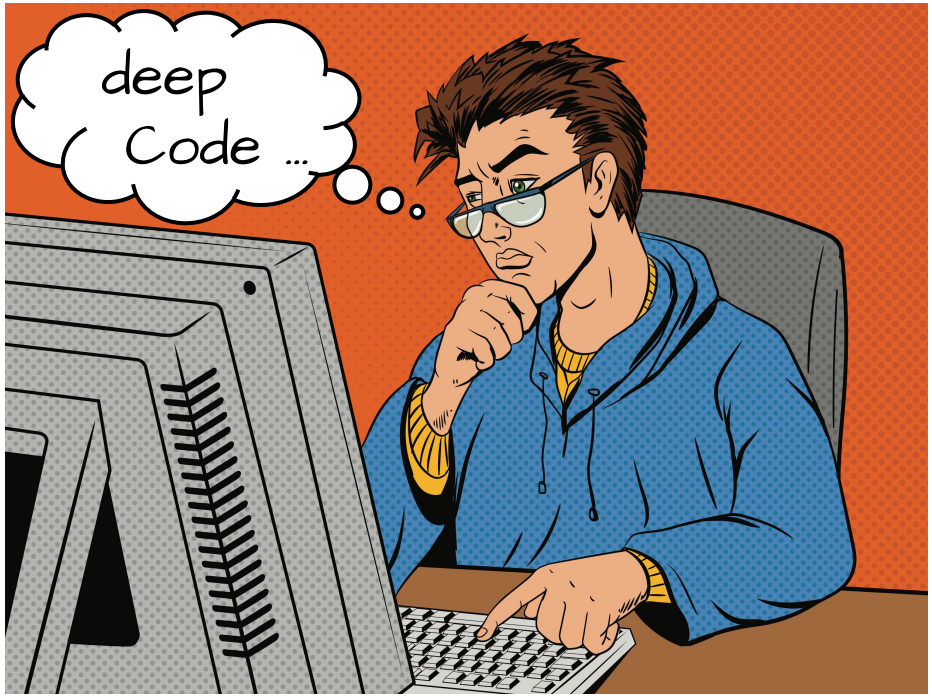
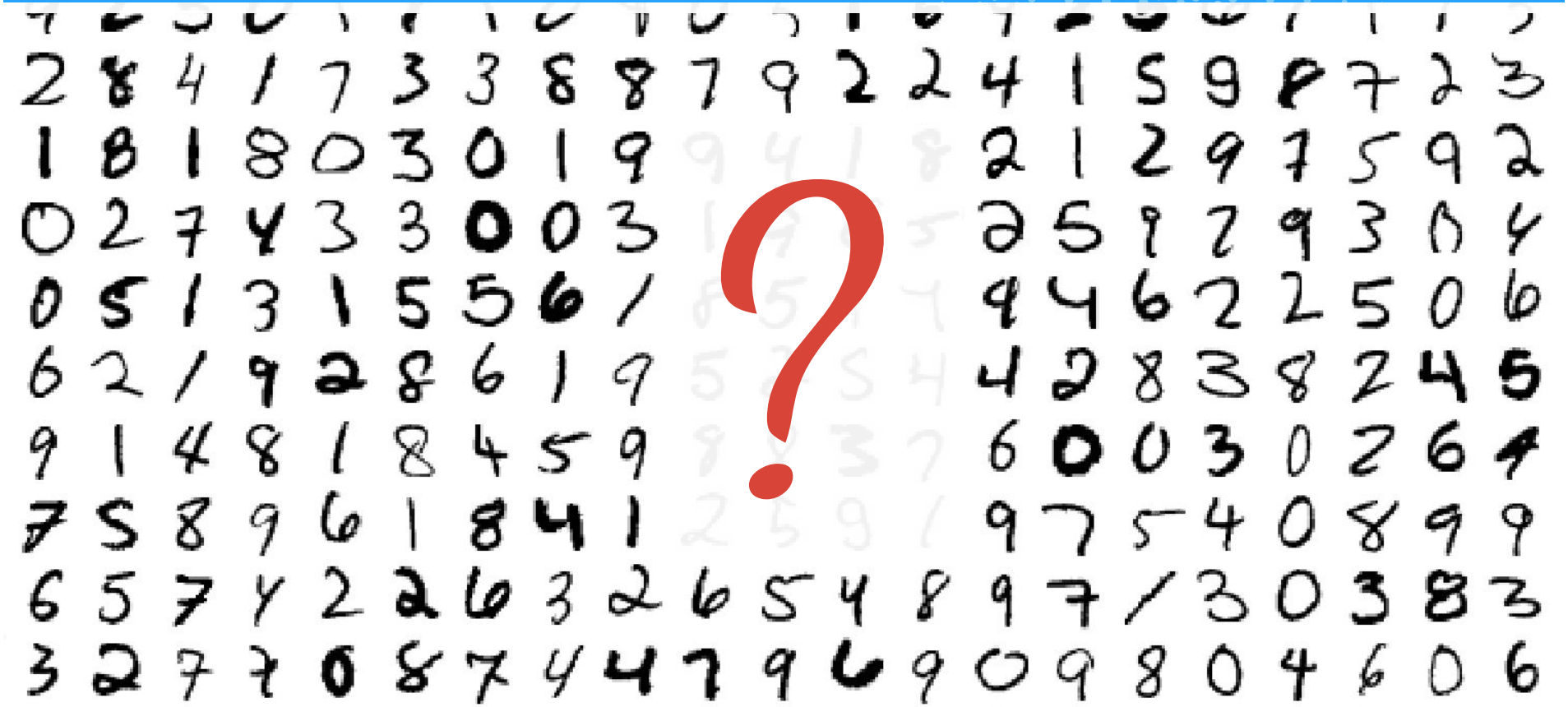
>TensorFlow and deep learning_

without a PhD

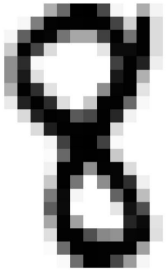


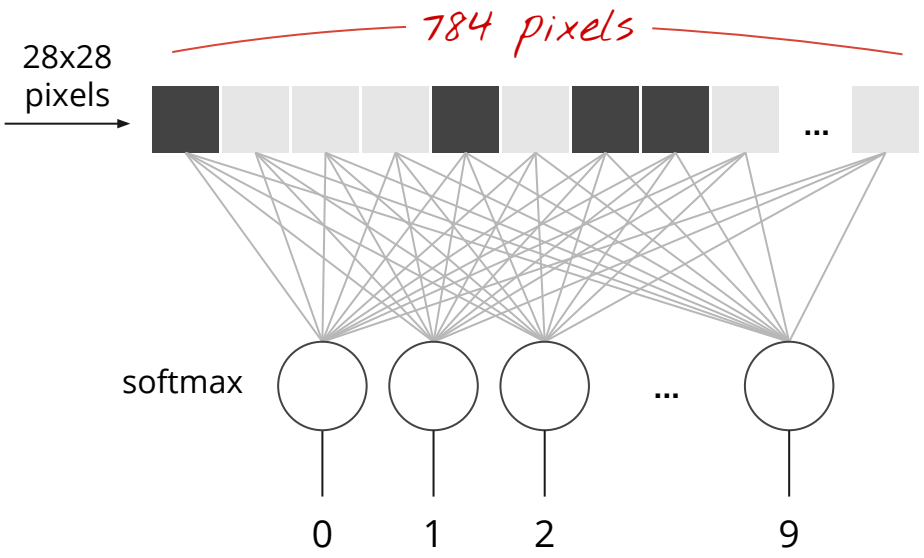
Hello World: handwritten digits classification - MNIST



