Fashion-MNIST

Fully Connected Neural Networks

Stephanie & Fanny

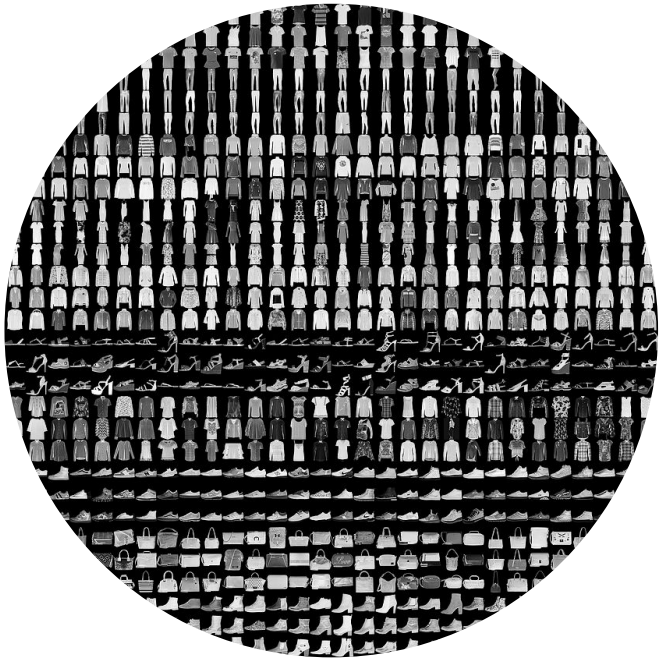
Objectives

1. Brief overview of Fashion-MNIST Data set
2. Fully connected neural network with 1 layer
3. 5 layers with sigmoid
4. 5 layers with relu
5. 5 layers relu & exponential decay
6. 5 layers relu exponential decay & dropout

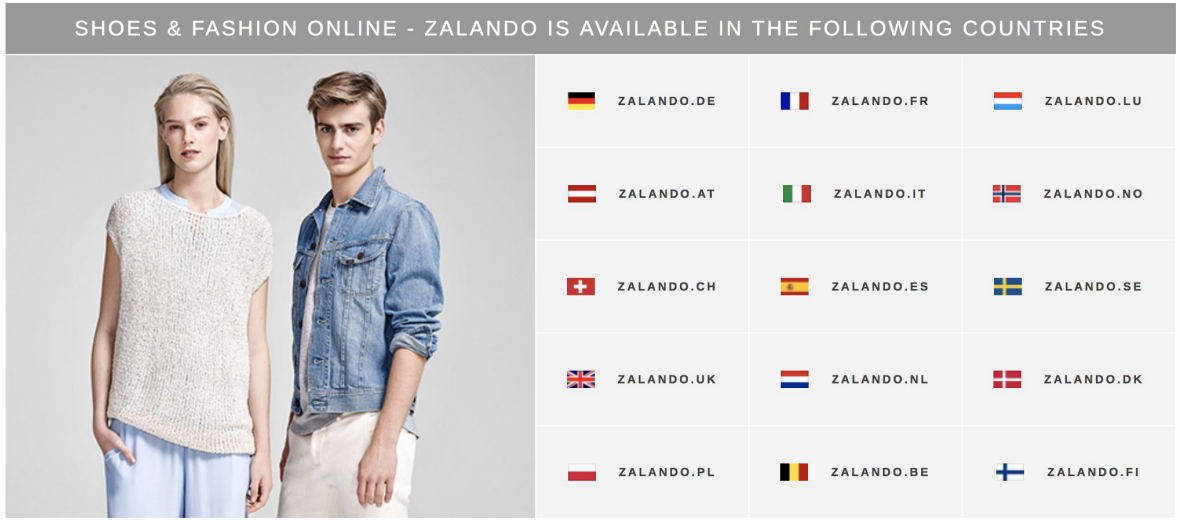
FASHION-MNIST

Zalando Research created:

- 60,000 training images
- 10,000 test images
- 28x28 gray-scale image
  - 784 pixels



zalando



Legal notice

Zalando SE is a German electronic commerce company seated in Berlin. The company maintains a cross-platform online store that sells shoes, clothing and other fashion items.

“overall purpose of wanting to deliver fashion for the good of all.”

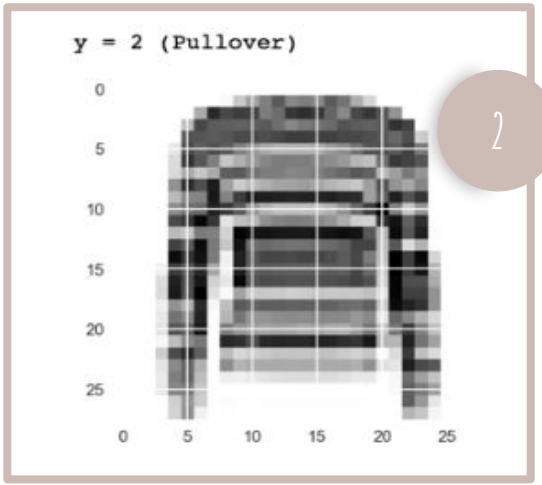
Zalando Launches Research Lab

by Dr. Reiner Kraft - 28 Sep 2016

Zalando Tech is embarking on an exciting new chapter in its evolution. We’ve recently launched Zalando Research – an endeavour to place Zalando at the forefront of cutting-edge research, to complement our already strong foothold on technology.

