Lab 10

NN, ReLu, Xavier, Dropout, and Adam

Sung Kim <hunkim+ml@gmail.com>
http://hunkim.github.io/ml/

Examples

- https://github.com/aymericdamien/TensorFlow-Examples

Softmax classifier for MNIST

```python
# tf Graph Input
x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

# Create model

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# Construct model
activation = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(activation), reduction_indices=1)) # Cross entropy
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) # Gradient Descent

# Initializing the variables
init = tf.initialize_all_variables()

# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
            print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost)

    print "Optimization Finished!"

    # Test model
    correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))
    # Calculate accuracy
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    print "Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels})
```

softmax classifier

```
Epoch: 0001 cost= 1.174406660
Epoch: 0002 cost= 0.661967539
Epoch: 0003 cost= 0.550489192
Epoch: 0004 cost= 0.496657414
Epoch: 0005 cost= 0.463665792
Epoch: 0006 cost= 0.440912077
Epoch: 0007 cost= 0.423909424
Epoch: 0008 cost= 0.410630655
Epoch: 0009 cost= 0.399893884
Epoch: 0010 cost= 0.390907963
Epoch: 0011 cost= 0.383317497
Epoch: 0012 cost= 0.376792131
Epoch: 0013 cost= 0.371025368
Epoch: 0014 cost= 0.365951805
Epoch: 0015 cost= 0.361361689
Epoch: 0016 cost= 0.357238019
Epoch: 0017 cost= 0.353540161
Epoch: 0018 cost= 0.350144092
Epoch: 0019 cost= 0.347053342
Epoch: 0020 cost= 0.344076798
Epoch: 0021 cost= 0.341447881
Epoch: 0022 cost= 0.339008725
Epoch: 0023 cost= 0.336701365
Epoch: 0024 cost= 0.334450486
Epoch: 0025 cost= 0.332461696
Optimization Finished!
```

Accuracy: 0.9139

Neural Nets (NN) for MNIST

```
# Parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
display_step = 1

# tf Graph input
X = tf.placeholder("float", [None, 784]) # MNIST data input (img shape: 28*28)
Y = tf.placeholder("float", [None, 10])  # MNIST total classes (0-9 digits)

# Store layers weight & bias
W1 = tf.Variable(tf.random_normal([784, 256]))
W2 = tf.Variable(tf.random_normal([256, 256]))
W3 = tf.Variable(tf.random_normal([256, 10]))

B1 = tf.Variable(tf.random_normal([256]))
B2 = tf.Variable(tf.random_normal([256]))
B3 = tf.Variable(tf.random_normal([10]))

# Construct model
L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
L2 = tf.nn.relu(tf.add(tf.matmul(L1, W2), B2)) #Hidden layer with RELU activation
hypothesis = tf.add(tf.matmul(L2, W3), B3) # No need to use softmax here

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(hypothesis, Y)) # Softmax loss
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost) # Adam Optimizer
```

Neural Nets (NN) for MNIST

```
# Initializing the variables
init = tf.initialize_all_variables()

# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={X: batch_xs, Y: batch_ys})/total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
            print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost)

    print "Optimization Finished!"

    # Test model
    correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
    # Calculate accuracy
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    print "Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels})
```

softmax classifier

Epoch: 0001 cost= 1.174406660
Epoch: 0002 cost= 0.661967539
Epoch: 0003 cost= 0.550489192
Epoch: 0004 cost= 0.496657414
Epoch: 0005 cost= 0.463665792
Epoch: 0006 cost= 0.440912077
Epoch: 0007 cost= 0.423909424
Epoch: 0008 cost= 0.410630655
Epoch: 0009 cost= 0.399893884
Epoch: 0010 cost= 0.390907963
Epoch: 0011 cost= 0.383317497
Epoch: 0012 cost= 0.376792131
Epoch: 0013 cost= 0.371025368
Epoch: 0014 cost= 0.365951805
Epoch: 0015 cost= 0.361361689
Epoch: 0016 cost= 0.357238019
Epoch: 0017 cost= 0.353540161
Epoch: 0018 cost= 0.350144092
Epoch: 0019 cost= 0.347053342
Epoch: 0020 cost= 0.344076798
Epoch: 0021 cost= 0.341447881
Epoch: 0022 cost= 0.339008725
Epoch: 0023 cost= 0.336701365
Epoch: 0024 cost= 0.334450486
Epoch: 0025 cost= 0.332461696
Optimization Finished!
Accuracy: 0.9139

