Hello World: handwritten digits classification - MNIST

MNIST = Mixed National Institute of Standards and Technology - Download the dataset at http://yann.lecun.com/exdb/mnist/
Very simple model: softmax classification

softmax\( (L_n) = \frac{e^{L_n}}{\|e^L\|} \)

neuron outputs

weighted sum of all pixels + bias

28x28 pixels

784 pixels
In matrix notation, 100 images at a time

\[
L = X \cdot W + b
\]

\(X\): 100 images, one per line, flattened

\(L\): 784 pixels

\(W\): 10 columns

\(b\): Same biases on all lines
Softmax, on a batch of images

\[ Y = \text{softmax}(X \cdot W + b) \]

- Predictions: \( Y[100, 10] \)
- Images: \( X[100, 748] \)
- Weights: \( W[748, 10] \)
- Biases: \( b[10] \)

Tensor shapes in [ ]
Y = tf.nn.softmax(tf.matmul(X, W) + b)

**tensor shapes:**
- X[100, 748]
- W[748,10]
- b[10]

**matrix multiply**

**broadcast on all lines**
Cross entropy: \[- \sum Y'_i \cdot \log(Y_i)\]

actual probabilities, "one-hot" encoded

computed probabilities

this is a "6"
Demo
Accuracy

Cross entropy loss

Weights

Biases

Training digits

Test digits

92%
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

init = tf.initialize_all_variables()
# model
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)

# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])

# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```python
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)
```
sess = tf.Session()
sess.run(init)

for i in range(1000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data = {X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ?
    a, c = sess.run([accuracy, cross_entropy], feed_dict=train_data)

    # success on test data ?
    test_data = {X: mnist.test.images, Y_: mnist.test.labels}
    a, c, _ = sess.run([accuracy, cross_entropy, It], feed=test_data)
```python
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
init = tf.initialize_all_variables()

# model
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)

# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])

# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)

sess = tf.Session()
sess.run(init)

for i in range(10000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ? add code to print it
    a, c = sess.run([accuracy, cross_entropy], feed=train_data)