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

Very simple model: softmax classification



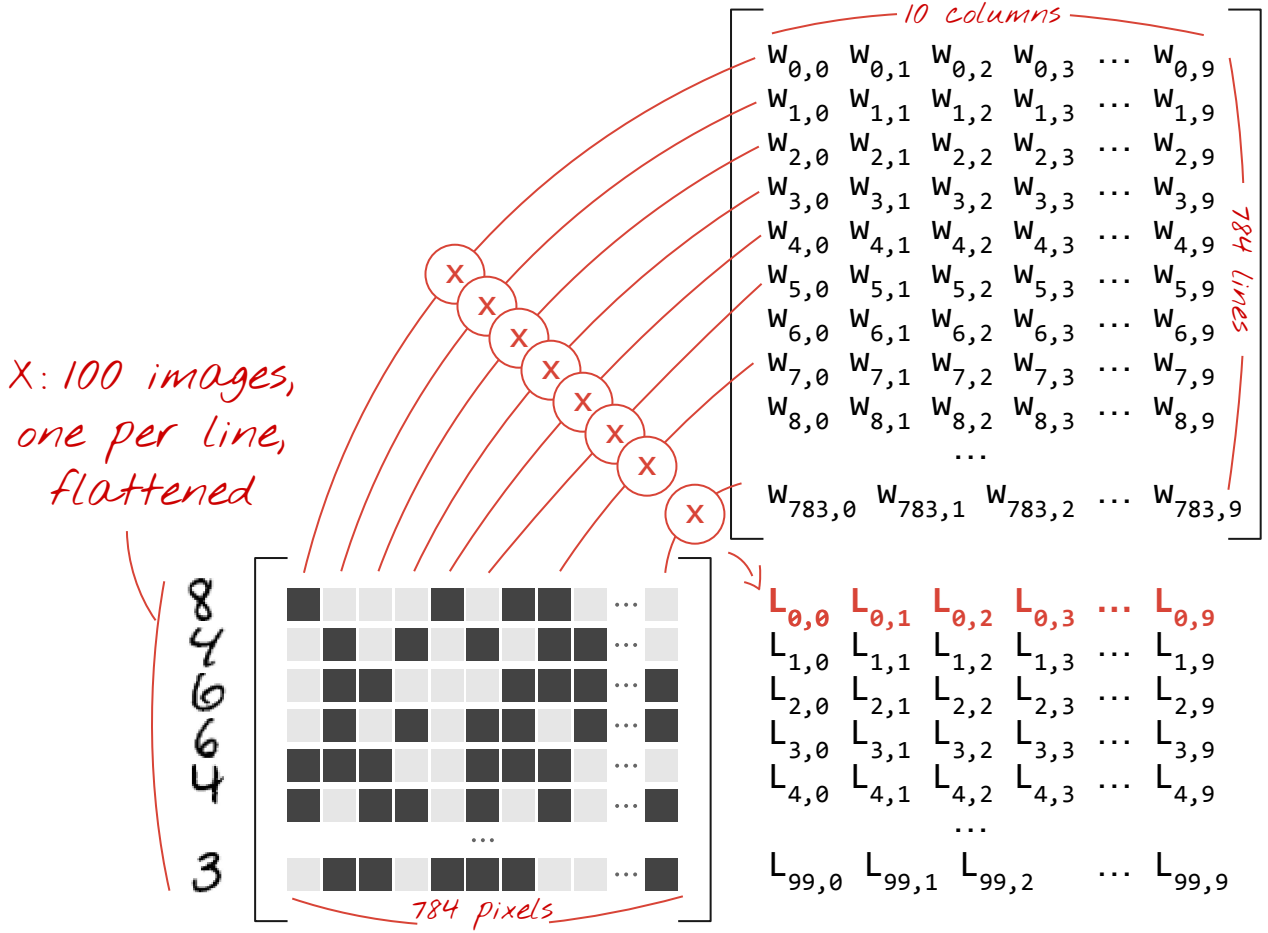


weighted sum of all pixels + bias

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

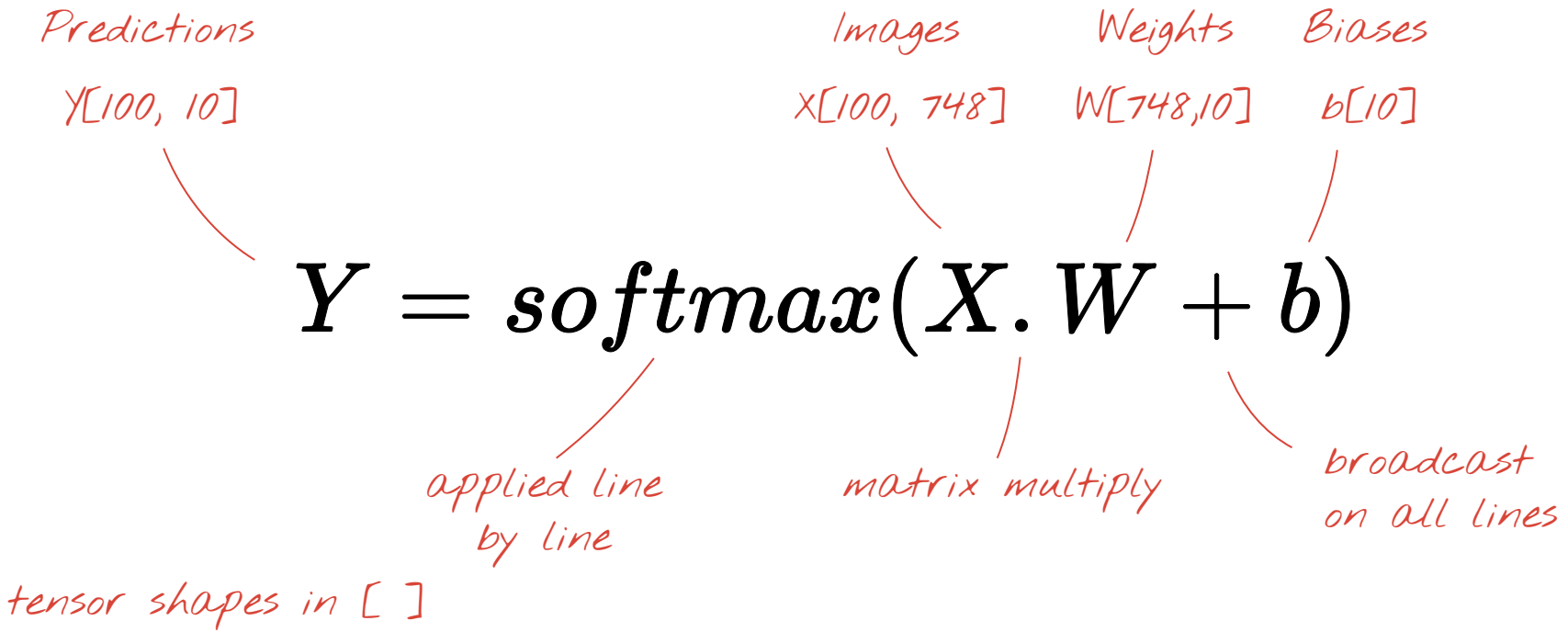
neuron outputs

In matrix notation, 100 images at a time

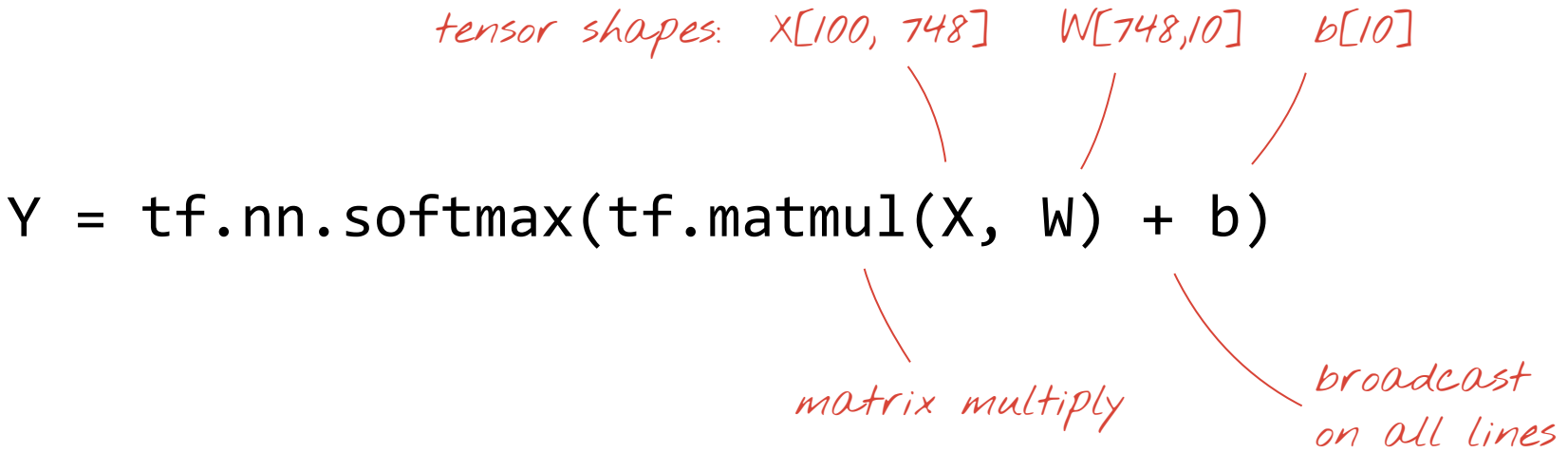


$$L = X.W + b$$

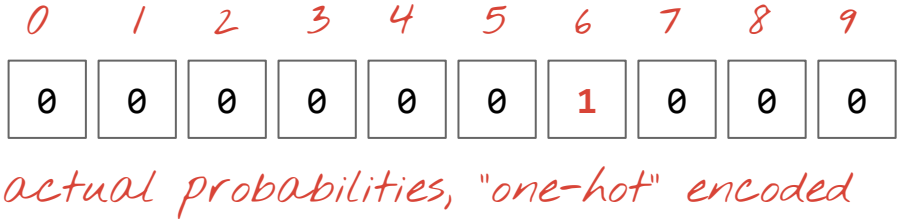
Softmax, on a batch of images



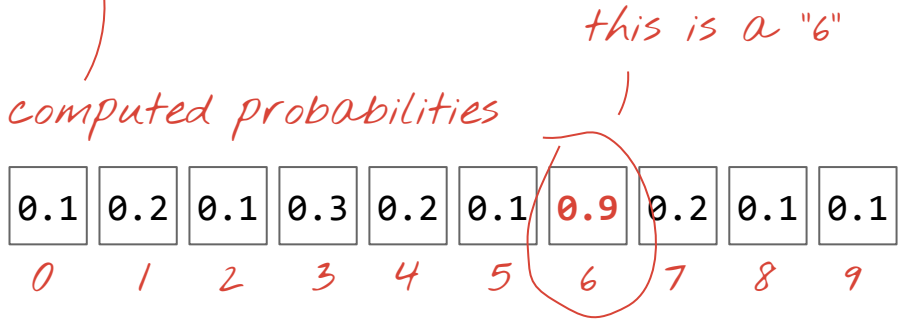
Now in TensorFlow (Python)



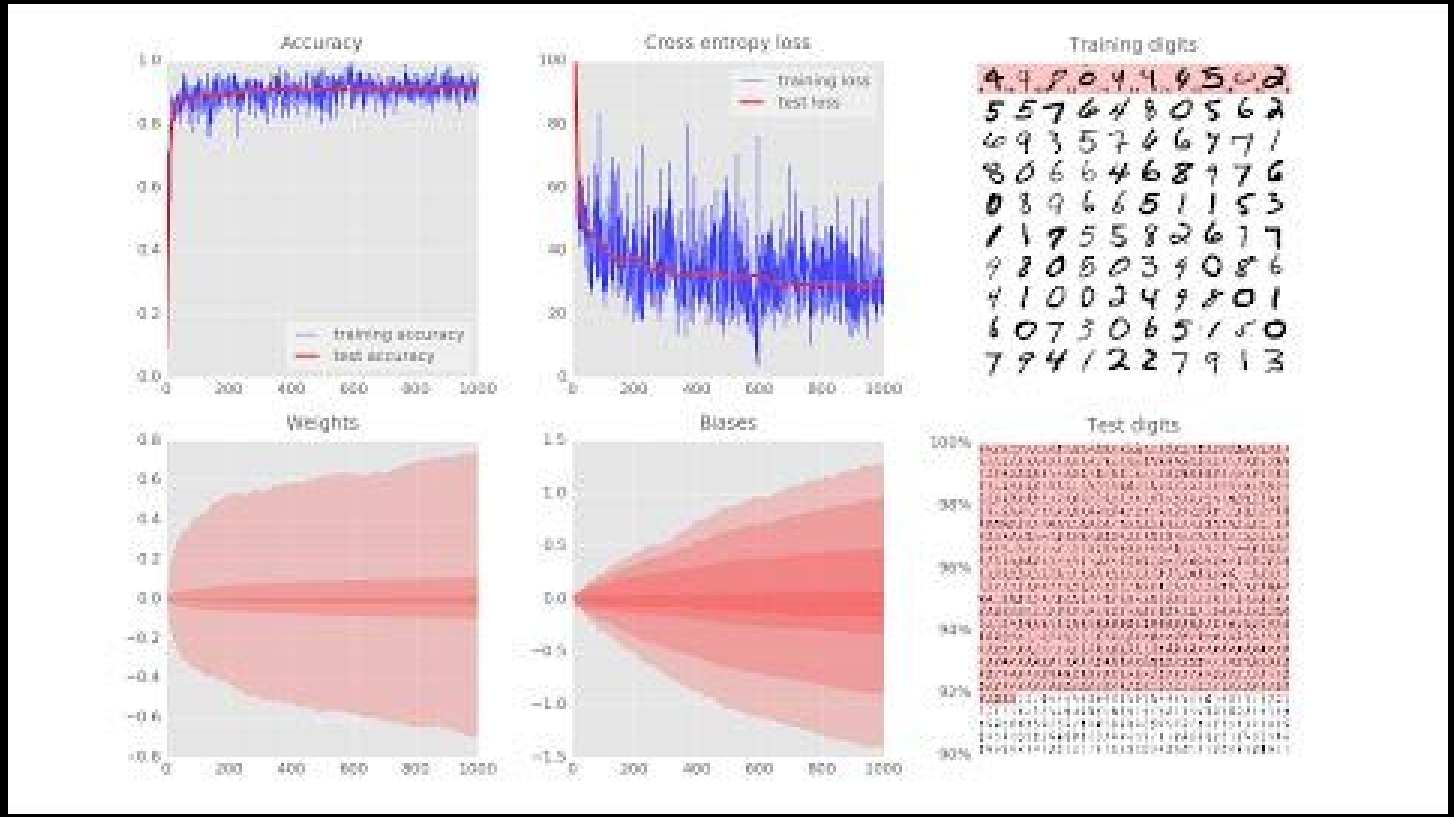
Success ?

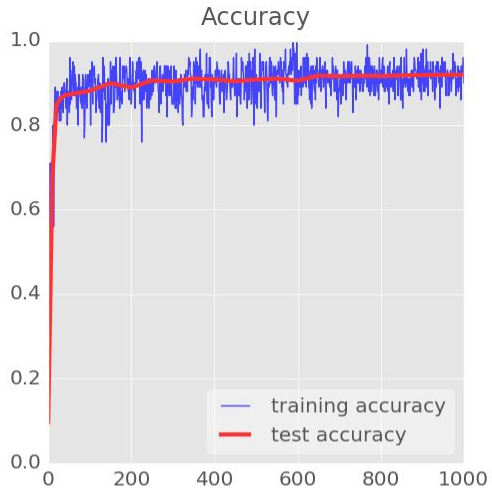


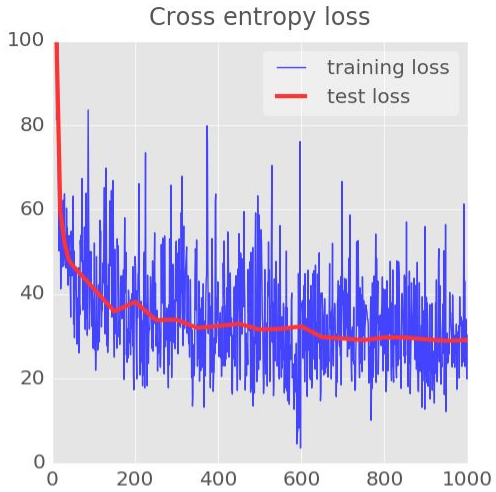
Cross entropy: $-\sum Y_i' . log(Y_i)$

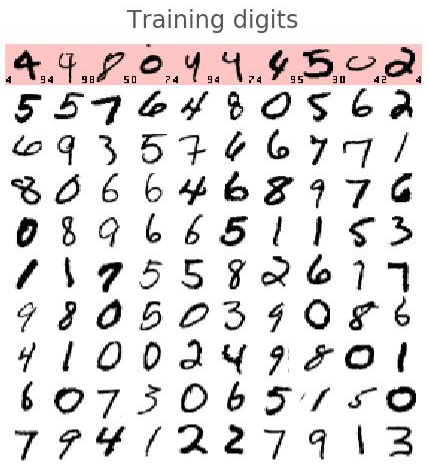


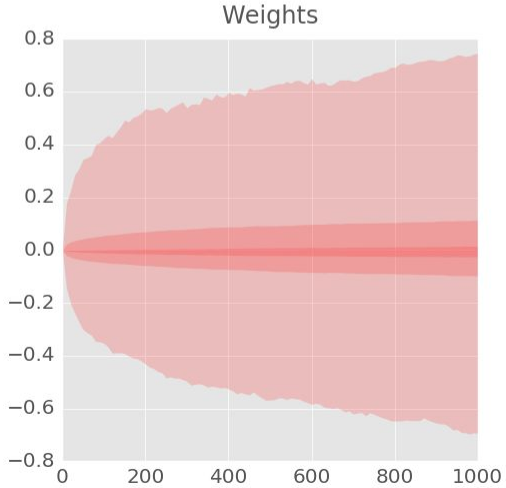
Demo

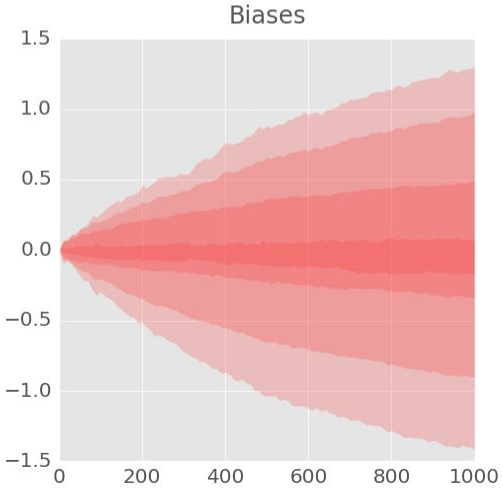

Accuracy
training accuracy
test accuracy
Cross entropy loss
training loss
test loss
Training digits
Weights
Biases
Test digits

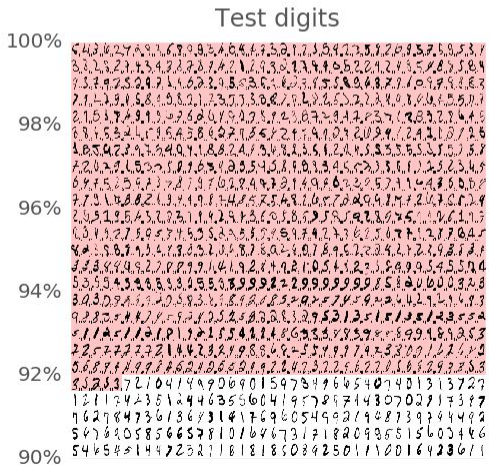

Accuracy
training accuracy
test accuracy


Cross entropy loss
training loss
test loss


Training digits


Weights


Biases


Test digits
92%

TensorFlow - initialisation

```
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()
```

this will become the batch size, 100

28 x 28 grayscale images

Training = computing variables W and b

TensorFlow - success metrics

flattening images

```
# 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])
```