Why are we doing this? We already boast great researchers at Zalando Tech, so we want to give them an outlet where they can be better organised and aligned. We also want to have a greater impact in the tech research community, especially in the fields of data science, machine learning, and artificial intelligence (AI). We want to carve out

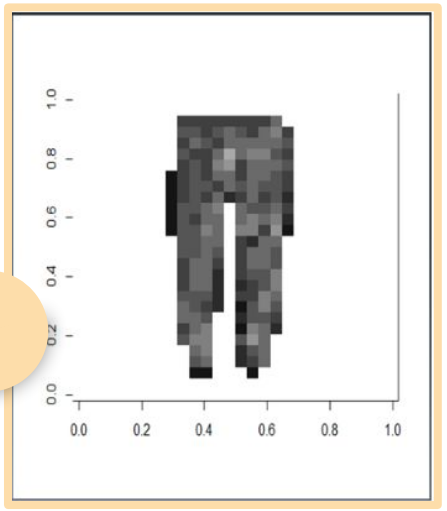
10 DIFFERENT LABELS



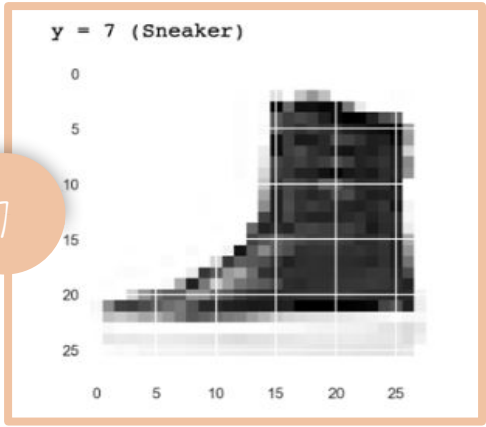
2

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

1



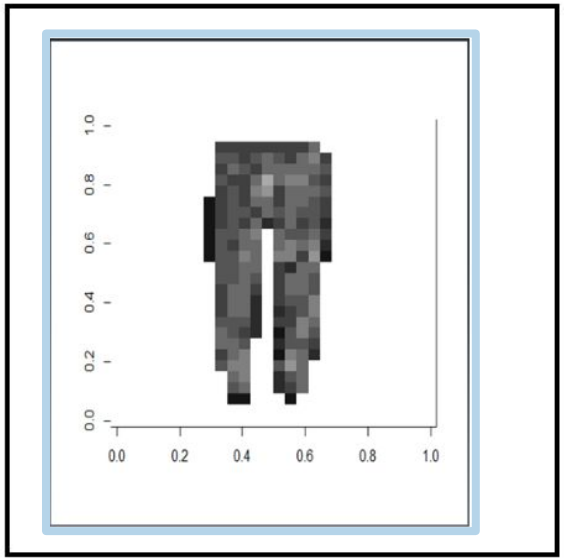
7



1. One layer FULLY-Connected

Let’s learn by explaining the foundations.

Every image can be thought of as an array of numbers describing how dark each pixel is.



≈

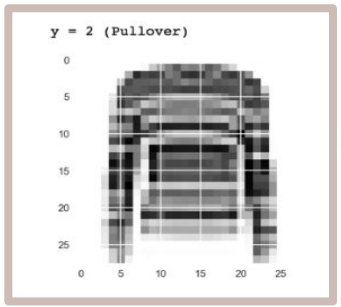
$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0. & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0. & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

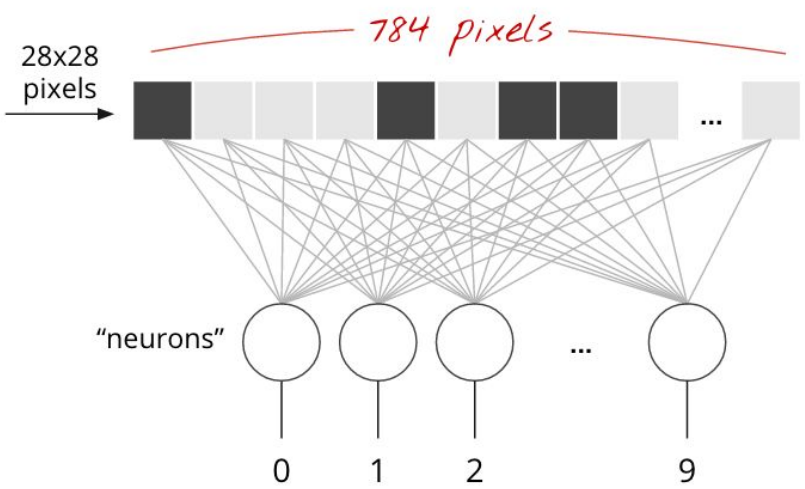
Read-in data

Download images and labels into *mnist.test* and *mnist.train*

mnist = mnist_data.read_data_sets("data/fashion",

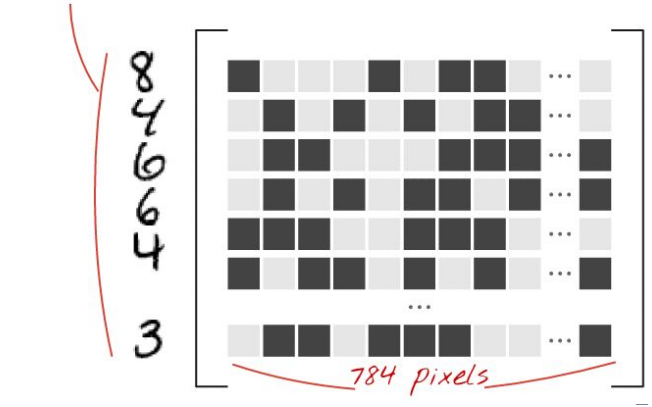
one_hot=True, reshape=False, validation_size=0)





How are weighted sums calculated?!