# success on test data ?
    test_data={X:mnist.test.images, Y_:mnist.test.labels}
    a, c = sess.run([accuracy, cross_entropy], feed=test_data)
```
Softmax
Cross-entropy
Mini-batch
Go deep!
Let’s try 5 fully-connected layers!

sigmoid function

softmax

overKil
TensorFlow - initialisation

K = 200
L = 100
M = 60
N = 30

W1 = tf.Variable(tf.truncated_normal([28*28, K], stddev=0.1))
B1 = tf.Variable(tf.zeros([K]))

W2 = tf.Variable(tf.truncated_normal([K, L], stddev=0.1))
B2 = tf.Variable(tf.zeros([L]))

W3 = tf.Variable(tf.truncated_normal([L, M], stddev=0.1))
B3 = tf.Variable(tf.zeros([M]))

W4 = tf.Variable(tf.truncated_normal([M, N], stddev=0.1))
B4 = tf.Variable(tf.zeros([N]))

W5 = tf.Variable(tf.truncated_normal([N, 10], stddev=0.1))
B5 = tf.Variable(tf.zeros([10]))
X = tf.reshape(X, [-1, 28*28])

Y1 = tf.nn.sigmoid(tf.matmul(X, W1) + B1)
Y2 = tf.nn.sigmoid(tf.matmul(Y1, W2) + B2)
Y3 = tf.nn.sigmoid(tf.matmul(Y2, W3) + B3)
Y4 = tf.nn.sigmoid(tf.matmul(Y3, W4) + B4)
Y = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
Demo - slow start?

**Accuracy**

- Training accuracy
- Test accuracy

**Cross entropy loss**

- Training loss
- Test loss
Relu!
RELU

RELU = Rectified Linear Unit

\[ Y = \text{tf.nn.relu}(\text{tf.matmul}(X, W) + b) \]
Demo - noisy accuracy curve?

Accuracy

Cross entropy loss

yuck!
Slow down...

Learning rate decay
Learning rate decay

Learning rate 0.003 at start then dropping exponentially to 0.0001.
Demo - dying neurons
Dropout
Dropout

\[
p\text{keep} = \text{tf.placeholder(tf.float32)}
\]

\[
Y_f = \text{tf.nn.relu(tf.matmul}(X, W) + B)
\]

\[
Y = \text{tf.nn.dropout}(Y_f, p\text{keep})
\]
Dropout

Logits

Activations

with dropout
Accuracy

Cross entropy loss

<table>
<thead>
<tr>
<th>Training digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>407 1 034 180</td>
</tr>
<tr>
<td>500 8 360 183</td>
</tr>
<tr>
<td>600 3 185 349</td>
</tr>
<tr>
<td>1925802010</td>
</tr>
<tr>
<td>8483984258</td>
</tr>
<tr>
<td>0614538551</td>
</tr>
<tr>
<td>3276015854</td>
</tr>
<tr>
<td>8662091045</td>
</tr>
<tr>
<td>9964825151</td>
</tr>
<tr>
<td>0581127990</td>
</tr>
</tbody>
</table>

Test digits

- 100%
- 98%
- 96%
- 94%
- 92%
- 90%

Logits

Activations
All the party tricks

**Accuracy**

- Sigmoid, learning rate = 0.003
- **RELU**, learning rate = 0.003
- **RELU**, decaying learning rate 0.003 -> 0.0001
- **RELU**, decaying learning rate 0.003 -> 0.0001 and dropout 0.75

- 97.9%
- 98.2%

**Cross entropy loss**

- **RELU**, decaying learning rate 0.003 -> 0.0001 and dropout 0.75

- 98.2% sustained
Overfitting

Cross-entropy loss

- training loss
- test loss

Overfitting
Overfitting ?!

Too many neurons

Not enough DATA

BAD Network
Convolutional layer

Convolutional layer

- Padding
- Stride

$W_{[4, 4, 3]} \rightarrow W_{[4, 4, 3]} \rightarrow W_{[4, 4, 3, 2]}

Filter size
Input channels
Output channels

Convolutional subsampling
Convolutional subsampling
Convolutional subsampling
Hacker’s tip

ALL Convolutional
Convolutional neural network

- Convolutional layer, 4 channels
  \( W_1[5, 5, 1, 4] \) stride 1

- Convolutional layer, 8 channels
  \( W_2[4, 4, 4, 8] \) stride 2

- Convolutional layer, 12 channels
  \( W_3[4, 4, 8, 12] \) stride 2

- Fully connected layer
  \( W_4[7x7x12, 200] \)

- Softmax readout layer
  \( W_5[200, 10] \)

+ Biases on all layers
K=4
L=8
M=12

\[ W_1 = \text{tf.Variable(tf.truncated_normal([5, 5, 1, K], stddev=0.1))} \]

\[ B_1 = \text{tf.Variable(tf.ones([K])/10)} \]

\[ W_2 = \text{tf.Variable(tf.truncated_normal([5, 5, K, L], stddev=0.1))} \]

\[ B_2 = \text{tf.Variable(tf.ones([L])/10)} \]

\[ W_3 = \text{tf.Variable(tf.truncated_normal([4, 4, L, M], stddev=0.1))} \]

\[ B_3 = \text{tf.Variable(tf.ones([M])/10)} \]

N=200

\[ W_4 = \text{tf.Variable(tf.truncated_normal([7*7*M, N], stddev=0.1))} \]

\[ B_4 = \text{tf.Variable(tf.ones([N])/10)} \]