"one-hot" encoded

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

"one-hot" decoding

```
# % 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))
```

TensorFlow - training

learning rate

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

loss function

TensorFlow - run !

```
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)
```

running a Tensorflow computation, feeding placeholders

Tip: do this every 100 iterations

TensorFlow - full python code

initialisation

```
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

```
# 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])
```

success metrics

```
# 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))
```

training step

```
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)
```

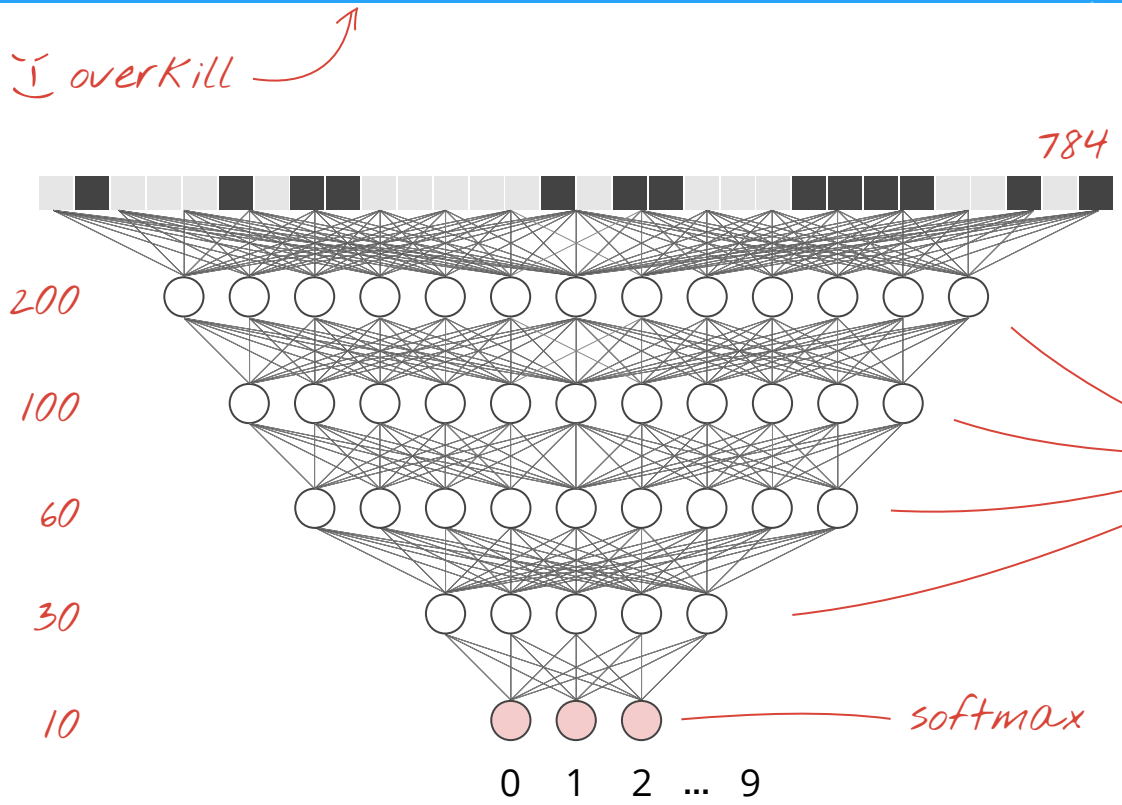
Run

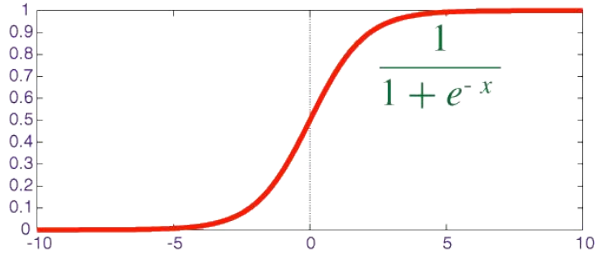
Cookbook

Softmax
Cross-entropy
Mini-batch


Go deep !

Let's try 5 fully-connected layers !





sigmoïd function

TensorFlow - initialisation

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

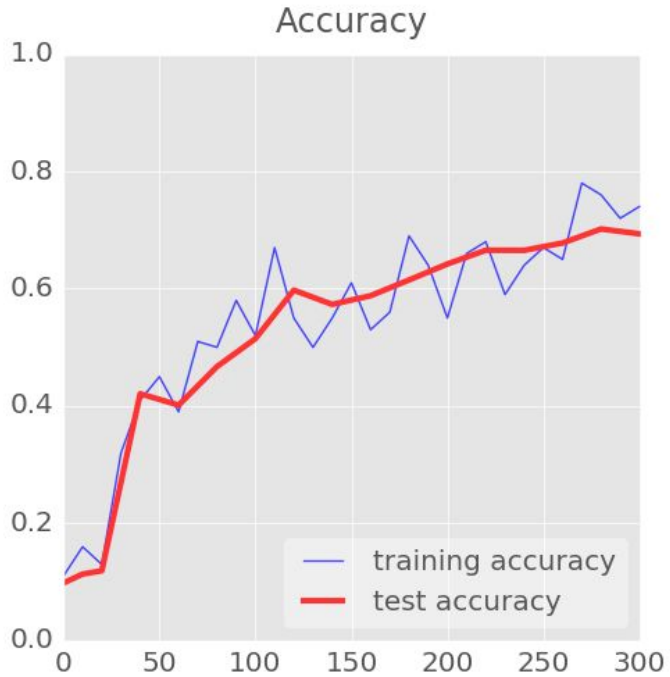
weights initialised with random values

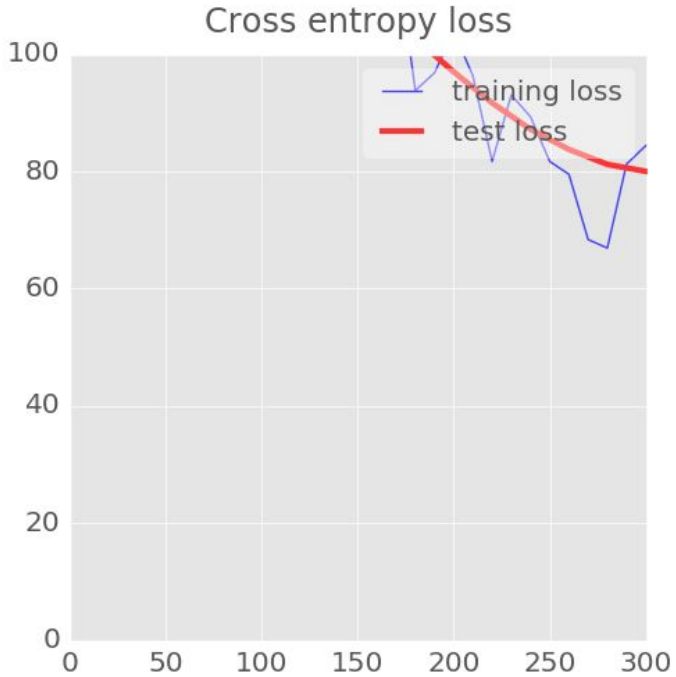
```
W1 = tf.Variable(tf.truncated_normal([28*28, K] ,stddev=0.1))
B1 = tf.Variable(tf.zeros([K])

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]))
```

TensorFlow - the model


```
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 ?

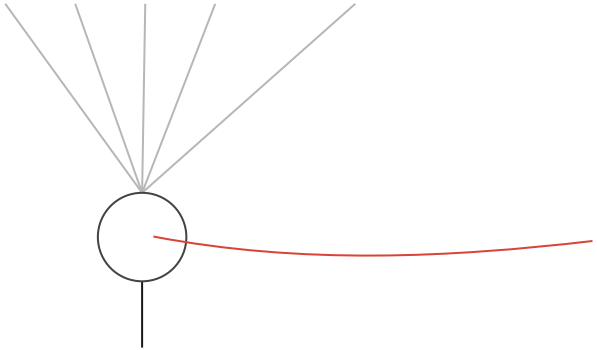


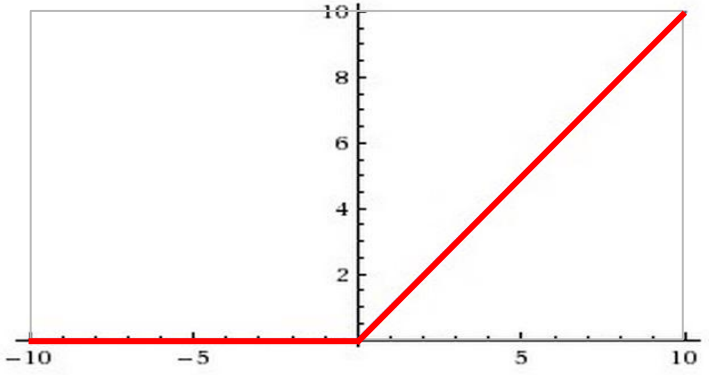



Relu !