X: 100 images, one per line, flattened



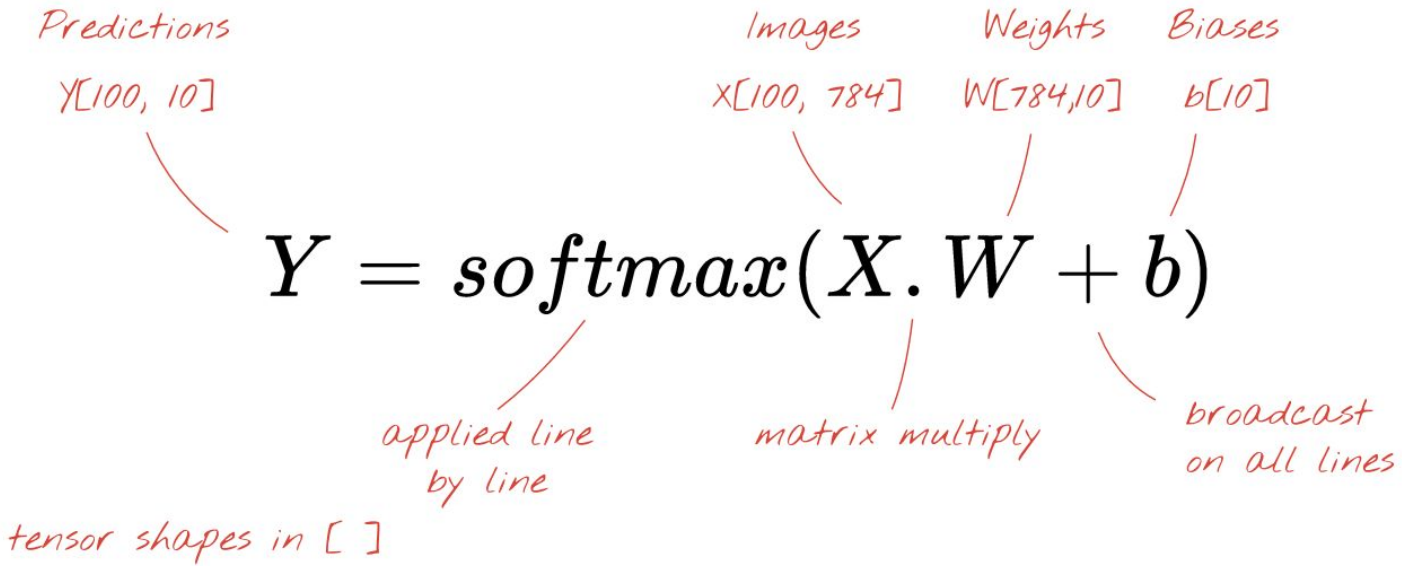
$$\begin{bmatrix} W_{0,0} & W_{0,1} & W_{0,2} & W_{0,3} & \cdots & W_{0,9} \\ W_{1,0} & W_{1,1} & W_{1,2} & W_{1,3} & \cdots & W_{1,9} \\ W_{2,0} & W_{2,1} & W_{2,2} & W_{2,3} & \cdots & W_{2,9} \\ W_{3,0} & W_{3,1} & W_{3,2} & W_{3,3} & \cdots & W_{3,9} \\ W_{4,0} & W_{4,1} & W_{4,2} & W_{4,3} & \cdots & W_{4,9} \\ W_{5,0} & W_{5,1} & W_{5,2} & W_{5,3} & \cdots & W_{5,9} \\ W_{6,0} & W_{6,1} & W_{6,2} & W_{6,3} & \cdots & W_{6,9} \\ W_{7,0} & W_{7,1} & W_{7,2} & W_{7,3} & \cdots & W_{7,9} \\ W_{8,0} & W_{8,1} & W_{8,2} & W_{8,3} & \cdots & W_{8,9} \\ & & \cdots & & & \\ W_{783,0} & W_{783,1} & W_{783,2} & \cdots & & W_{783,9} \end{bmatrix}$$

10 columns

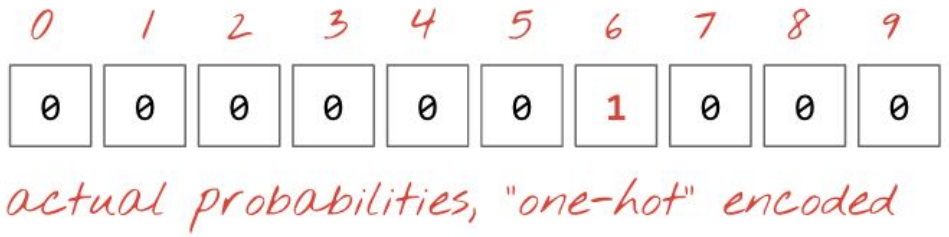
784 lines

broadcast

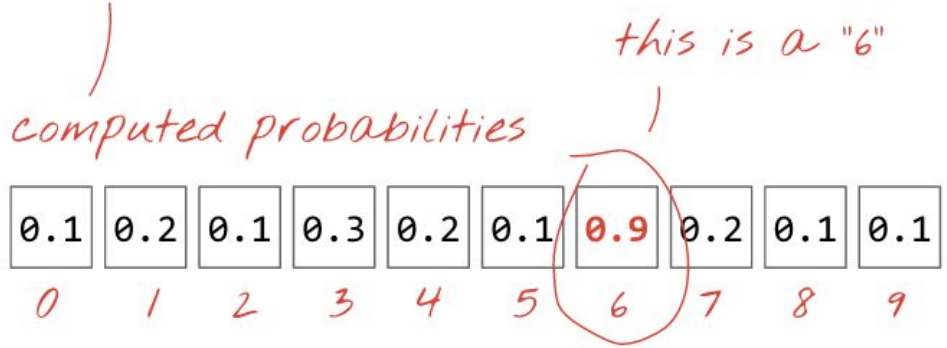
$$L = X.W + b$$




```
# The model
Y = tf.nn.softmax(tf.matmul(XX, W) + b)
```



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



Training

Goal: Find weights and biases that minimize the distance between what system predicts and what we know to be true.