NN

Epoch: 0001 cost= 153.374492868
Epoch: 0002 cost= 41.126819546
Epoch: 0003 cost= 25.309642092
Epoch: 0004 cost= 17.206465834
Epoch: 0005 cost= 12.155490249
Epoch: 0006 cost= 8.755095852
Epoch: 0007 cost= 6.392030562
Epoch: 0008 cost= 4.629136964
Epoch: 0009 cost= 3.347306573
Epoch: 0010 cost= 2.372126589
Epoch: 0011 cost= 1.667233310
Epoch: 0012 cost= 1.202339336
Epoch: 0013 cost= 0.837206638
Epoch: 0014 cost= 0.593220934
Epoch: 0015 cost= 0.431912481
Optimization Finished!
Accuracy: 0.9446

Xavier initialization

```python
def xavier_init(n_inputs, n_outputs, uniform=True):
    """Set the parameter initialization using the method described.
    This method is designed to keep the scale of the gradients roughly the same
    in all layers.
    Xavier Glorot and Yoshua Bengio (2010):
             Understanding the difficulty of training deep feedforward neural
             networks. International conference on artificial intelligence and
             statistics.
    Args:
      n_inputs: The number of input nodes into each output.
      n_outputs: The number of output nodes for each input.
      uniform: If true use a uniform distribution, otherwise use a normal.
    Returns:
      An initializer.
    """
    if uniform:
      # 6 was used in the paper.
      init_range = tf.sqrt(6.0 / (n_inputs + n_outputs))
      return tf.random_uniform_initializer(-init_range, init_range)
    else:
      # 3 gives us approximately the same limits as above since this repicks
      # values greater than 2 standard deviations from the mean.
      stddev = tf.sqrt(3.0 / (n_inputs + n_outputs))
      return tf.truncated_normal_initializer(stddev=stddev)
```

```python
# Store layers weight & bias
W1 = tf.get_variable("W1", shape=[784, 256], initializer=xavier_init(784,256))
W2 = tf.get_variable("W2", shape=[256, 256], initializer=xavier_init(256,256))
W3 = tf.get_variable("W3", shape=[256, 10], initializer=xavier_init(256,10))
```

http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow

NN

Epoch: 0001 cost= 153.374492868
Epoch: 0002 cost= 41.126819546
Epoch: 0003 cost= 25.309642092
Epoch: 0004 cost= 17.206465834
Epoch: 0005 cost= 12.155490249
Epoch: 0006 cost= 8.755095852
Epoch: 0007 cost= 6.392030562
Epoch: 0008 cost= 4.629136964
Epoch: 0009 cost= 3.347306573
Epoch: 0010 cost= 2.372126589
Epoch: 0011 cost= 1.667233310
Epoch: 0012 cost= 1.202339336
Epoch: 0013 cost= 0.837206638
Epoch: 0014 cost= 0.593220934
Epoch: 0015 cost= 0.431912481
Optimization Finished!
Accuracy: 0.9446

NN with xavier initialization

Epoch: 0001 cost= 0.330929694
Epoch: 0002 cost= 0.110038888
Epoch: 0003 cost= 0.067369296
Epoch: 0004 cost= 0.045064388
Epoch: 0005 cost= 0.031090851
Epoch: 0006 cost= 0.022001974
Epoch: 0007 cost= 0.016603567
Epoch: 0008 cost= 0.011094349
Epoch: 0009 cost= 0.008923969
Epoch: 0010 cost= 0.007312808
Epoch: 0011 cost= 0.006277084
Epoch: 0012 cost= 0.004857574
Epoch: 0013 cost= 0.004891470
Epoch: 0014 cost= 0.004491583
Epoch: 0015 cost= 0.003429245
Optimization Finished!
Accuracy: 0.9779

More deep & dropout

```python
# Construct model
dropout_rate = tf.placeholder("float")
_L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1)) #Hidden layer with RELU activation
L1 = tf.nn.dropout(_L1, dropout_rate)
_L2 = tf.nn.relu(tf.add(tf.matmul(L1, W2), B2)) #Hidden layer with RELU activation
L2 = tf.nn.dropout(_L2, dropout_rate)
_L3 = tf.nn.relu(tf.add(tf.matmul(L2, W3), B3)) #Hidden layer with RELU activation
L3 = tf.nn.dropout(_L3, dropout_rate)
_L4 = tf.nn.relu(tf.add(tf.matmul(L3, W4), B4)) #Hidden layer with RELU activation
L4 = tf.nn.dropout(_L4, dropout_rate)

hypothesis = tf.add(tf.matmul(L4, W5), B5) # No need to use softmax here
```

```python
for i in range(total_batch):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)
    # Fit training using batch data
    sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys, dropout_rate: 0.7})
```

```python
# Test model
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print "Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels, dropout_rate: 1})
```