RELU

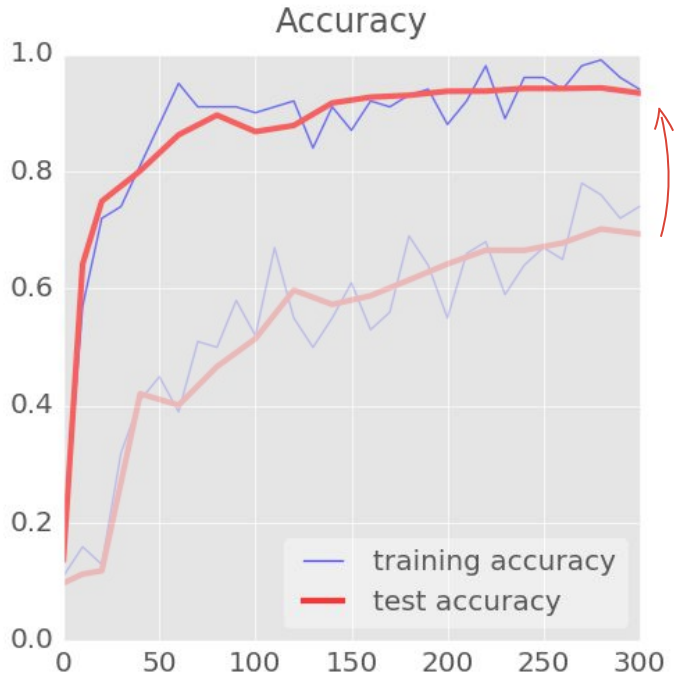
RELU = Rectified Linear Unit

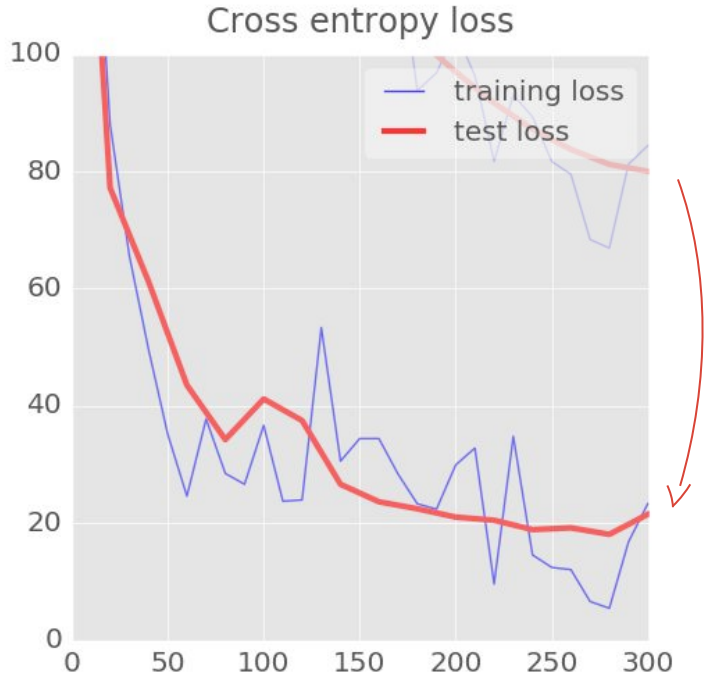




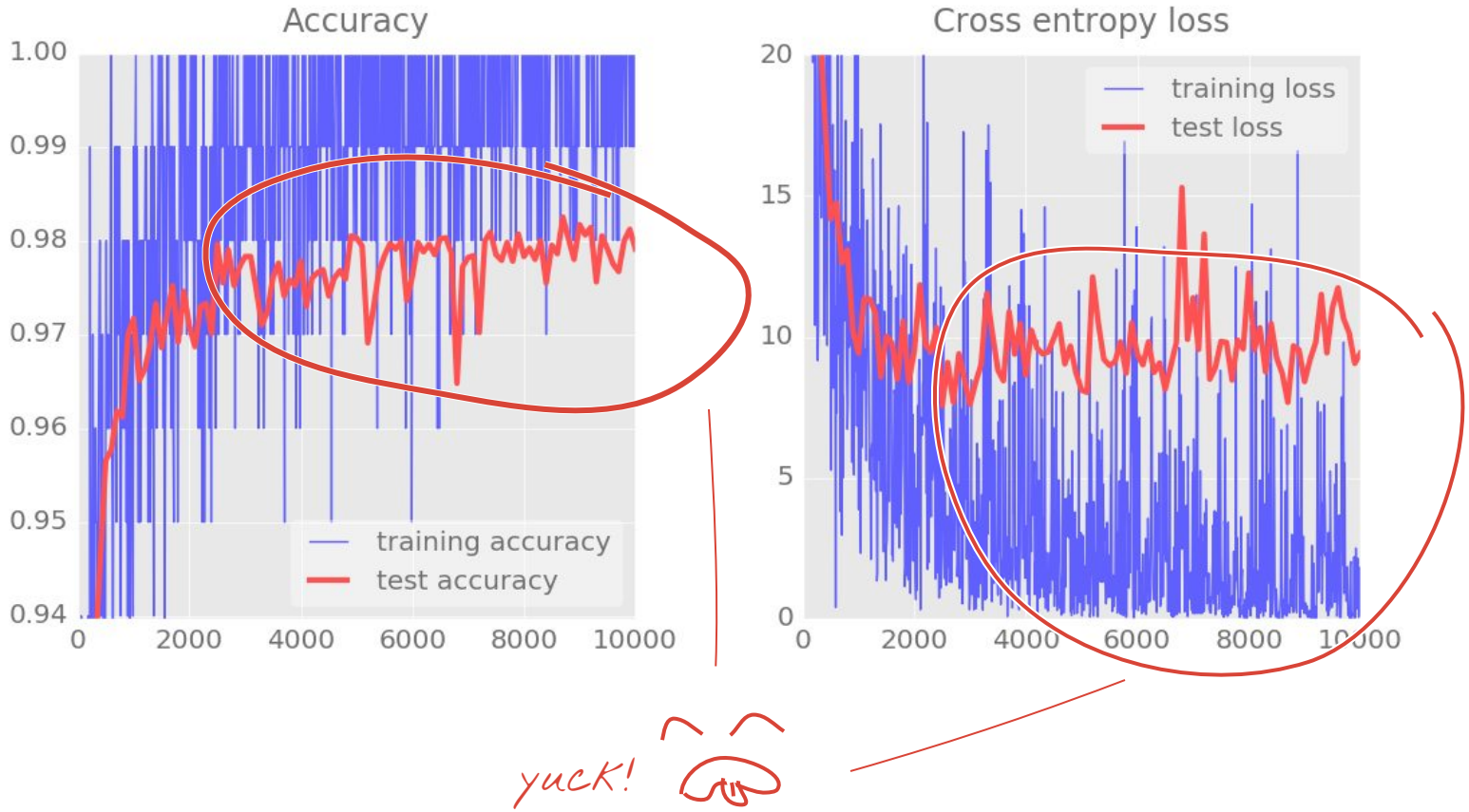

```
Y = tf.nn.relu(tf.matmul(X, W) + b)
```

RELU


Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300
training accuracy
test accuracy

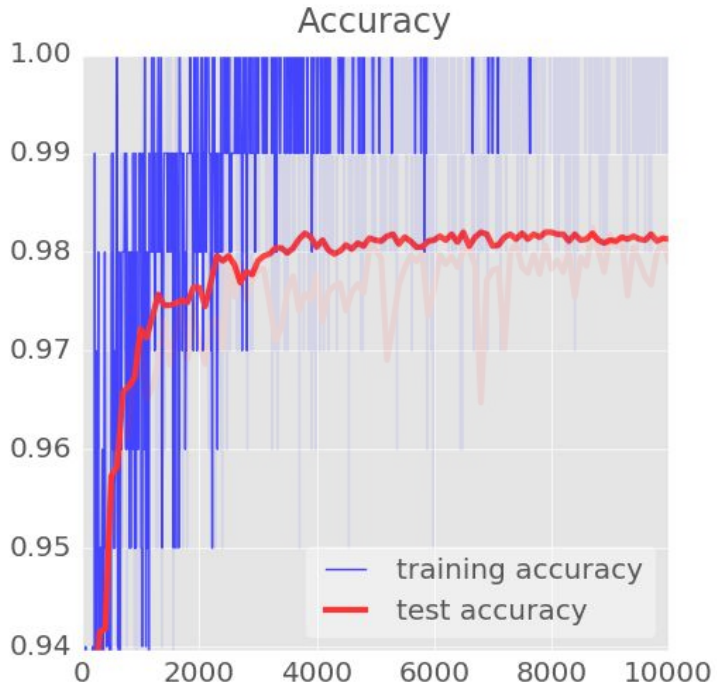

Cross entropy loss
100
80
60
40
20
0
0
50
100
150
200
250
300
training loss
test loss

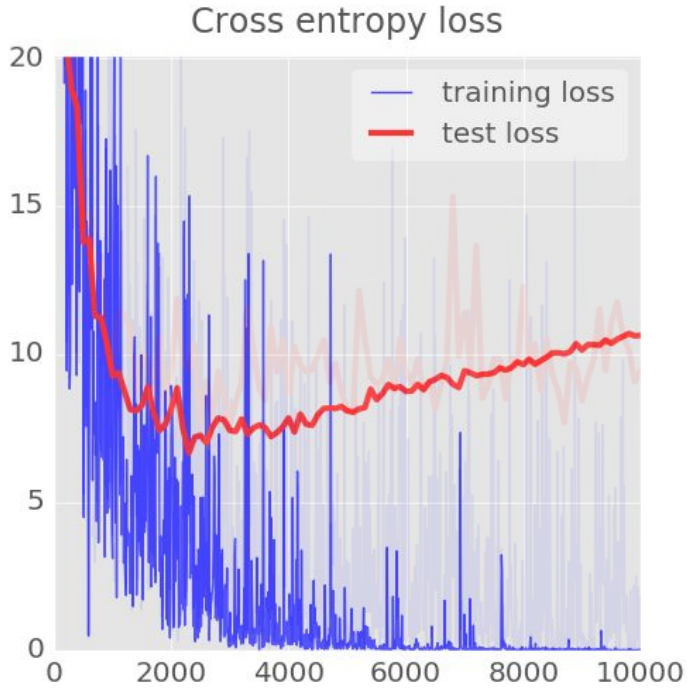
Demo - noisy accuracy curve ?



Slow down...

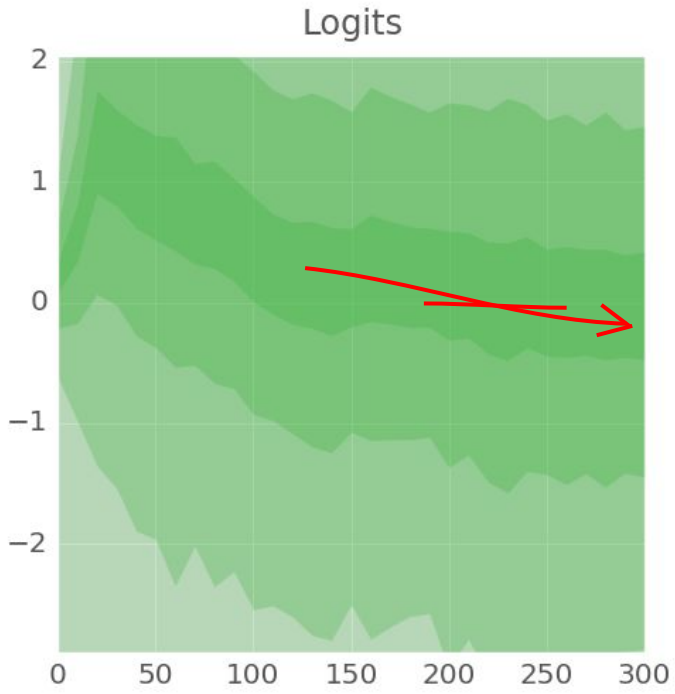
Learning rate decay

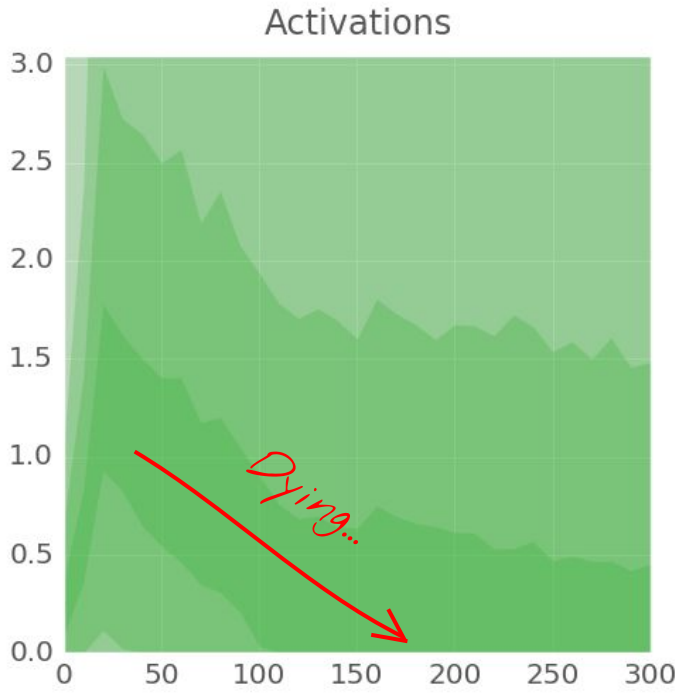




Learning rate 0.003 at start then dropping exponentially to 0.0001

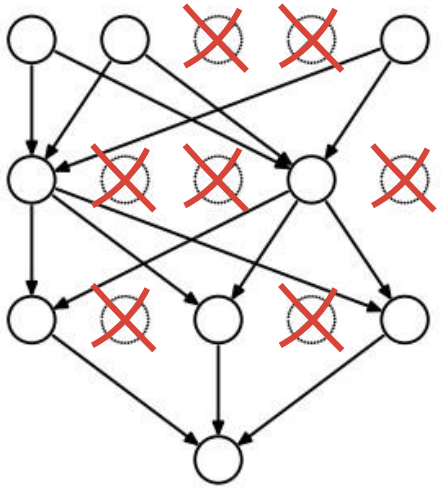
Demo - dying neurons



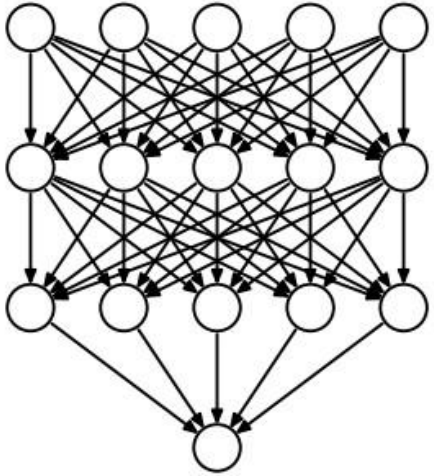



Dropout

Dropout



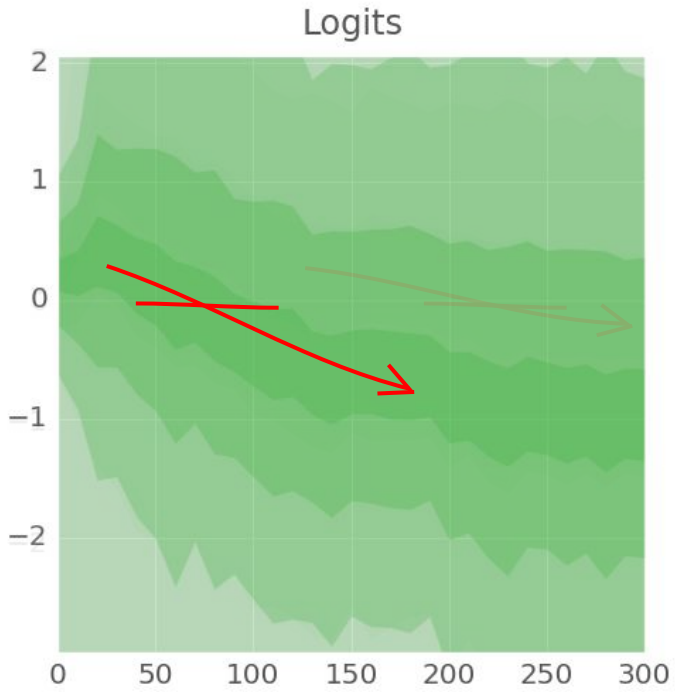
TRAINING
pKeep=0.75

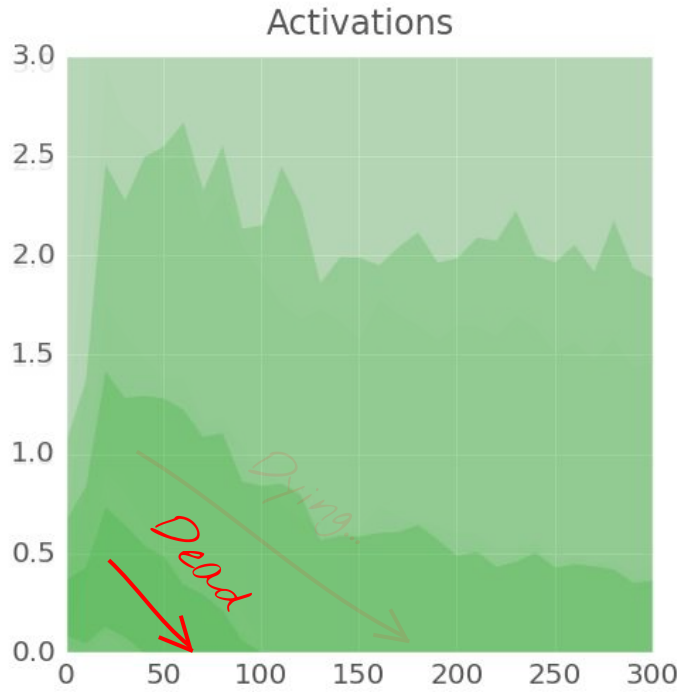


EVALUATION
pKeep=1

```
pkeep =
tf.placeholder(tf.float32)

Yf = tf.nn.relu(tf.matmul(X, W) + B)
Y  = tf.nn.dropout(Yf, pkeep)
```

