

With TF 1.0!



Lab 10

NN, ReLu, Xavier, Dropout, and Adam

Sung Kim <hunkim+ml@gmail.com>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



Call for comments

Please feel free to add comments directly on these slides

Other slides: <https://goo.gl/jPtVNT>



With TF 1.0!



Lab 10

NN, ReLu, Xavier, Dropout, and Adam

Sung Kim <hunkim+ml@gmail.com>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<https://github.com/hunkim/DeepLearningZeroToAll/>



zeran4

1 commit / 5 ++ / 4 --

#11



jennykang

19 commits / 940 ++ / 253 --

#2



GzuPark

14 commits / 41 ++ / 31 --

#3



kkweon

5 commits / 372 ++ / 296 --

#4



BlueMelon715

4 commits / 45 ++ / 34 --

#5



jin-chong

2 commits / 4 ++ / 4 --

#6



FuZer

2 commits / 37 ++ / 30 --

#7



cynthia

1 commit / 28 ++ / 28 --

#8



keon

1 commit / 3 ++ / 3 --

#9



allieus

1 commit / 55 ++ / 59 --

#10

Softmax classifier for MNIST

```
# 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=hypothesis, 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}))
```

```
Epoch: 0001 cost = 5.888845987
Epoch: 0002 cost = 1.860620173
Epoch: 0003 cost = 1.159035648
Epoch: 0004 cost = 0.892340870
Epoch: 0005 cost = 0.751155428
Epoch: 0006 cost = 0.662484806
Epoch: 0007 cost = 0.601544010
Epoch: 0008 cost = 0.556526115
Epoch: 0009 cost = 0.521186961
Epoch: 0010 cost = 0.493068354
Epoch: 0011 cost = 0.469686249
Epoch: 0012 cost = 0.449967254
Epoch: 0013 cost = 0.433519321
Epoch: 0014 cost = 0.419000337
Epoch: 0015 cost = 0.406490815
Learning Finished!
Accuracy: 0.9035
```

With TF 1.0!

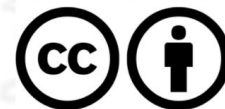


Lab 7-2

MNIST data

Sung Kim <hunkim+ml@gmail.com>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



Softmax classifier for MNIST

```
# 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=hypothesis, 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}))
```

```
Epoch: 0001 cost = 5.888845987
Epoch: 0002 cost = 1.860620173
Epoch: 0003 cost = 1.159035648
Epoch: 0004 cost = 0.892340870
Epoch: 0005 cost = 0.751155428
Epoch: 0006 cost = 0.662484806
Epoch: 0007 cost = 0.601544010
Epoch: 0008 cost = 0.556526115
Epoch: 0009 cost = 0.521186961
Epoch: 0010 cost = 0.493068354
Epoch: 0011 cost = 0.469686249
Epoch: 0012 cost = 0.449967254
Epoch: 0013 cost = 0.433519321
Epoch: 0014 cost = 0.419000337
Epoch: 0015 cost = 0.406490815
Learning Finished!
Accuracy: 0.9035
```

NN for MNIST

```
# 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)
```

```
Epoch: 0001 cost = 141.207671860
Epoch: 0002 cost = 38.788445864
Epoch: 0003 cost = 23.977515479
Epoch: 0004 cost = 16.315132428
Epoch: 0005 cost = 11.702554882
Epoch: 0006 cost = 8.573139748
Epoch: 0007 cost = 6.370995680
Epoch: 0008 cost = 4.537178684
Epoch: 0009 cost = 3.216900532
Epoch: 0010 cost = 2.329708954
Epoch: 0011 cost = 1.715552875
Epoch: 0012 cost = 1.189857912
Epoch: 0013 cost = 0.820965160
Epoch: 0014 cost = 0.624131458
Epoch: 0015 cost = 0.454633765
Learning Finished!
Accuracy: 0.9455
```




All

Videos

News

Images

Maps

More

Settings

Tools

About 2,000 results (0.43 seconds)

python - How to do Xavier initialization on TensorFlow - Stack Overflow

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

Nov 10, 2015 - I'm porting my Caffe network over to **TensorFlow** but it doesn't seem to ... Does **Xavier initialization** changes how the biases are initialized ...