learning rate

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

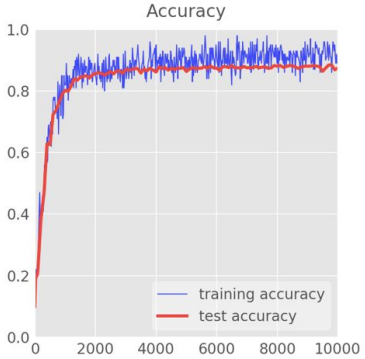
loss function

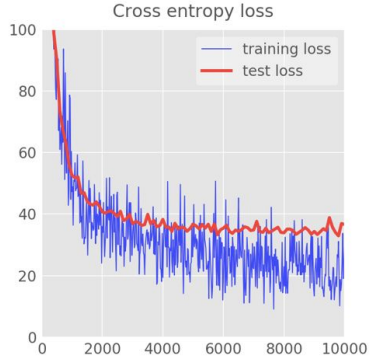
Results

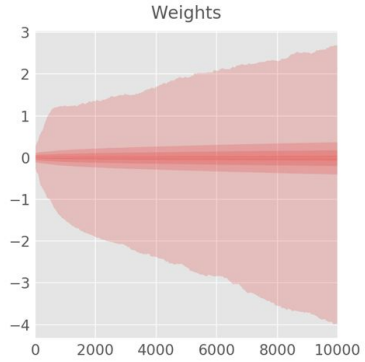
Softmax 1 layer

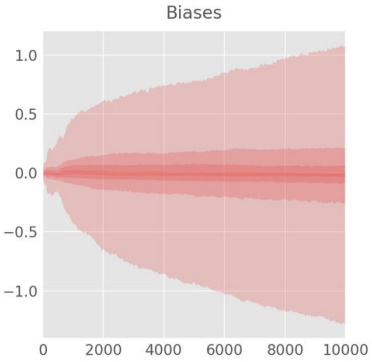
Fashion MNIST
Max test accuracy: 83.13%

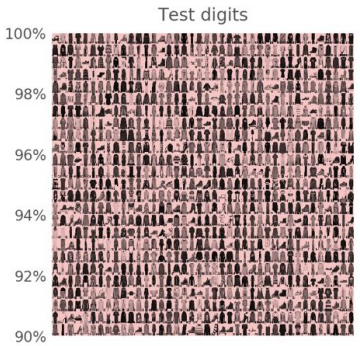
Digits Mnist
accuracy: 92%





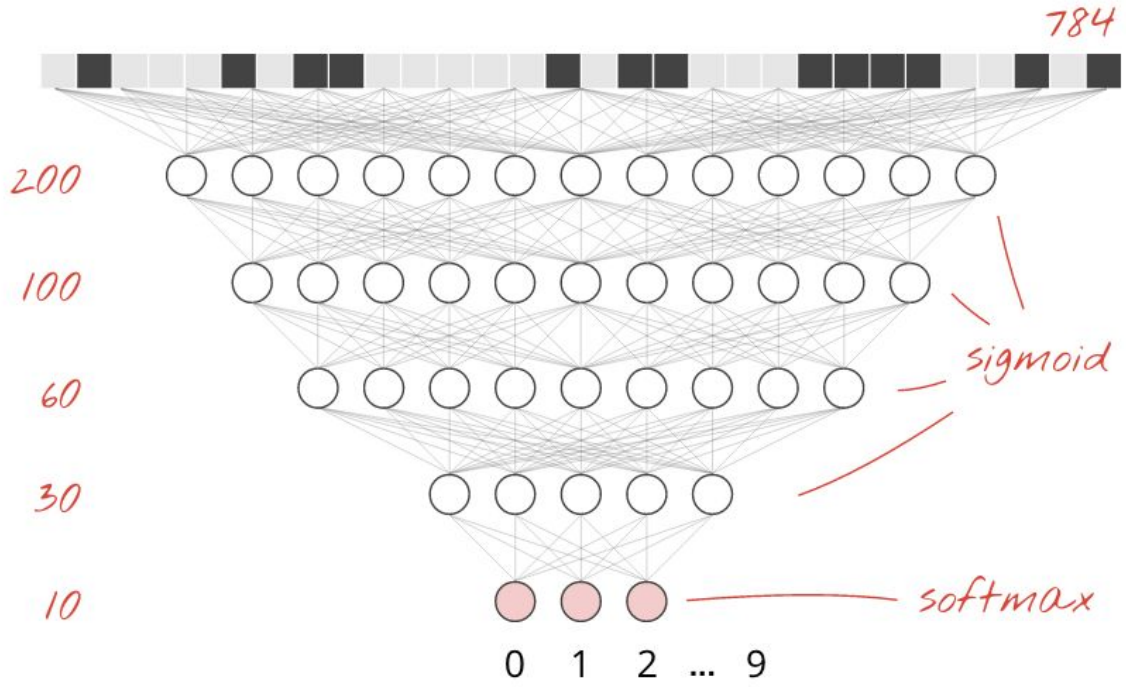







Go deep !