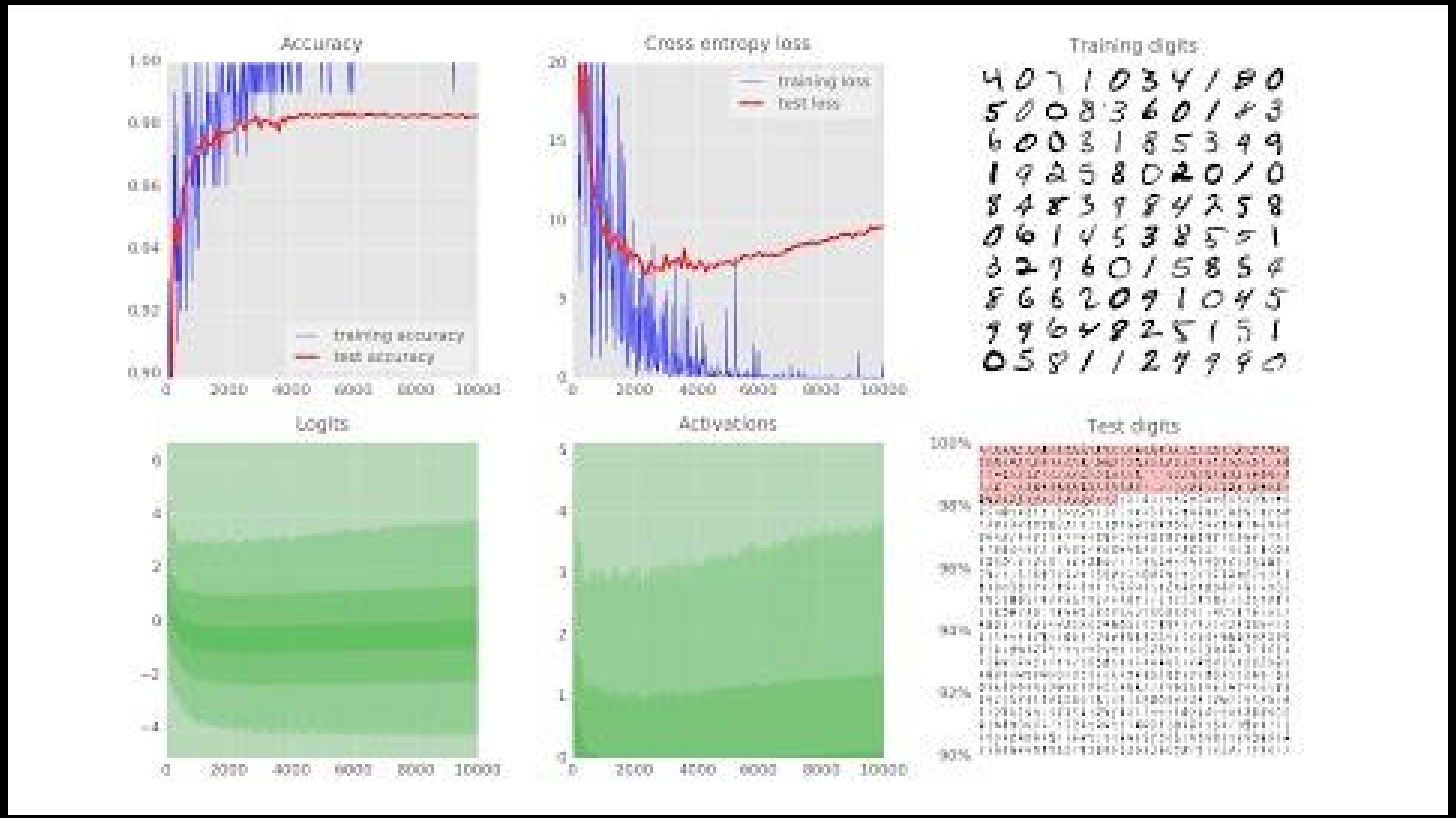
Dropout

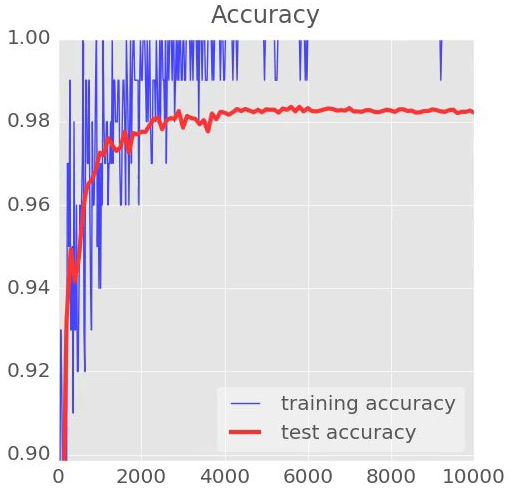


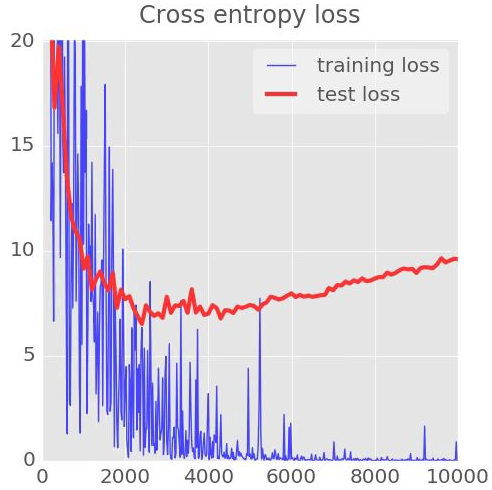


with dropout

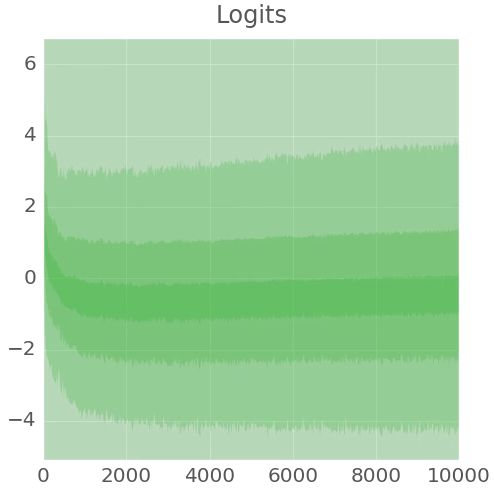
Demo

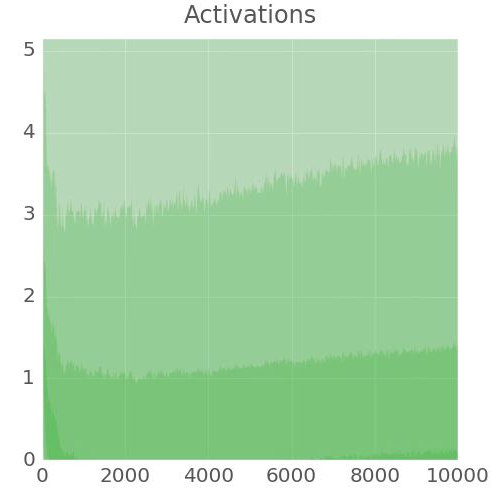

Accuracy
training accuracy
test accuracy
Cross entropy loss
training loss
test loss
Training digits
Logits
Activations
Test digits

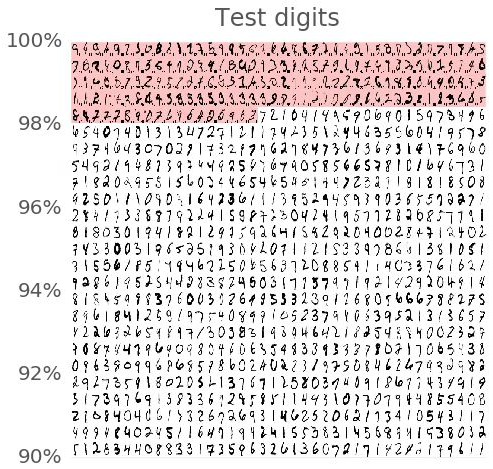

Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0
2000
4000
6000
8000
10000
training accuracy
test accuracy


Cross entropy loss
20
15
10
5
0
2000
4000
6000
8000
10000
training loss
test loss