You've visited this page many times. Last visit: 1/26/17

Tags

Users

on TensorFlow

to TensorFlow but it doesn't seem to have xavier initialization. I'm seems to be making it a lot harder to train.

edited Dec 19 '15 at 22:25



Hooked

32.2k ● 13 ● 108 ● 169

asked Nov 10 '15 at 22:07



Aleph7

2,646 ● 1 ● 14 ● 24

Does Xavier initialization changes how the biases are initialized? – Pinocchio Jul 25 '16 at 20:07

[add a comment](#)[start a bounty](#)

4 Answers

active

oldest

votes



Since version 0.8 there is a Xavier initializer, [see here for the docs](#).

55

You can use something like this:



```
W = tf.get_variable("W", shape=[784, 256],  
                    initializer=tf.contrib.layers.xavier_initializer())
```

[share](#) [edit](#) [delete](#) [flag](#)

edited Dec 1 '16 at 16:32



fabian789

5,296 ● 3 ● 32 ● 75

answered Apr 22 '16 at 4:23



Sung Kim

2,339 ● 1 ● 15 ● 24

Xavier for MNIST

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

# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581
W1 = tf.get_variable("W1", shape=[784, 256],
                    initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[256, 256],
                    initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[256, 10],
                    initializer=tf.contrib.layers.xavier_initializer())
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)
```

```
Epoch: 0001 cost = 0.301498963
Epoch: 0002 cost = 0.107252513
Epoch: 0003 cost = 0.064888892
Epoch: 0004 cost = 0.044463030
Epoch: 0005 cost = 0.029951642
Epoch: 0006 cost = 0.020663404
Epoch: 0007 cost = 0.015853033
Epoch: 0008 cost = 0.011764387
Epoch: 0009 cost = 0.008598264
Epoch: 0010 cost = 0.007383116
Epoch: 0011 cost = 0.006839140
Epoch: 0012 cost = 0.004672963
Epoch: 0013 cost = 0.003979437
Epoch: 0014 cost = 0.002714260
Epoch: 0015 cost = 0.004707661
Learning Finished!
```

Accuracy: **0.9783**

Xavier for MNIST

```
# input place holders
```

```
X = tf.placeholder(tf.float32, [None, 784])
```

```
Y = tf.placeholder(tf.float32, [None, 10])
```

```
# weights & bias for nn layers
```

```
# http://stackoverflow.com/questions/3364
```

```
W1 = tf.get_variable("W1", shape=[784, 256],
```

```
initializer=tf.contrib
```

```
b1 = tf.Variable(tf.random_normal([256]))
```

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