5 LAYERS FULLY-CONNECTED WITH SIGMOID



Here is for example a 5-layer fully connected neural network

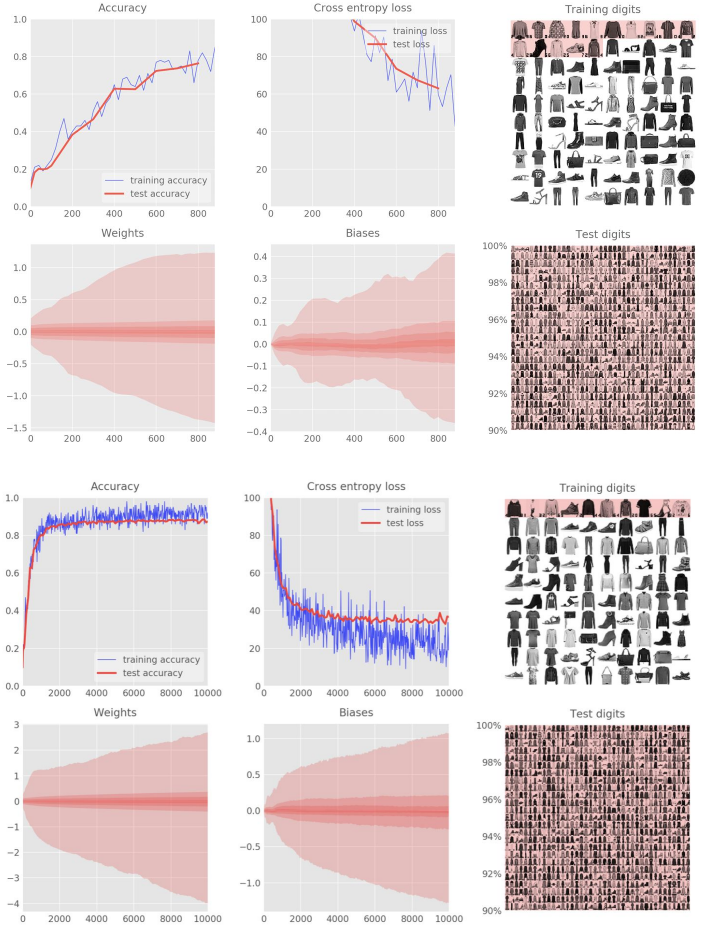
The neurons in the second layer compute weighted sums of neuron outputs from the previous layer.

Pros

- **Max test accuracy: 88.45%**
- Improvement compared to 1 layer softmax max test accuracy: 83.13%
- Adding layers improves accuracy

Cons

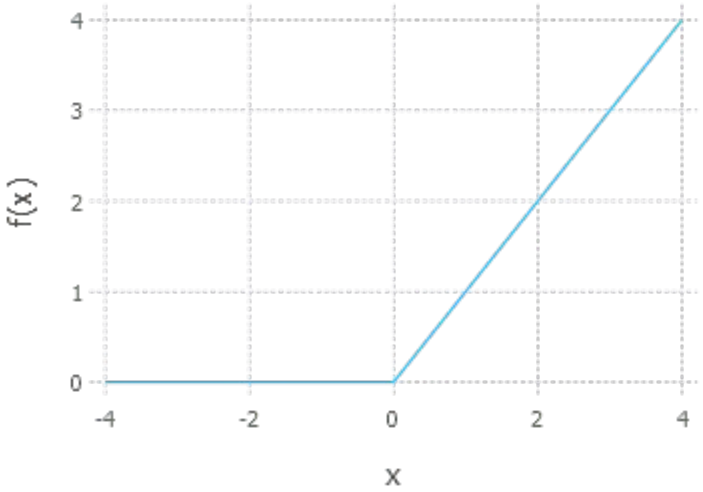
- Sigmoid activation function with many layers in initial iterations accuracy increases slowly



3.5 LAYER FULLY-CONNECTED WITH RELU

Sigmoid does not perform as well with more layers. Let's use the ReLu as our activation function instead.

Replace sigmoid Activation function with ReLu (Rectified linear unit)



X	Y
negative	0
positive	positive

Swap

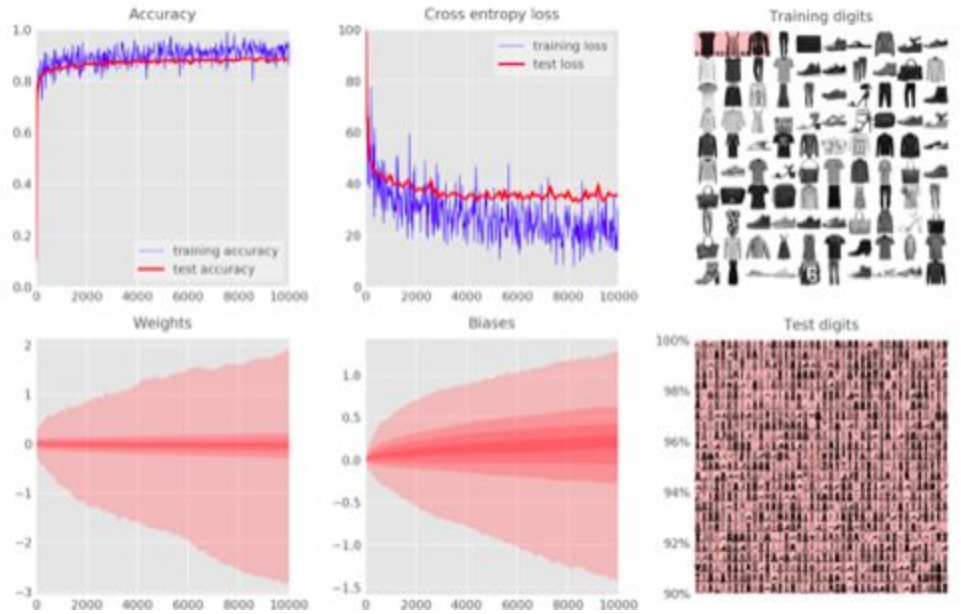
```
# previous model
XX = tf.reshape(X, [-1, 784])
Y1 = tf.nn.sigmoid(tf.matmul(XX, W1) + B1)
Y2 = tf.nn.sigmoid(tf.matmul(Y1, W2) + B2)
Y3 = tf.nn.sigmoid(tf.matmul(Y2, W3) + B3)
Y4 = tf.nn.sigmoid(tf.matmul(Y3, W4) + B4)
Ylogits = tf.matmul(Y4, W5) + B5
Y = tf.nn.softmax(Ylogits)
```

```
# new model
XX = tf.reshape(X, [-1, 784])
Y1 = tf.nn.relu(tf.matmul(XX, W1) + B1)
Y2 = tf.nn.relu(tf.matmul(Y1, W2) + B2)
Y3 = tf.nn.relu(tf.matmul(Y2, W3) + B3)
Y4 = tf.nn.relu(tf.matmul(Y3, W4) + B4)
Ylogits = tf.matmul(Y4, W5) + B5
Y = tf.nn.softmax(Ylogits)
```