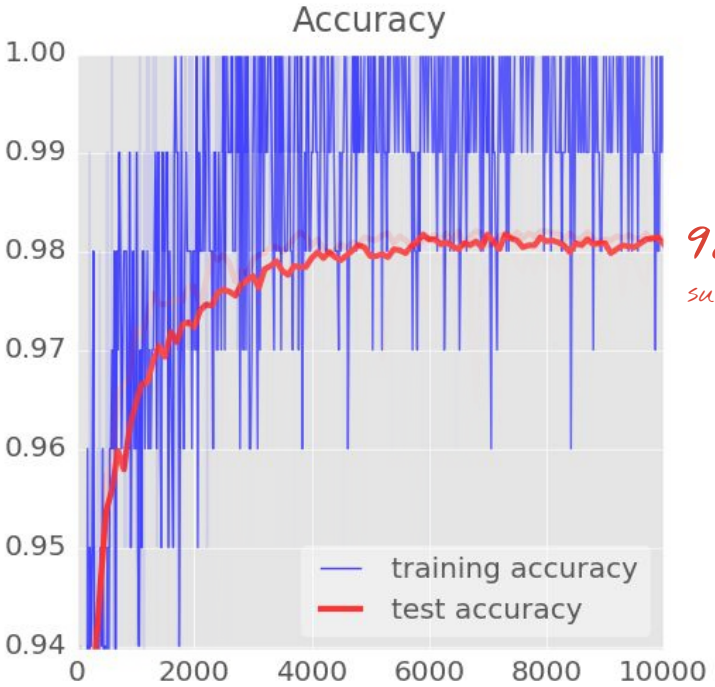

Training digits


Logits
6
4
2
0
−2
−4
0
2000
4000
6000
8000
10000

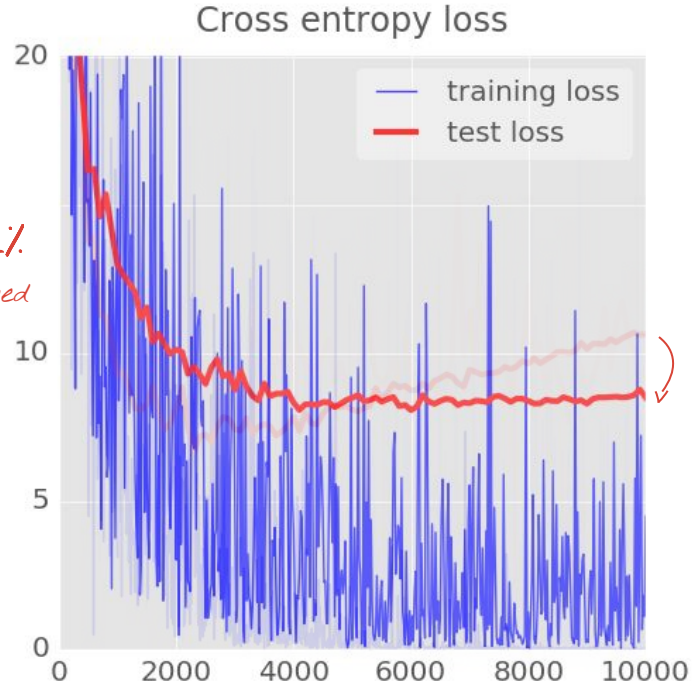

Activations
5
4
3
2
1
0
0
2000
4000
6000
8000
10000


Test digits
100%
98%
96%
94%
92%
90%
98%

All the party tricks

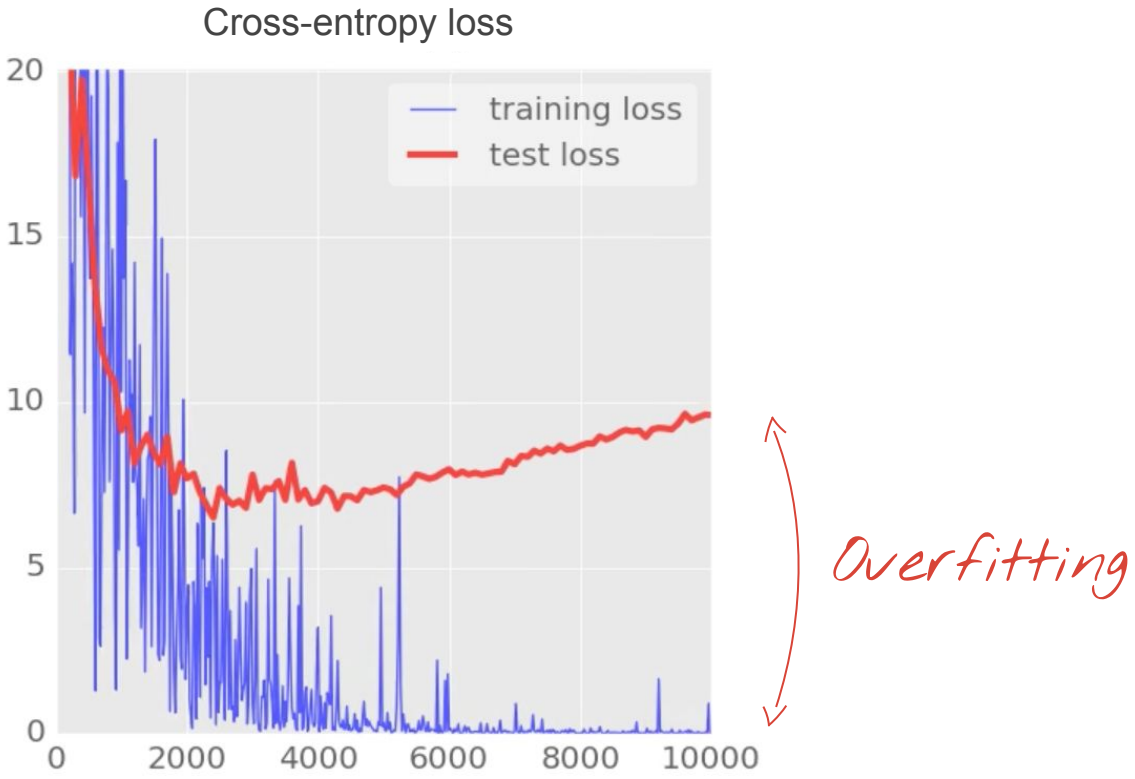


98.2% sustained



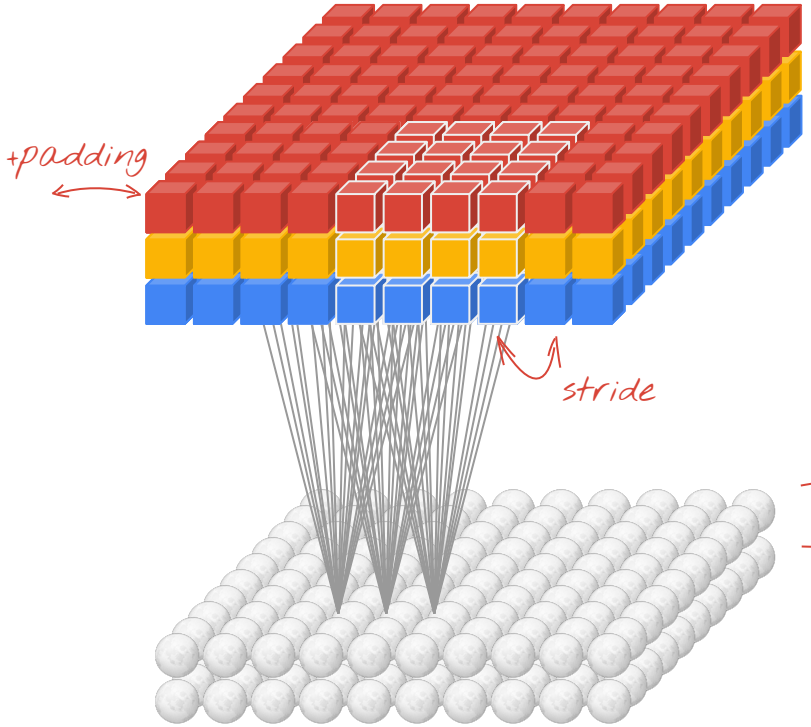
RELU, decaying learning rate 0.003 -> 0.0001 and dropout 0.75

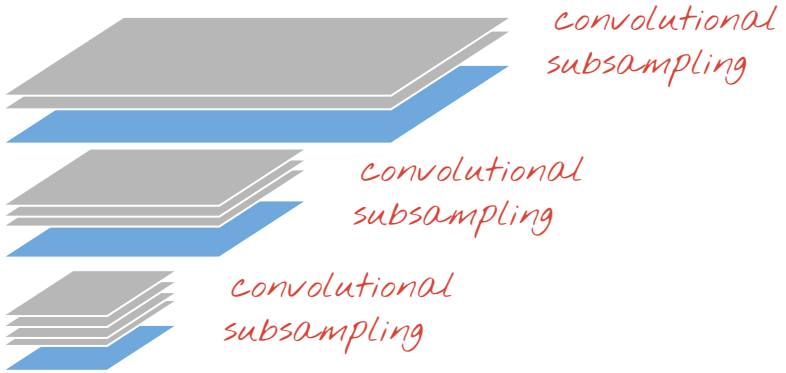
Overfitting




Overfitting
?!?
Too many
neurons
Not
enough
DATA
BAD
Network

Convolutional layer





$W_1[4, 4, 3]$

$W_2[4, 4, 3]$

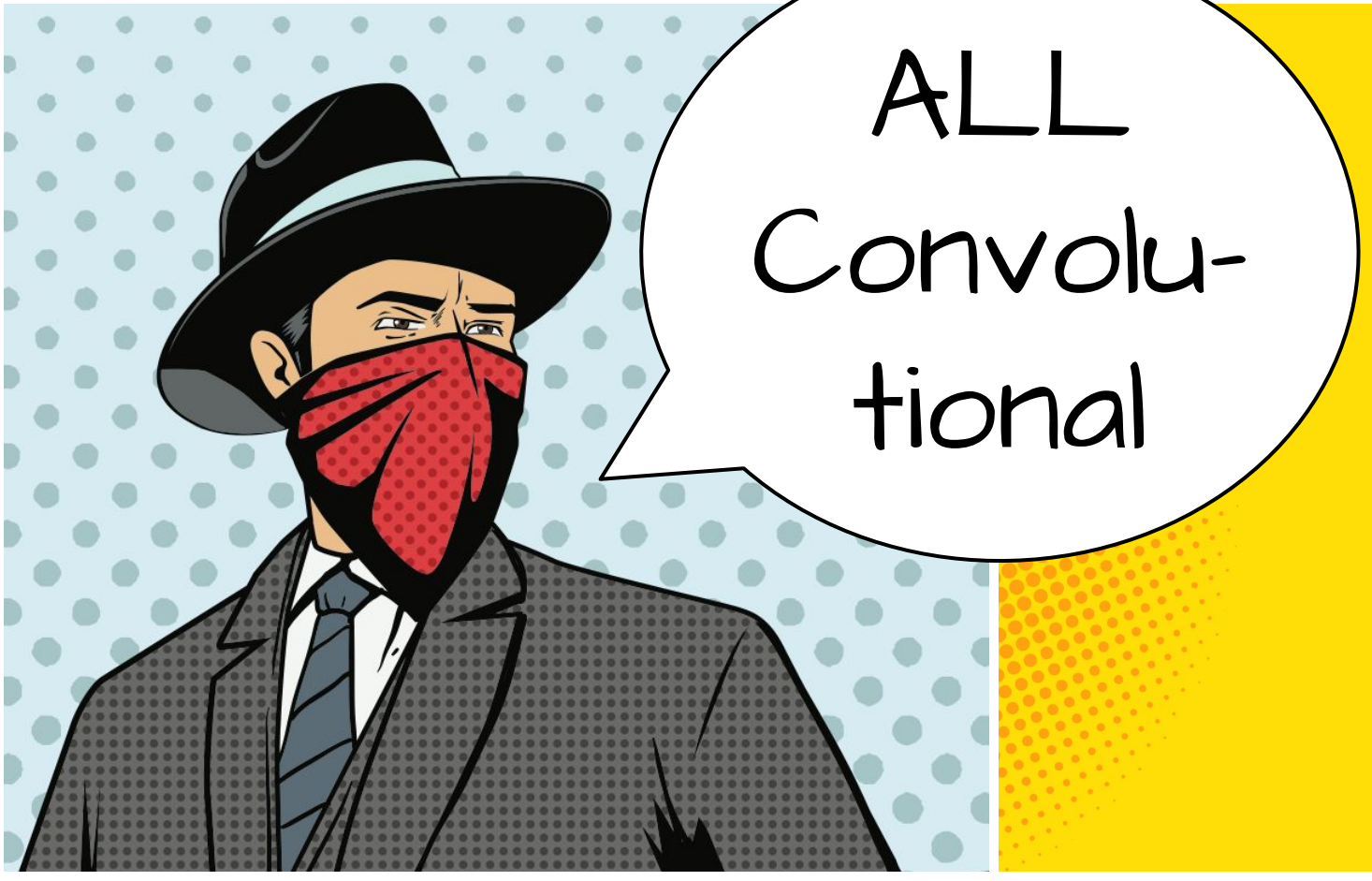
$W[4, 4, 3, 2]$

filter size

input channels

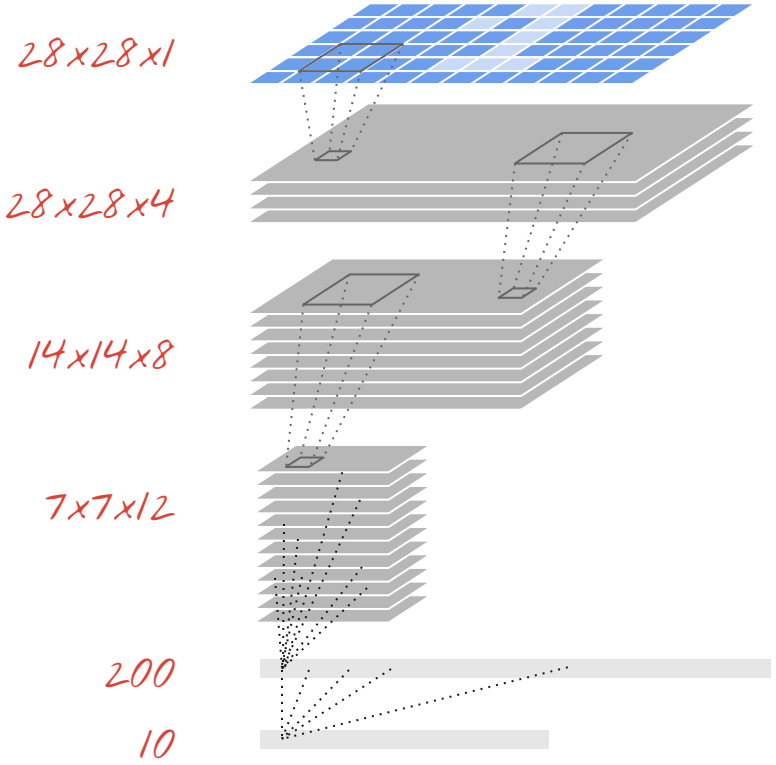
output channels

Hacker's tip



Convolutional neural network

+ biases on all layers



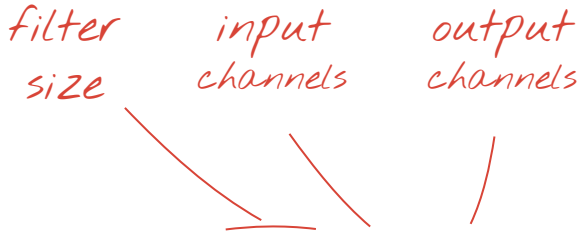
convolutional layer, 4 channels
W1[5, 5, 1, 4] stride 1

convolutional layer, 8 channels
W2[4, 4, 4, 8] stride 2

convolutional layer, 12 channels
W3[4, 4, 8, 12] stride 2

fully connected layer W4[7x7x12, 200]
softmax readout layer W5[200, 10]

Tensorflow - initialisation




```
K=4
L=8
M=12

W1 = tf.Variable(tf.truncated_normal([5, 5, 1, K] ,stddev=0.1))
B1 = tf.Variable(tf.ones([K])/10)
W2 = tf.Variable(tf.truncated_normal([5, 5, K, L] ,stddev=0.1))
B2 = tf.Variable(tf.ones([L])/10)
W3 = tf.Variable(tf.truncated_normal([4, 4, L, M] ,stddev=0.1))
B3 = tf.Variable(tf.ones([M])/10)

N=200

W4 = tf.Variable(tf.truncated_normal([7*7*M, N] ,stddev=0.1))
B4 = tf.Variable(tf.ones([N])/10)
W5 = tf.Variable(tf.truncated_normal([N, 10] ,stddev=0.1))
B5 = tf.Variable(tf.zeros([10])/10)
```

weights initialised with random values

Tensorflow - the model

input image batch
X[100, 28, 28, 1]