NN deep + dropout

Epoch: 0001 cost= 0.584449715
Epoch: 0002 cost= 0.215399251
Epoch: 0003 cost= 0.160561109
Epoch: 0004 cost= 0.132314345
Epoch: 0005 cost= 0.114490116
Epoch: 0006 cost= 0.103506013
Epoch: 0007 cost= 0.095571726
Epoch: 0008 cost= 0.084172901
Epoch: 0009 cost= 0.079563179
Epoch: 0010 cost= 0.073859323
Epoch: 0011 cost= 0.071492671
Epoch: 0012 cost= 0.066446339
Epoch: 0013 cost= 0.061337474
Epoch: 0014 cost= 0.058591939
Epoch: 0015 cost= 0.055077895
Optimization Finished!
Accuracy: 0.9803

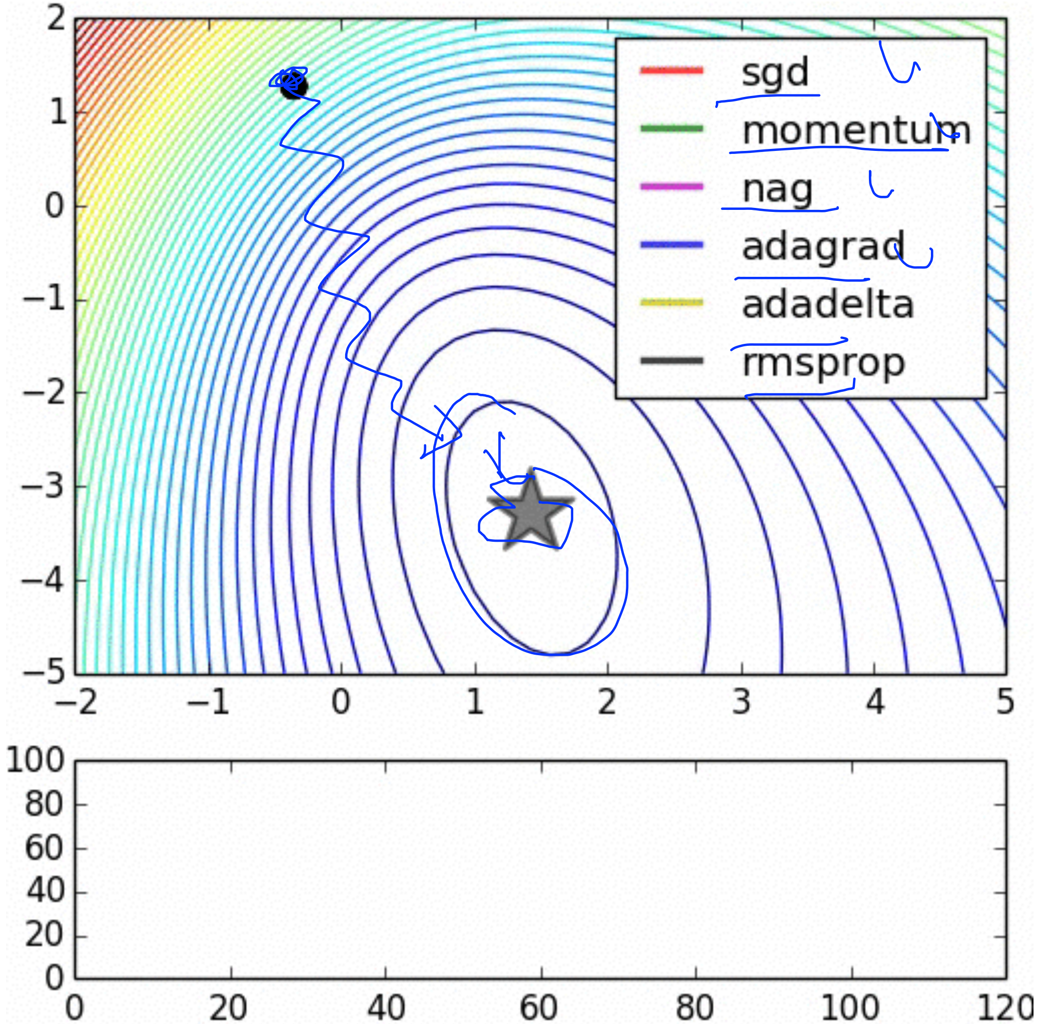
NN with xavier initialization

Epoch: 0001 cost= 0.330929694
Epoch: 0002 cost= 0.110038888
Epoch: 0003 cost= 0.067369296
Epoch: 0004 cost= 0.045064388
Epoch: 0005 cost= 0.031090851
Epoch: 0006 cost= 0.022001974
Epoch: 0007 cost= 0.016603567
Epoch: 0008 cost= 0.011094349
Epoch: 0009 cost= 0.008923969
Epoch: 0010 cost= 0.007312808
Epoch: 0011 cost= 0.006277084
Epoch: 0012 cost= 0.004857574
Epoch: 0013 cost= 0.004891470
Epoch: 0014 cost= 0.004491583
Epoch: 0015 cost= 0.003429245
Optimization Finished!
Accuracy: 0.9779

