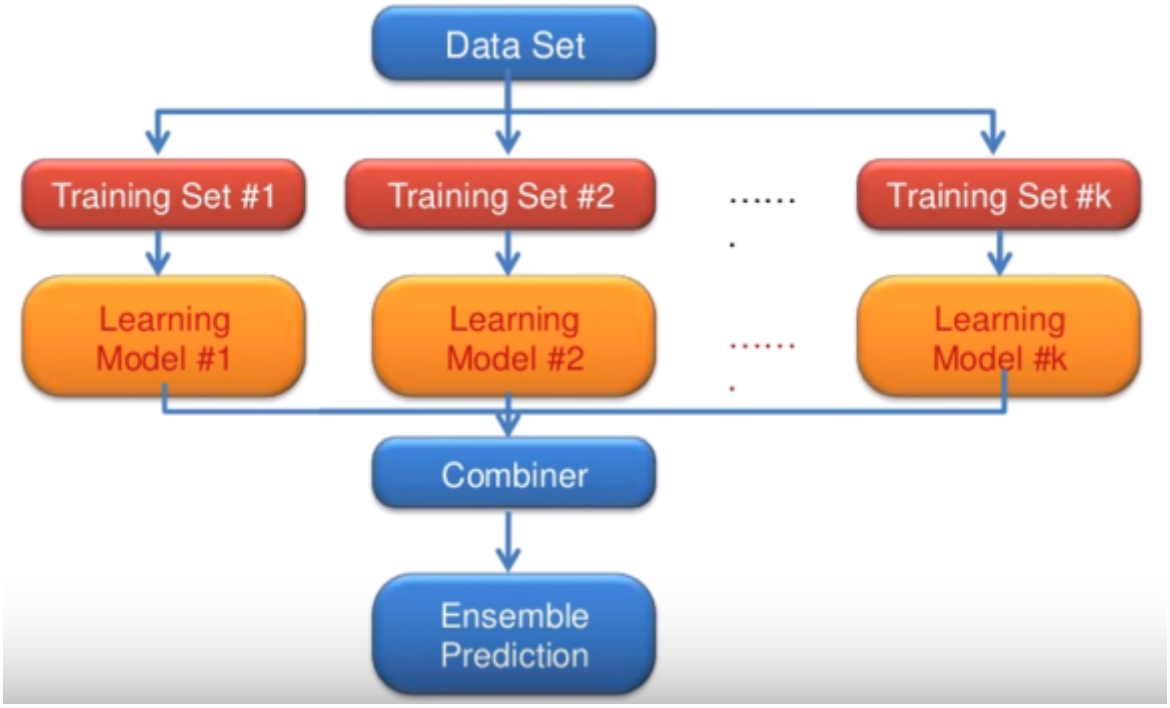
↘ 랜덤하게, 보통 0.5 (반을 잘라냄)

```
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})
```

잘라내는 건 train 동안만 함
실전에는 총 동원해서 모두 참여해야 함, 그렇기 때문에 evaluation에서 dropout_rate: 1

- **Ensemble**



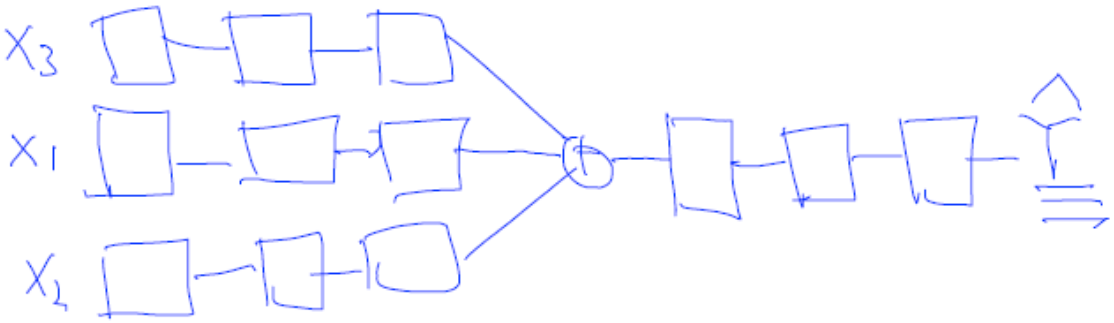
= NN LEGO PLAY

- Feedforward NN

몇 단씩 쌓은 구조

- Fast forward

- split & merge



- Recurrent network - RNN