```
W2 = tf.get_variable("W2", shape=[256, 256],
```

```
initializer=tf.contrib
```

```
b2 = tf.Variable(tf.random_normal([256]))
```

```
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
```

```
W3 = tf.get_variable("W3", shape=[256, 10],
```

```
initializer=tf.contrib
```

```
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)
```

```
Epoch: 0001 cost = 141.207671860
```

```
Epoch: 0002 cost = 38.788445864
```

```
Epoch: 0003 cost = 23.977515479
```

```
Epoch: 0004 cost = 16.315132428)
```

```
Epoch: 0005 cost = 11.702554882
```

```
Epoch: 0006 cost = 8.573139748
```

```
Epoch: 0007 cost = 6.370995680
```

```
Epoch: 0008 cost = 4.537178684
```

```
Epoch: 0009 cost = 3.216900532())
```

```
Epoch: 0010 cost = 2.329708954
```

```
Epoch: 0011 cost = 1.715552875
```

```
Epoch: 0012 cost = 1.189857912
```

```
Epoch: 0013 cost = 0.820965160
```

```
Epoch: 0014 cost = 0.624131458())
```

```
Epoch: 0015 cost = 0.454633765
```

```
Learning Finished!
```

```
Accuracy: 0.9455 (normal dist)
```

```
Epoch: 0001 cost = 0.301498963
```

```
Epoch: 0002 cost = 0.107252513
```

```
Epoch: 0003 cost = 0.064888892
```

```
Epoch: 0004 cost = 0.044463030
```

```
Epoch: 0005 cost = 0.029951642
```

```
Epoch: 0006 cost = 0.020663404
```

```
Epoch: 0007 cost = 0.015853033
```

```
Epoch: 0008 cost = 0.011764387
```

```
Epoch: 0009 cost = 0.008598264
```

```
Epoch: 0010 cost = 0.007383116
```

```
Epoch: 0011 cost = 0.006839140
```

```
Epoch: 0012 cost = 0.004672963
```

```
Epoch: 0013 cost = 0.003979437
```

```
Epoch: 0014 cost = 0.002714260
```

```
Epoch: 0015 cost = 0.004707661
```

```
Learning Finished!
```

```
Accuracy: 0.9783 (xavier)
```

Deep NN for MNIST

```
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)
```

```
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)
```

```
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)
```

```
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)
```

```
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
```

```
# 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)
```

```
Epoch: 0001 cost = 0.266061549  
Epoch: 0002 cost = 0.080796588  
Epoch: 0003 cost = 0.049075800  
Epoch: 0004 cost = 0.034772298  
Epoch: 0005 cost = 0.024780529  
Epoch: 0006 cost = 0.017072763  
Epoch: 0007 cost = 0.014031383  
Epoch: 0008 cost = 0.013763446  
Epoch: 0009 cost = 0.009164047  
Epoch: 0010 cost = 0.008291388  
Epoch: 0011 cost = 0.007319742  
Epoch: 0012 cost = 0.006434021  
Epoch: 0013 cost = 0.005684378  
Epoch: 0014 cost = 0.004781207  
Epoch: 0015 cost = 0.004342310  
Learning Finished!
```

Accuracy: **0.9742**