\[ W_5 = \text{tf.Variable(tf.truncated_normal([N, 10], stddev=0.1))} \]

\[ B_5 = \text{tf.Variable(tf.zeros([10])/10)} \]
Y1 = tf.nn.relu(tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME') + B1)  
Y2 = tf.nn.relu(tf.nn.conv2d(Y1, W2, strides=[1, 2, 2, 1], padding='SAME') + B2)  
Y3 = tf.nn.relu(tf.nn.conv2d(Y2, W3, strides=[1, 2, 2, 1], padding='SAME') + B3)  
YY = tf.reshape(Y3, shape=[-1, 7 * 7 * M])  
Y4 = tf.nn.relu(tf.matmul(YY, W4) + B4)  
Y  = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
Data & Analytics

Accuracy

Cross entropy loss

Weights

Biases

Training digits

Test digits

98.9%
WTXH ???

Accuracy

Cross entropy loss

training accuracy
test accuracy

training loss
test loss

???
Bigger convolutional network + dropout

28x28x1

28x28x6

14x14x12

7x7x24

200
10

convolutional layer, 6 channels
$W_1[6, 6, 1, 6]$ stride 1

convolutional layer, 12 channels
$W_2[5, 5, 6, 12]$ stride 2

convolutional layer, 24 channels
$W_3[4, 4, 12, 24]$ stride 2

fully connected layer
$W_4[7x7x24, 200]$

softmax readout layer
$W_5[200, 10]$

+ biases on all layers

+ DROPOUT $p=0.75$
Accuracy

Cross entropy loss

Weights

Biases

Training digits

Test digits

46.3%
Accuracy

Cross entropy loss

- training accuracy
- test accuracy

with dropout
Too many neurons?!?

Not enough DATA

BAD Network

Learning rate decay

Relu!

Dropout

Softmax

Cross-entropy

Mini-batch

ALL

Convolu-tional

Overfitting ?!

Go deep
Have fun!

Google Cloud Platform - cloud.google.com

Cloud ML ALPHAYour TensorFlow models trained in Google’s cloud, fast.

Pre-trained models:
- Cloud Vision API
- Cloud Speech API ALPHAI
- Google Translate API

All code snippets are on GitHub: github.com/martin-gorner/tensorflow-mnist-tutorial

This presentation: goo.gl/pHeXe7

That’s all folks...
Keyboard shortcuts for the visualisation GUI:

1. display 1\textsuperscript{st} graph only
2. display 2\textsuperscript{nd} graph only
3. display 3\textsuperscript{rd} graph only
4. display 4\textsuperscript{th} graph only
5. display 5\textsuperscript{th} graph only
6. display 6\textsuperscript{th} graph only
7. display graphs 1 and 2
8. display graphs 4 and 5
9. display graphs 3 and 6
ESC or 0  back to displaying all graphs

SPACE . . . . . pause/resume
0 . . . . . . . box zoom mode (then use mouse)
H . . . . . . . reset all zooms
Ctrl-S . . . save current image
1. Theory (sit back and listen)
Softmax classifier, mini-batch, cross-entropy and how to implement them in Tensorflow (slides 1-14)

2. Practice
Open file: mnist_1.0_softmax.py
Run it, play with the visualisations (see instructions on previous slide), read and understand the code as well as the basic structure of a Tensorflow program.

3. Theory (sit back and listen)
Hidden layers, sigmoid activation function (slides 16-19)

4. Practice
Start from the file you have and add one or two hidden layers. Use `cross_entropy_with_logits` to avoid numerical instabilities with log(0).
Solution in: mnist_2.0_five_layers_sigmoid.py

5. Theory (sit back and listen)
The neural network toolbox: RELUs, learning rate decay, dropout, overfitting (slides 20-35)

6. Practice
Replace all your sigmoids with RELUs. Test. Then add learning rate decay from 0.003 to 0.0001 using the formula \( lr = lr_{\text{min}} + (lr_{\text{max}} - lr_{\text{min}}) \times \exp(-i/2000) \).
Solution in: mnist_2.1_five_layers_relu_lrdecay.py

7. Practice (if time allows)
Add dropout on all layers using a value between 0.5 and 0.8 for pkeep.
Solution in: mnist_2.2_five_layers_relu_lrdecay_dropout.py

8. Theory (sit back and listen)
Convolutional networks (slides 36-42)

9. Practice
Replace your model with a convolutional network, without dropout.
Solution in: mnist_3.0_convolutional.py

10. Practice (if time allows)
Try a bigger neural network (good hyperparameters on slide 44) and add dropout on the last layer.
Solution in: mnist_3.0_convolutional_bigger_dropout.py