weights

stride

biases

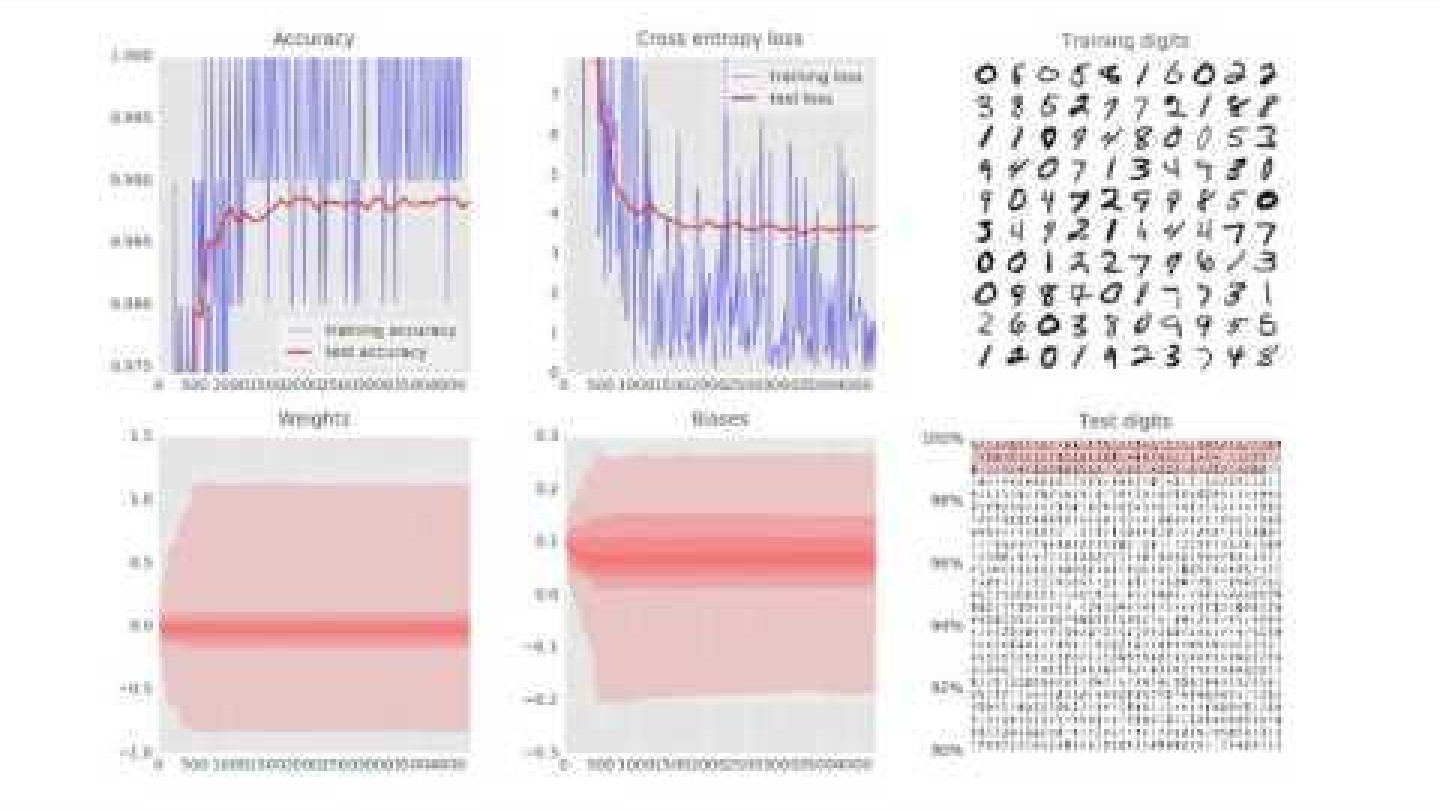
```
Y1 = tf.nn.relu(tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME') + B1)
Y2 = tf.nn.relu(tf.nn.conv2d(Y1, W2, strides=[1, 2, 2, 1], padding='SAME') + B2)
Y3 = tf.nn.relu(tf.nn.conv2d(Y2, W3, strides=[1, 2, 2, 1], padding='SAME') + B3)

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)
```

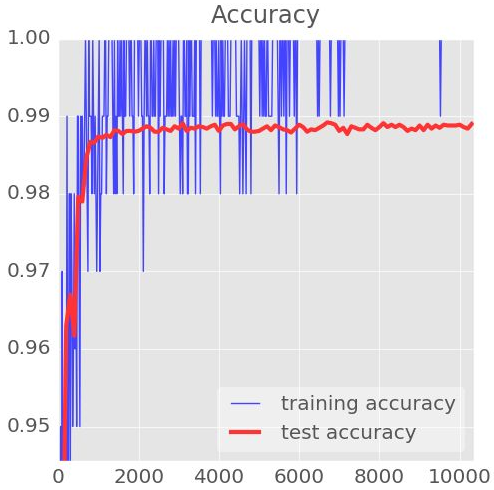
flatten all values for fully connected layer

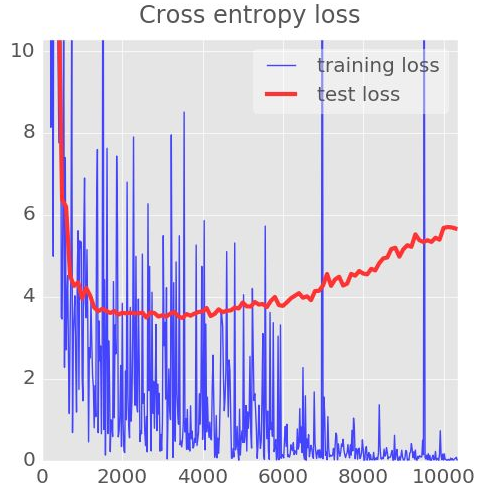
Y3 [100, 7, 7, 12]

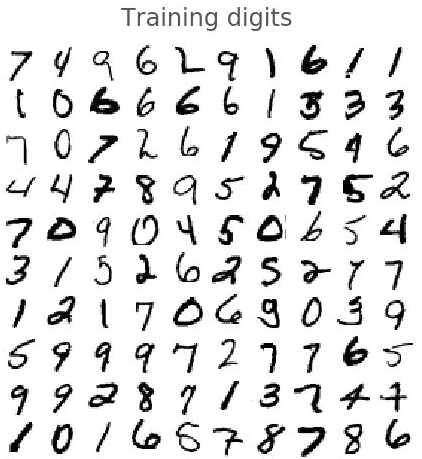
YY [100, 7x7x12]

Demo

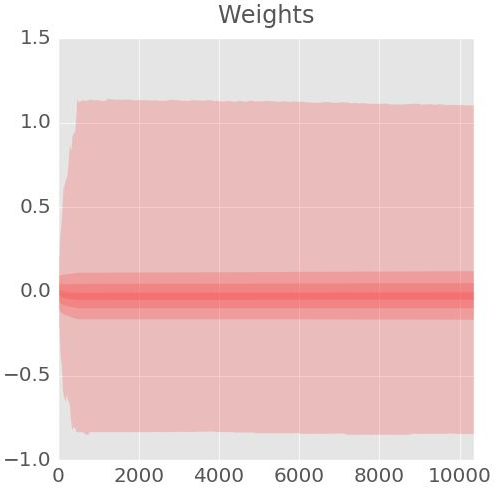

Accuracy
Cross entropy loss
Training digits
Weights
Biases
Test digits

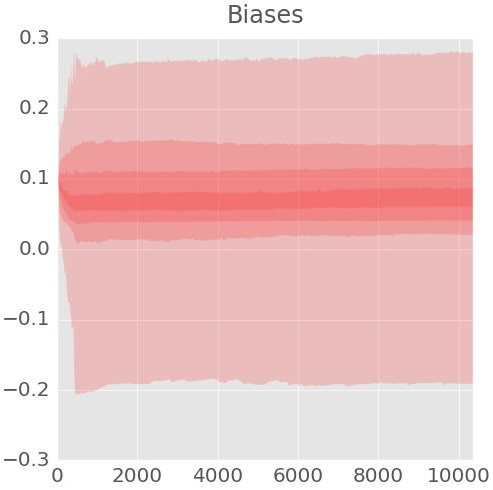

Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0
2000
4000
6000
8000
10000
training accuracy
test accuracy

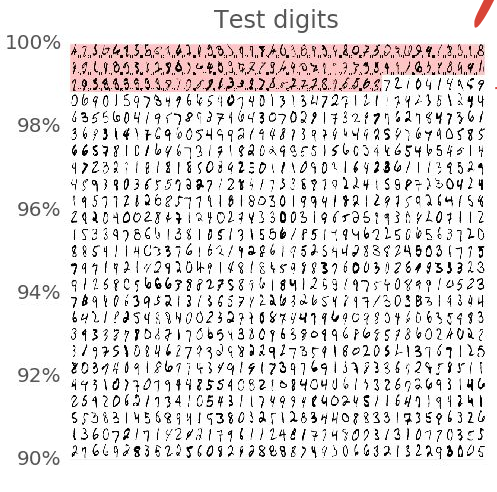

Cross entropy loss
10
8
6
4
2
0
0
2000
4000
6000
8000
10000
training loss
test loss


Training digits

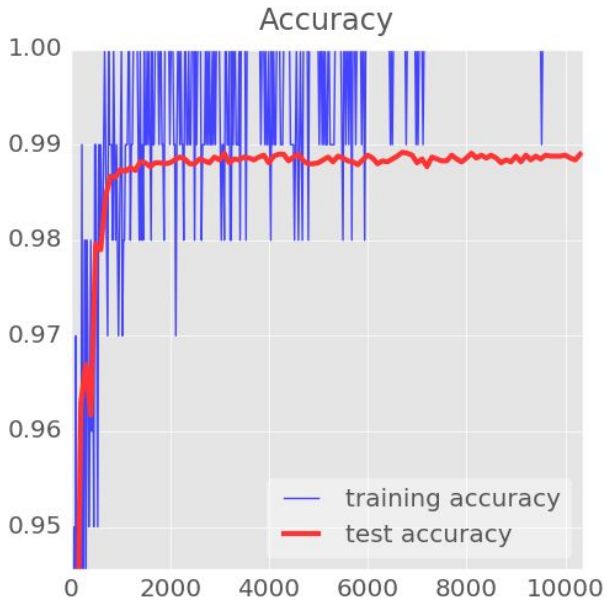
98.9%

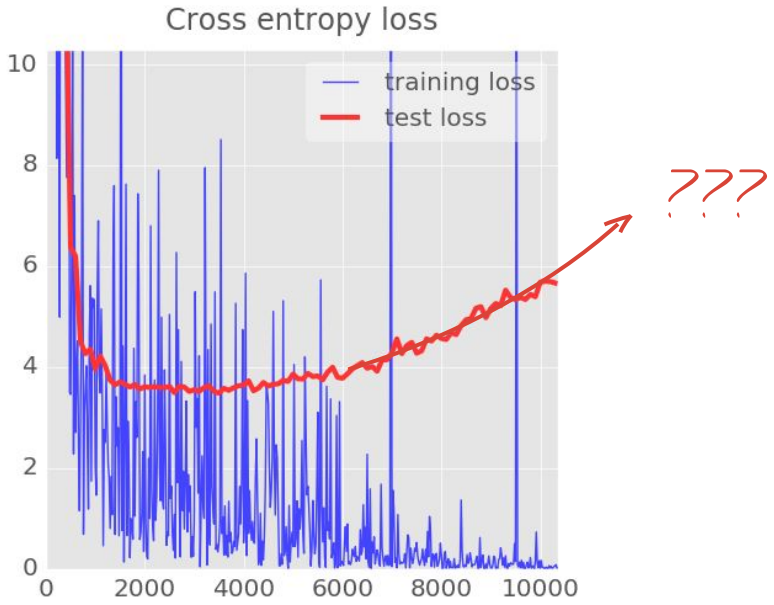

Weights
1.5
1.0
0.5
0.0
−0.5
−1.0
0
2000
4000
6000
8000
10000


Biases
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
2000
4000
6000
8000
10000


Test digits
100%
98%
96%
94%
92%
90%

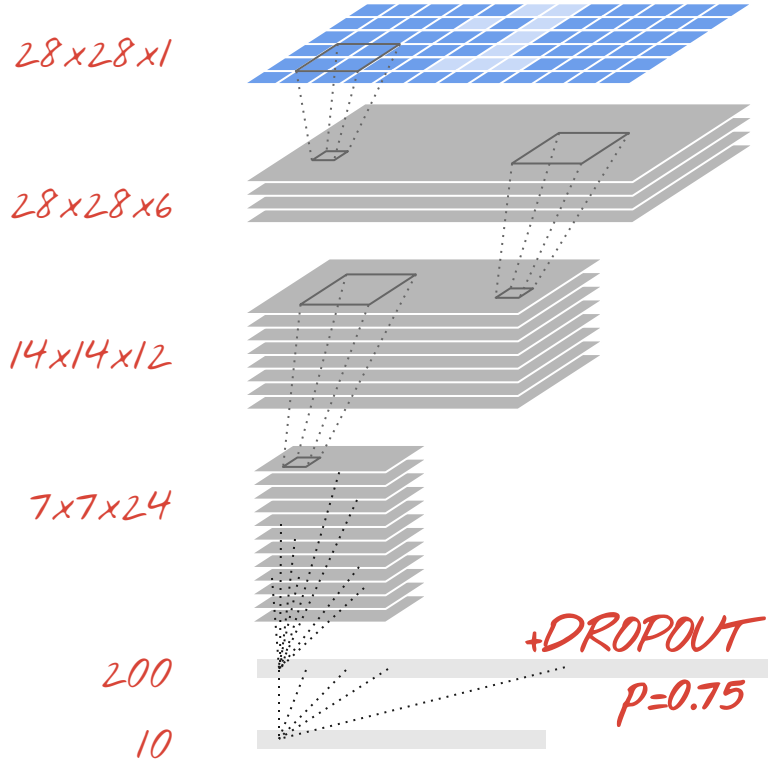
WTFH ???


Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0
2000
4000
6000
8000
10000
training accuracy
test accuracy


Cross entropy loss
10
8
6
4
2
0
0
2000
4000
6000
8000
10000
training loss
test loss
???