5 layer fully-connected with relu Results

	Sigmoid	ReLu
Max test accuracy	0.8813	0.8893

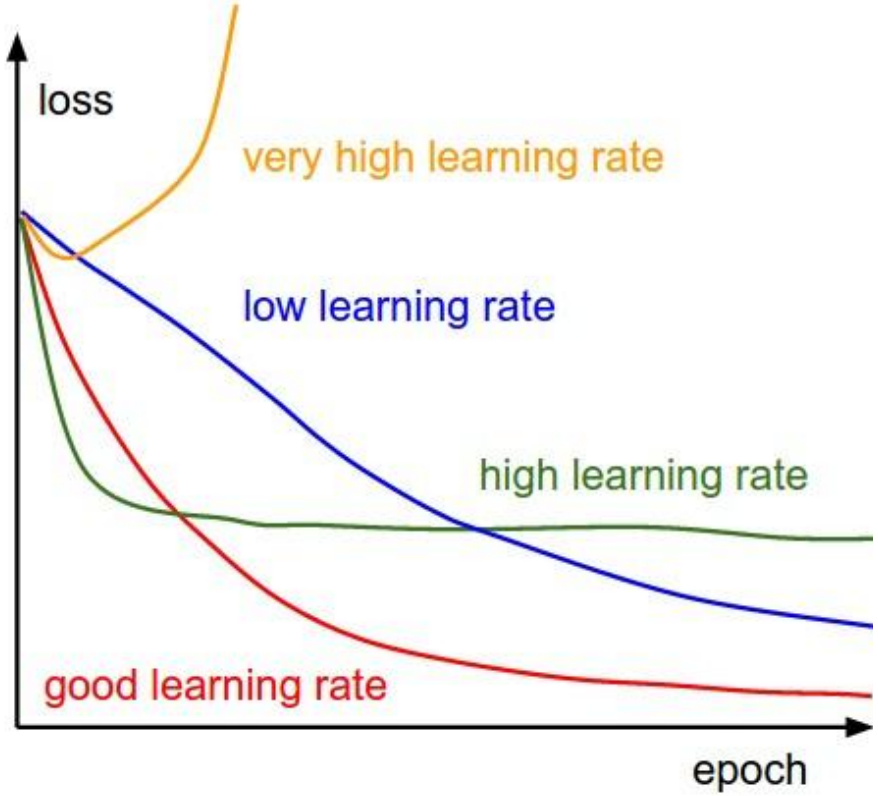
Cons	Pros
Noisy accuracy curve	Faster training
Noisy cross entropy loss	Improved accuracy

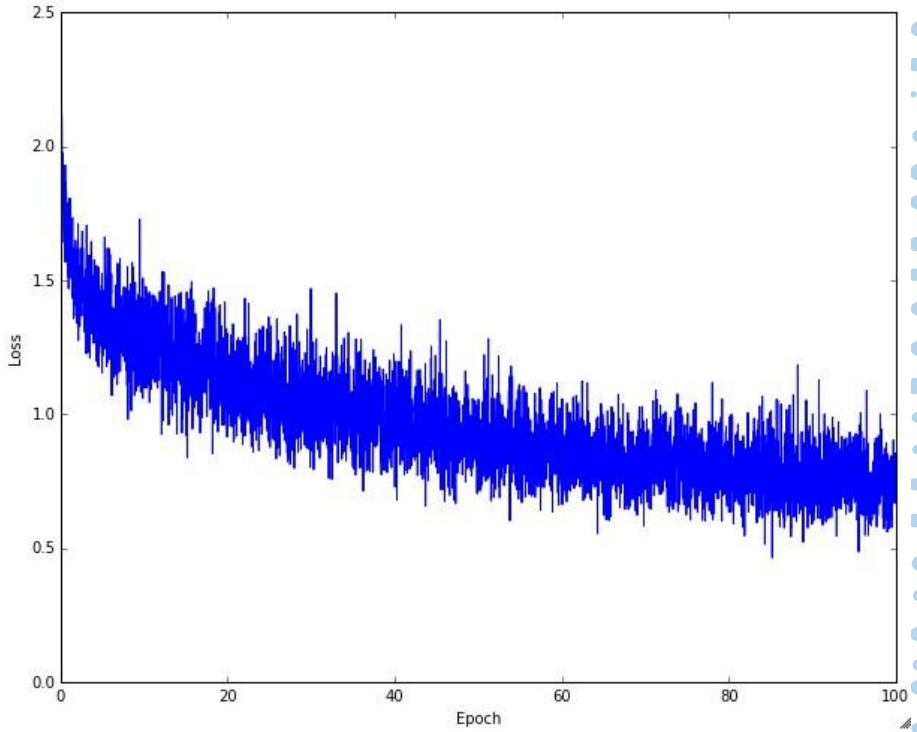


4. 5 layer fully-connected with ReLU & decay learning rate

What happens when you add a decay learning rate?

