**데이터 마이닝 Assignment #2**

**20121555 강승현**

**Exercise 1**

**1.**

**코드:**

with tf.variable\_scope('layer1') as scope: //hidden layer1.

W1 = tf.get\_variable("W", shape = [784,512], initializer = tf.contrib.layers.xavier\_initializer())

//weight [784,512]의 size를 가지고 초기화를 시켜준다.

b1 = tf.Variable(tf.random\_normal([512]))

//bias : 덧셈을 수행 해야 하므로 [512]의 size를 가진다.(덧셈에서 부족한 size는 큰 배열에 맞추어 자동으로 계산된다,)

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

//결과값 X\*W + b 가 된다.

tf.summary.histogram("X",X)

tf.summary.histogram("weights", W1)

tf.summary.histogram("bias", b1)

tf.summary.histogram("layer",L1)

with tf.variable\_scope('layer3') as scope: //hidden layer2.

W3 = tf.get\_variable("W", shape = [512,512], initializer = tf.contrib.layers.xavier\_initializer())

// L1의 size가 512이므로 두번째 layer의 weight size는 [512,512]가 된다.

b3 = tf.Variable(tf.random\_normal([512]))

//bias : 덧셈을 수행 해야 하므로 [512]의 size를 가진다.(덧셈에서 부족한 size는 큰 배열에 맞추어 자동으로 계산된다,)

L3 = tf.nn.relu(tf.add(tf.matmul(L1, W3),b3))

//L1\*W3+b3 가 된다.

tf.summary.histogram("weights", W3)

tf.summary.histogram("bias", b3)

tf.summary.histogram("layer",L3)

with tf.variable\_scope('layer2') as scope: //last layer

W2 = tf.get\_variable("W", shape=[512, 10], initializer = tf.contrib.layers.xavier\_initializer())

//output은 10개의 label로 표현되어야 하므로 size는 [512,10]이다.

b2 = tf.Variable(tf.random\_normal([10]))

//bias 역시 size는 10이다.

y\_ = tf.add(tf.matmul(L3, W2), b2)

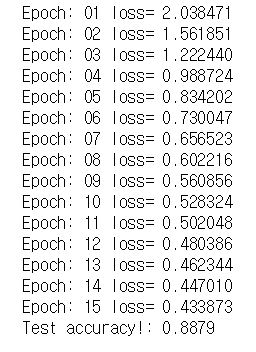
//L3\*W2 +b2를 계산하여 output을 산출한다.

tf.summary.histogram("weights", W2)

tf.summary.histogram("bias", b2)