Optimizer

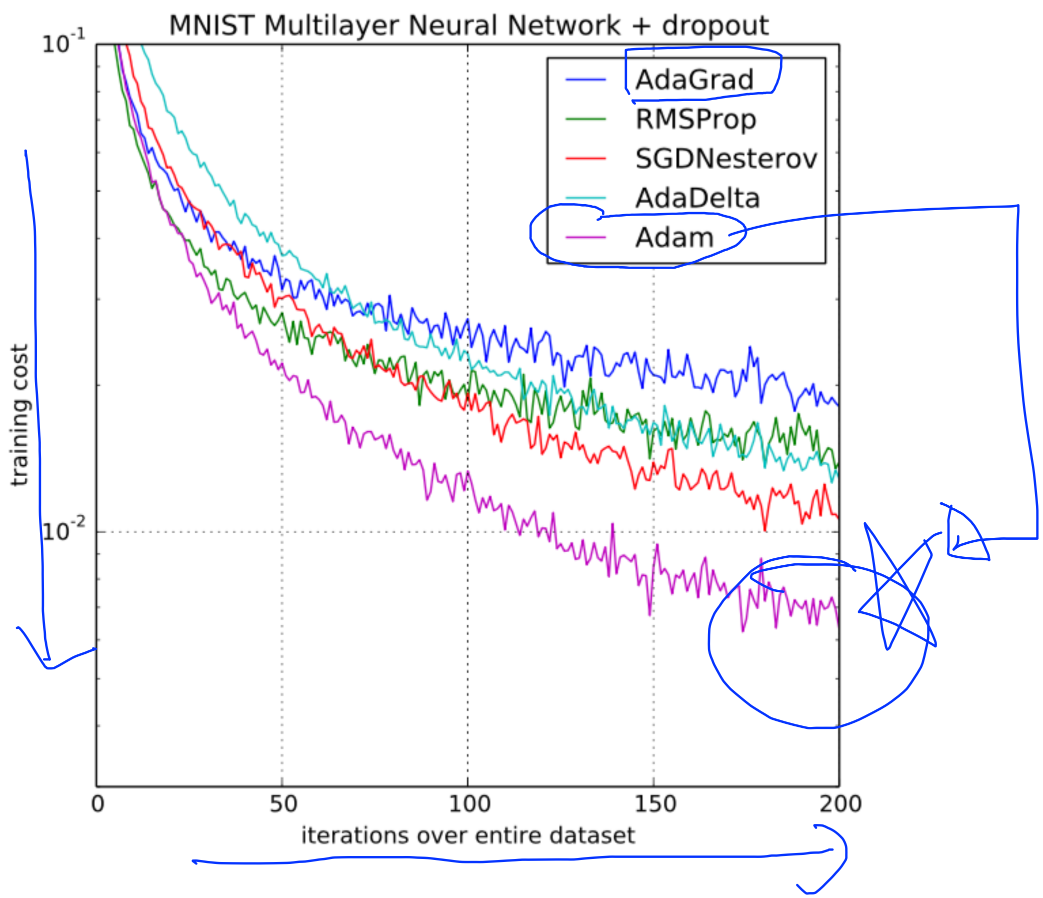
```
# Construct model
activation = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(activation), reduction_indices=1)) # Cross entropy
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) # Gradient Descent
```



http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

ADAM: a method for stochastic optimization
[Kingma et al. 2015]



Use Adam Optimizer

```
# Construct model
activation = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(activation), reduction_indices=1)) # Cross entropy
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) # Gradient Descent
```

```
hypothesis = tf.add(tf.matmul(L4, W5), B5) # No need to use softmax here

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(hypothesis, Y)) # Softmax loss
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost) # Adam Optimizer
```

Lab summary

- Softmax VS Neural Nets for MNIST, 91.4% and 94.4%
- Xavier initialization: 97.8%
- Deep Neural Nets and Dropout: 98%
- Adam optimizer