Effects of different learning rates





Changing LEARNING RATE

CONSTANT LEARNING RATE

```
learning_rate = 0.003
train_step =
tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)
```

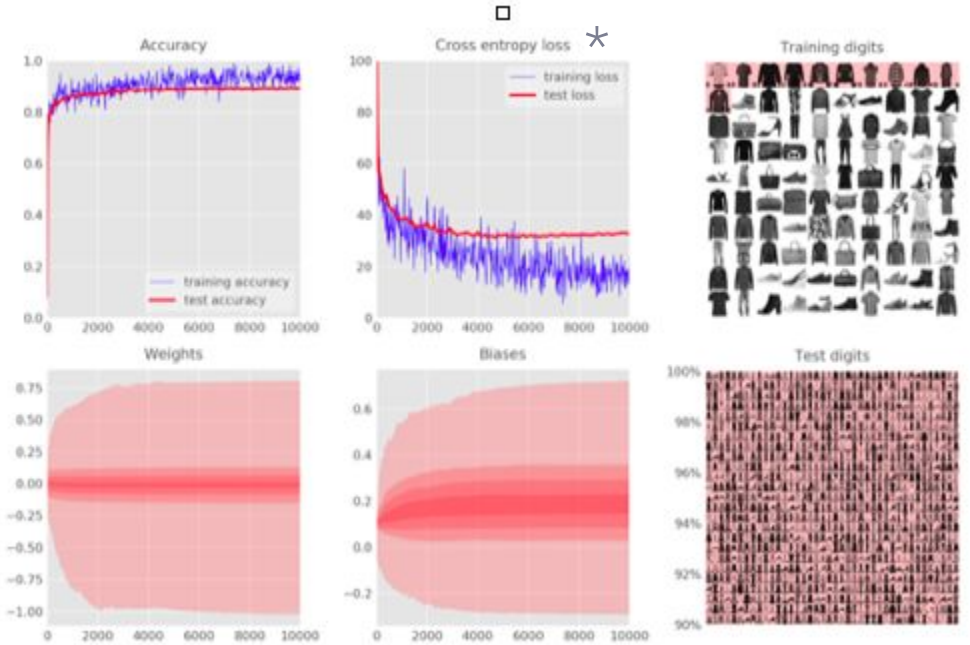
DECAY LEARNING RATE

```
# the learning rate is:  0.0001 + 0.003 * (1/e)^(step/2000))
# i.e. exponential decay from 0.003, then dropping exponentially  to 0.0001
lr = 0.0001 +  tf.train.exponential_decay(0.003, step, 2000, 1/math.e)
train_step = tf.train.AdamOptimizer(lr).minimize(cross_entropy)
```

Fully-connected neural network: relu decay learning Rate

	Without decay learning rate	Decay learning rate
Max test accuracy	0.8893	0.8949

Cons	Pros
Cross-entropy loss function shifting up	Faster training
Risk of overfitting*	Improved accuracy



5. 5 layer fully-connected with Relu + decay learning + rate + dropout

What happens when you shoot 25% of your neurons?

What is dropout?

- Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random.
- These units are not considered during a particular forward or backward pass.

- At each training stage, nodes are:
  - dropped with probability = 1 - p
  - Kept with probability = p
-

Why dropout?

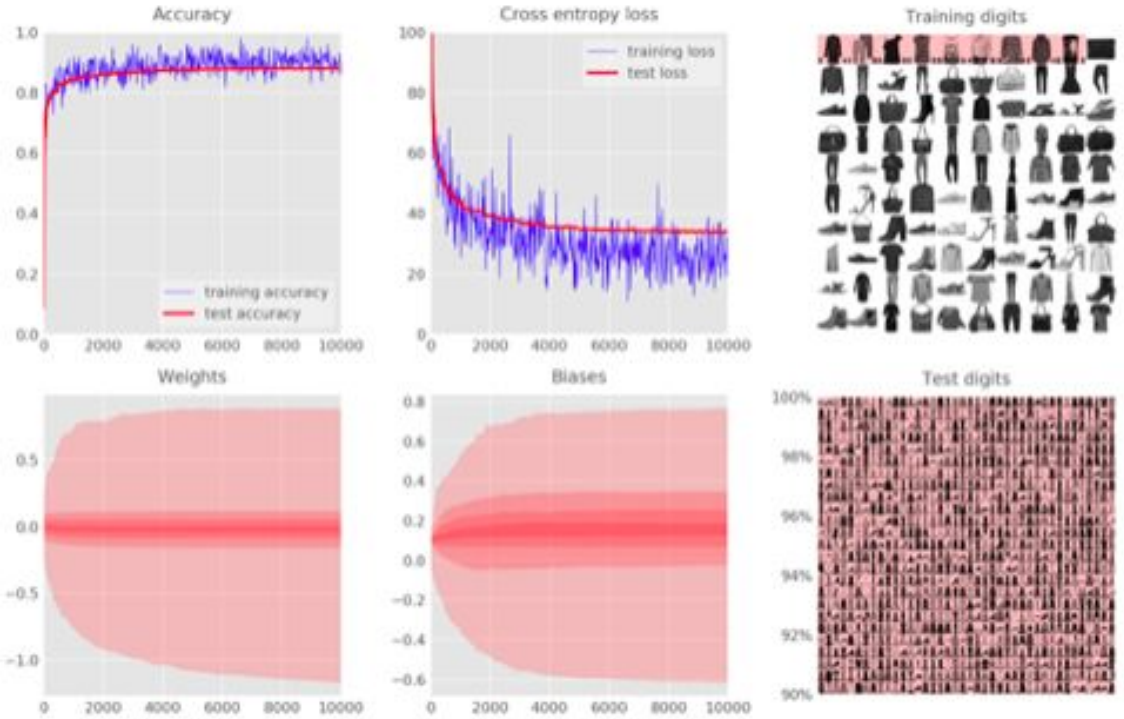
"To prevent overfitting"