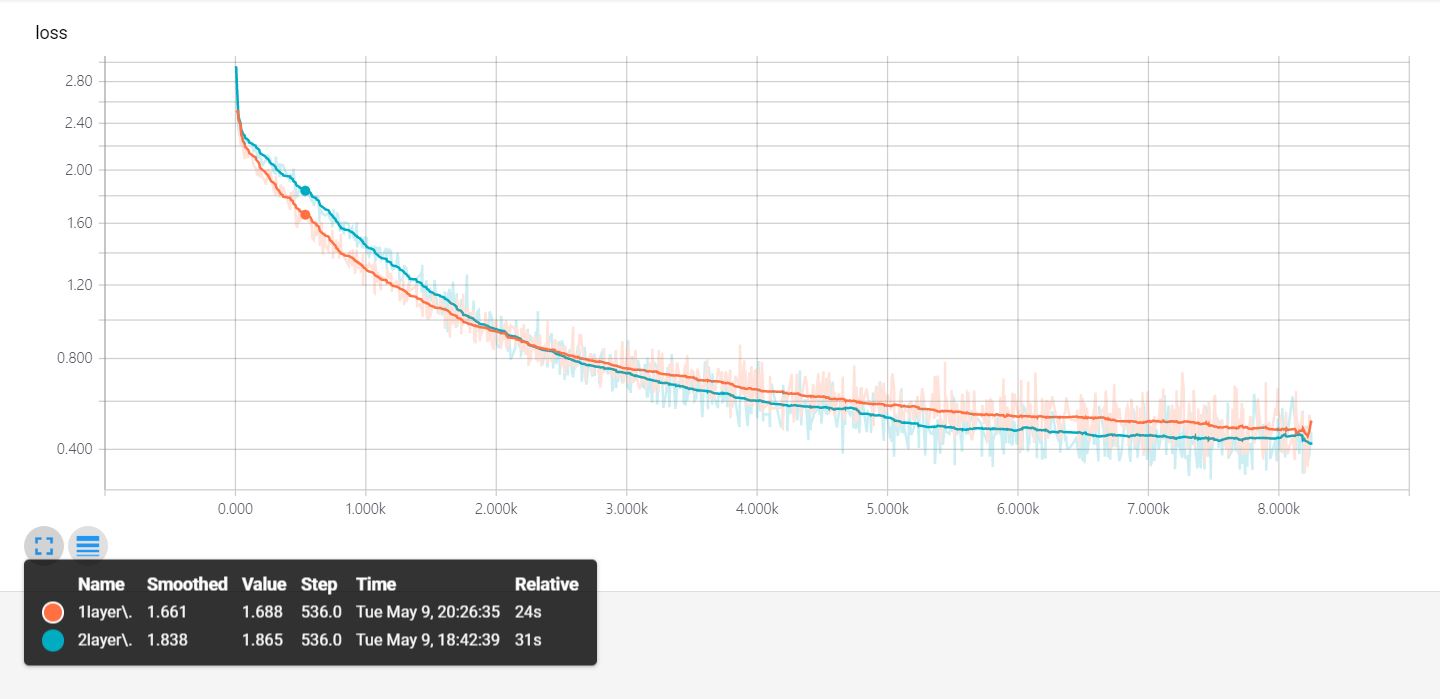
tf.summary.histogram("logits", y\_)

**결과:**

****

Hidden layer가 1개일 때는 accuracy가 0.7~0.8초반대가 나왔으므로 하나의 layer을 더 추가했을 때 약간 더 성능이 좋아진 것을 알 수 있다. 하지만 큰 차이는 없었다.

**2.**

****

**Exercise 2**

**1.**

**SGD** : parameter set을 error function이 가장 작아지도록 하는 것 .

Train set에서 하나의 sample만을 반복적으로 업데이트한다. 즉, 매 step마다 subset을뽑아서 사용하는 것. Sample이 커져도 하나의 sample을 사용하기 때문에 연산속도를 빠르게 할 수 있다. Gradient descent 의 approximation.

**AdamOptimizer**: AdamOptimizer는 parameters의 moving averages를 사용하는 방법이며 메모리 요구사항이 거의 없으며 경사도의 diagonal rescaling이불변하며 data의 사이즈가 클 때 굉장히 효과적이다. (노이즈가 크거나 경사도가 높은 상황에 적합) 최근 가장 많이 쓰인다.

**2.**

**Sigmoid function**: 표본들이 위치한 공식을 만들어 내는 것이 cost function인데, 이를 찾는 과정에서 Gradient Decent방법이 사용된다. 하지만 이 방법은 그래프의 모든 지점에서 미분 가능해야 한다. Sigmoid 함수는 미분이 되지 않는 곳에서 사용할 수 있는 방법이다. 계단형식의 함수를 미분 가능 하도록 곡선화 해주는 것이다.

**ReLu function** : sigmoid 함수에는 Gradient Vanisihing 문제가 있다. Simgoid function은 0에서 1사이의 값을 가지는데 gradient descent를 사용해 Backpropagation 수행시 layer를 지나면서 gradient를 계속 곱하므로 gradient는 0으로 수렴하게 된다. 따라서 layer가 많아지면 잘 작동하지 않게 되는 것이다. 이를 해결하기 위해 쓰이는 함수이며 입력값이 0보다 작으면 0이고 0보다 크면 입력값을 그대로 내보냄으로써 해결한다. Sparse activation, Efficient Gradient propagation, Efficient computation, scale invariant 등의 이점이 있다.

**3.**



다음과 같이 get\_train\_batch 함수를 작성하였습니다.