```
# 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])
```

```
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])
```

```
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)
```

```
...
```

```
# train my model
```

```
for epoch in range(training_epochs):
```

```
    ...
```

```
    for i in range(total_batch):
```

```
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
```

```
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
```

```
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
```

```
        avg_cost += c / total_batch
```

```
# 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, keep_prob: 1}))
```

Dropout for MNIST

Epoch: 0001 cost = 0.447322626

Epoch: 0002 cost = 0.157285590

Epoch: 0003 cost = 0.121884535

Epoch: 0004 cost = 0.098128681

Epoch: 0005 cost = 0.082901778

Epoch: 0006 cost = 0.075337573

Epoch: 0007 cost = 0.069752543

Epoch: 0008 cost = 0.060884363

Epoch: 0009 cost = 0.055276413

Epoch: 0010 cost = 0.054631256

Epoch: 0011 cost = 0.049675195

Epoch: 0012 cost = 0.049125314

Epoch: 0013 cost = 0.047231930

Epoch: 0014 cost = 0.041290121

Epoch: 0015 cost = 0.043621063

Learning Finished!

Accuracy: **0.9804!!**

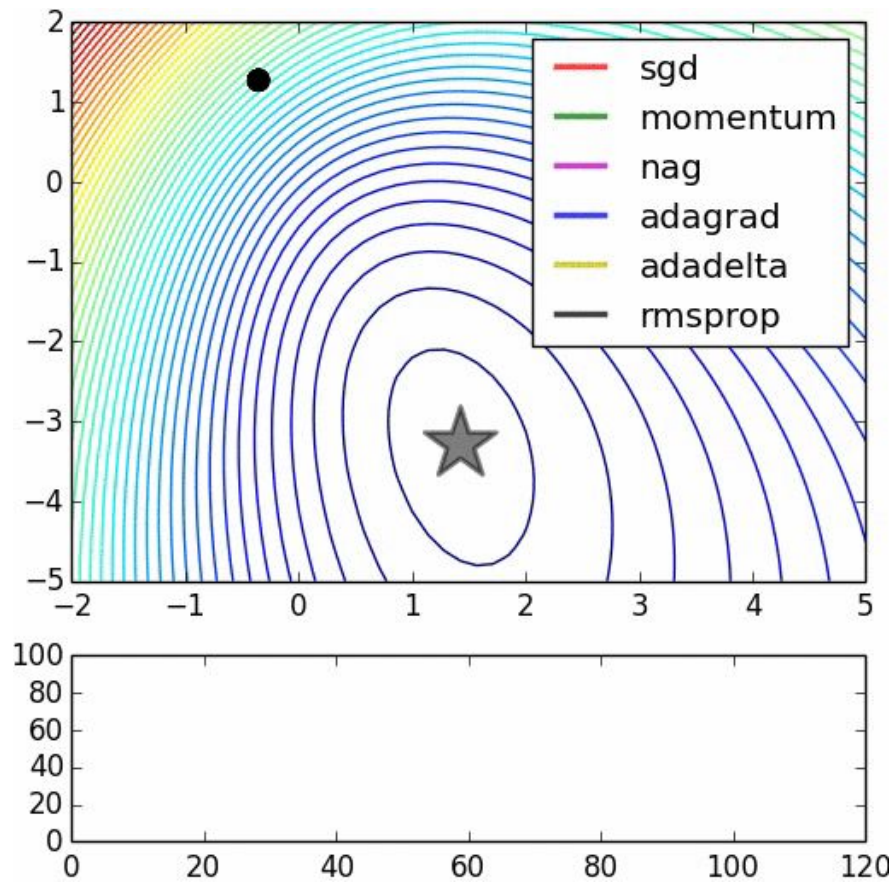
Optimizers

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

Optimizers

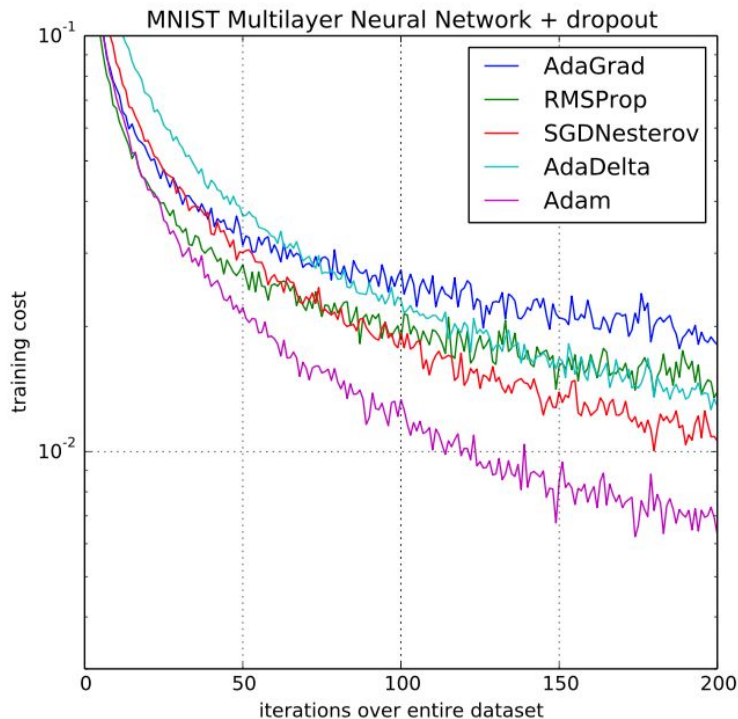
```
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

- `tf.train.AdadeltaOptimizer`
- `tf.train.AdagradOptimizer`
- `tf.train.AdagradDAOptimizer`
- `tf.train.MomentumOptimizer`
- `tf.train.AdamOptimizer`
- `tf.train.FtrlOptimizer`
- `tf.train.ProximalGradientDescentOptimizer`
- `tf.train.ProximalAdagradOptimizer`
- `tf.train.RMSPropOptimizer`



ADAM: a method for stochastic optimization

[Kingma et al. 2015]



Use Adam Optimizer

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)
```

Summary

- Softmax VS Neural Nets for MNIST, 90% and 94.5%
- Xavier initialization: 97.8%
- Deep Neural Nets with Dropout: **98%**
- Adam and other optimizers
- Exercise: Batch Normalization
 - https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-10-6-mnist_nn_batchnorm.ipynb

Lecture and Lab I I

CNN