Bigger convolutional network + dropout

+ biases on all layers



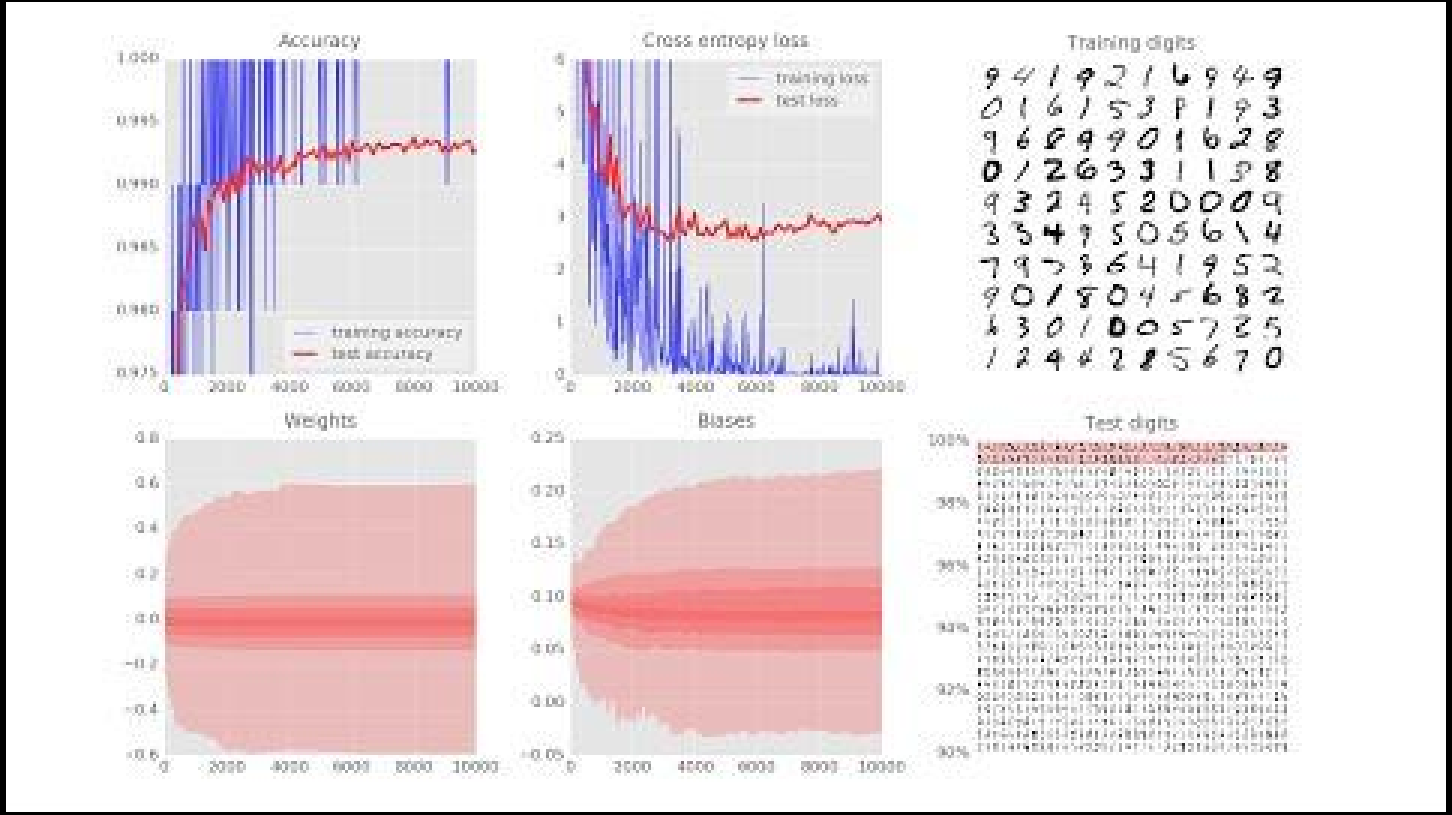
convolutional layer, **6 channels**
W1[**6**, **6**, 1, 6] stride 1

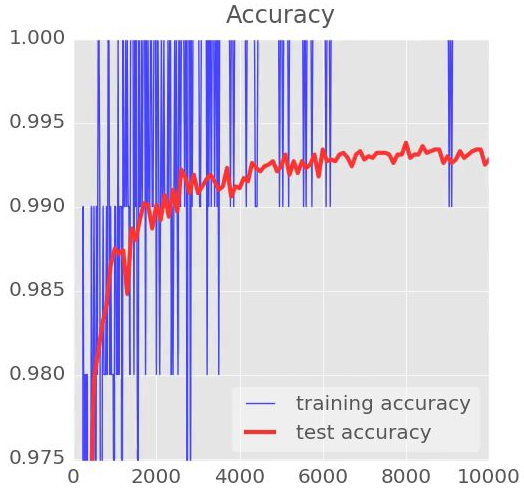
convolutional layer, **12 channels**
W2[**5**, **5**, 6, 12] stride 2

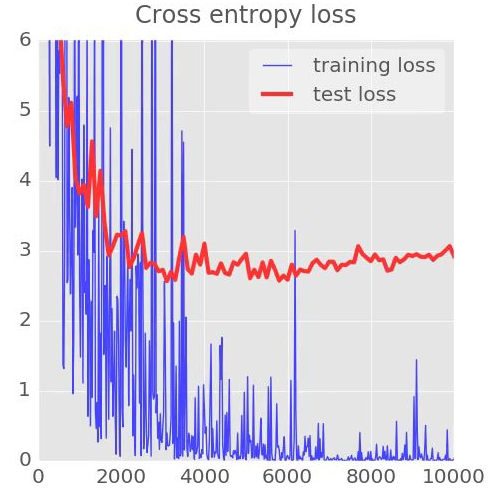
convolutional layer, **24 channels**
W3[**4**, **4**, 12, 24] stride 2

fully connected layer W4[7x7x24, 200]
softmax readout layer W5[200, 10]

Demo

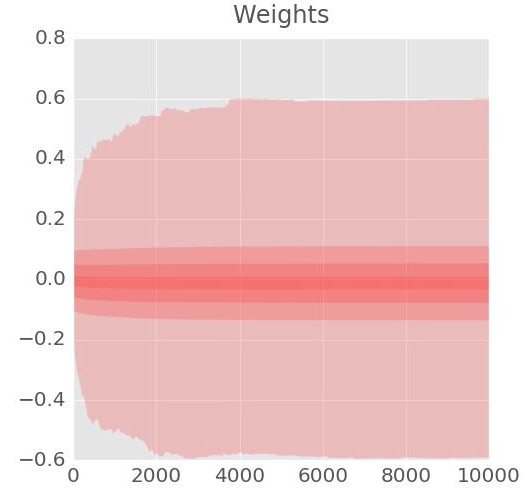

Accuracy
training accuracy
test accuracy
Cross entropy loss
training loss
test loss
Training digits
Weights
Biases
Test digits

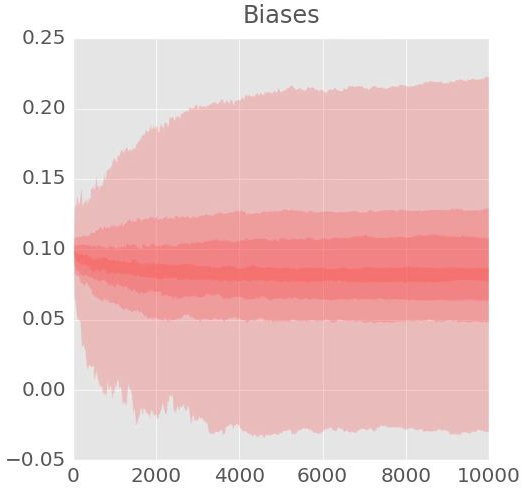

Accuracy
1.000
0.995
0.990
0.985
0.980
0.975
0
2000
4000
6000
8000
10000
training accuracy
test accuracy

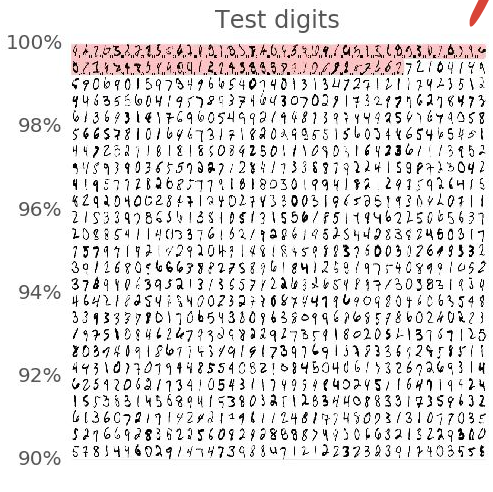

Cross entropy loss
6
5
4
3
2
1
0
2000
4000
6000
8000
10000
training loss
test loss


Training digits

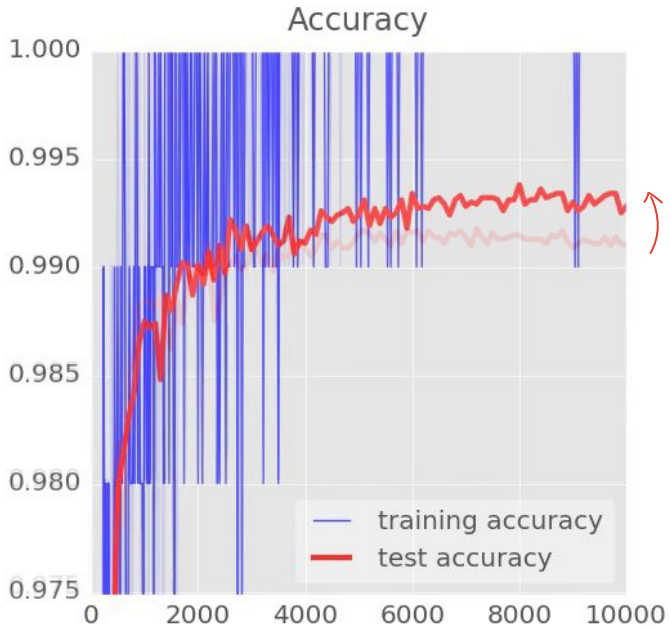
99.3%

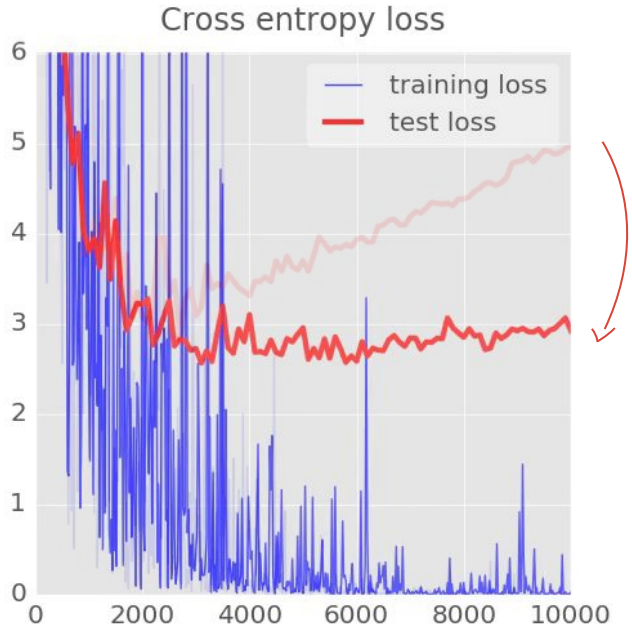

Weights
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
2000
4000
6000
8000
10000


Biases
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
0
2000
4000
6000
8000
10000


Test digits
100%
98%
96%
94%
92%
90%

YEAH !

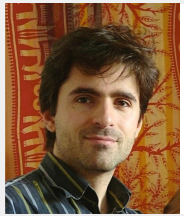




with dropout

Have fun !

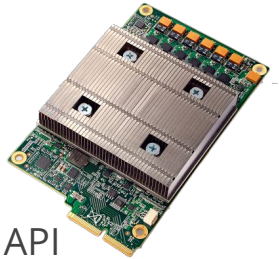
TensorFlow
tensorflow.org



Martin Görner
Google Developer relations
@martin_gorner
plus.google.com/+MartinGorner

Google Cloud Platform - cloud.google.com

Cloud ML ALPHA
your TensorFlow models trained in Google's cloud, fast.



Pre-trained models:

Cloud Vision API

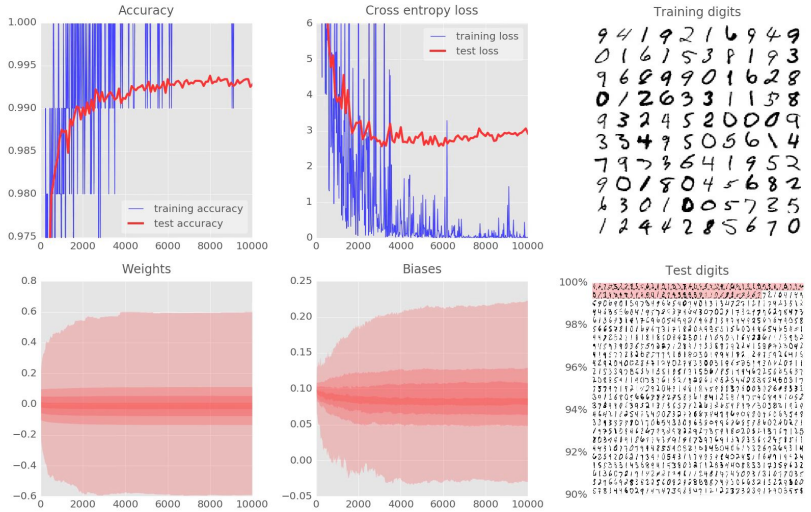
Cloud Speech API ALPHA

Google Translate API

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

This presentation:
goo.gl/pHeXe7

Workshop



Keyboard shortcuts for the visualisation GUI:

1	display 1st graph only
2	display 2nd graph only
3	display 3rd graph only
4	display 4th graph only
5	display 5th graph only
6	display 6th 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
O	box zoom mode (then use mouse)
H	reset all zooms
Ctrl-S	save current image

Workshop

Starter code and solutions:
github.com/martin-gorner/tensorflow-mnist-tutorial

1. Theory (sit back and listen)

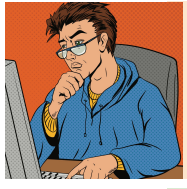
Softmax classifier, mini-batch, cross-entropy and how to implement them in Tensorflow (slides 1-14)

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)



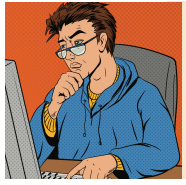
4. Practice

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 = lrmin+(lrmax-lrmin)*exp(-i/2000)`.

Solution in: `mnist_2.1_five_layers_relu_lrdecay.py`

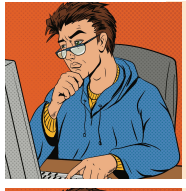
7. Practice (if time allows)

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)

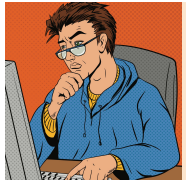
Convolutional networks (slides 36-42)



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`