Code the dropout

```
# the previous model
XX = tf.reshape(X, [-1, 784])
Y1 = tf.nn.relu(tf.matmul(XX, W1) + B1)
Y2 = tf.nn.relu(tf.matmul(Y1, W2) + B2)
Y3 = tf.nn.relu(tf.matmul(Y2, W3) + B3)
Y4 = tf.nn.relu(tf.matmul(Y3, W4) + B4)
Ylogits = tf.matmul(Y4, W5) + B5
Y = tf.nn.softmax(Ylogits)
```

```
# The model, with dropout at each layer
XX = tf.reshape(X, [-1, 28*28])

Y1 = tf.nn.relu(tf.matmul(XX, W1) + B1)
Y1d = tf.nn.dropout(Y1, pkeep)

Y2 = tf.nn.relu(tf.matmul(Y1d, W2) + B2)
Y2d = tf.nn.dropout(Y2, pkeep)

Y3 = tf.nn.relu(tf.matmul(Y2d, W3) + B3)
Y3d = tf.nn.dropout(Y3, pkeep)

Y4 = tf.nn.relu(tf.matmul(Y3d, W4) + B4)
Y4d = tf.nn.dropout(Y4, pkeep)
Ylogits = tf.matmul(Y4d, W5) + B5
Y = tf.nn.softmax(Ylogits)
```

Fully-connected neural network with relu + decay learning rate + dropout



max test accuracy: 0.882

Question

What do you think will happen when we change the learning rate and the number of layers?

Comparing accuracy rates for 5 layer relu w/ dropout

	Number of Layers	Number of Layers	Number of Layers
Exponential Decay	3	5	7
From .03 to .0001	0.814	0.7706	.1116
Default From .003 to 0.0001	0.8832	0.882	0.8886
From .0003 to .0001	0.8692	0.8731	0.8891

“

88.9% test accuracy at most?!

We can do better than that!

CNNs

What happens if we maintain the 2-D structure of images?

Next time, we'll discuss Convolutional Neural Networks!

Part 2:
Convolutional Neural Networks

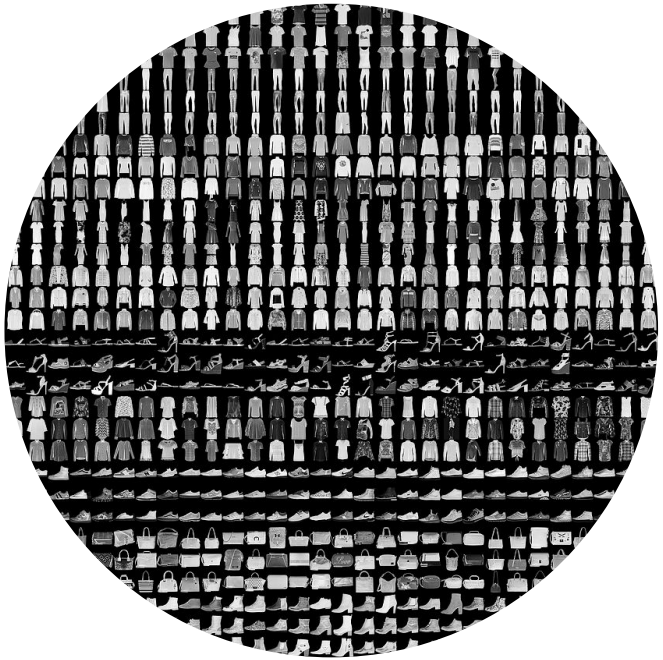
Objectives

1. Brief review of Fashion-MNIST data set
2. History of CNNs
3. Add layers! More degrees of freedom!
4. Add dropout (regularization)
    a. P = 0.75
    b. Change that up!
5. CNN with Batch Normalization

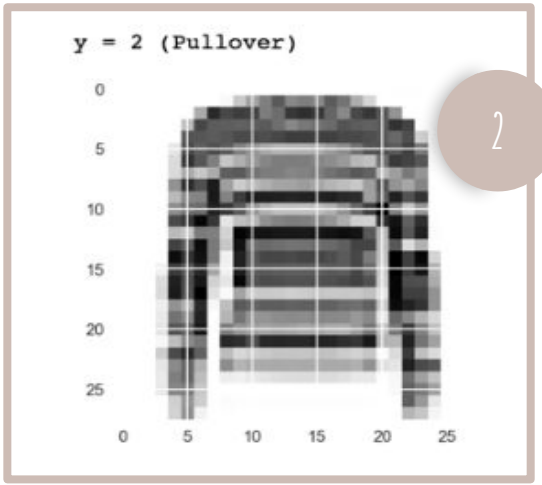
FASHION-MNIST

Zalando Research created:

- 60,000 training images
- 10,000 test images
- 28x28 gray-scale image
  - 784 pixels



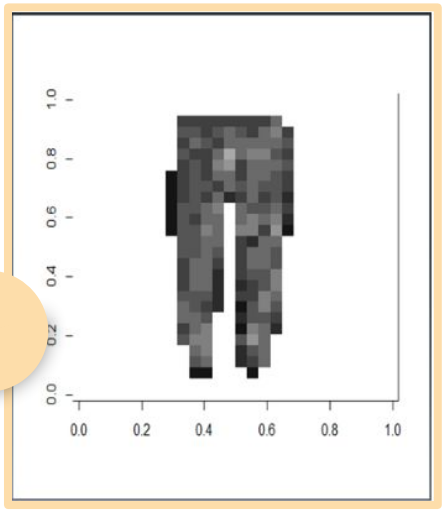
10 DIFFERENT LABELS



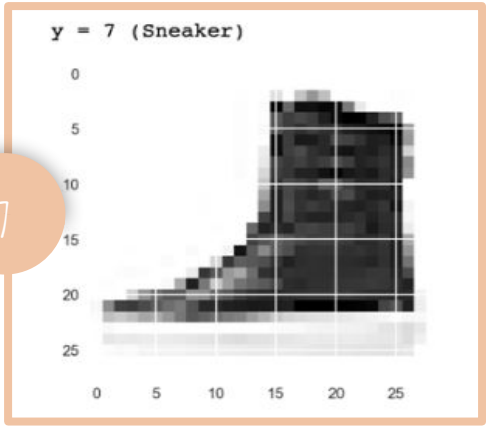
2

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

1

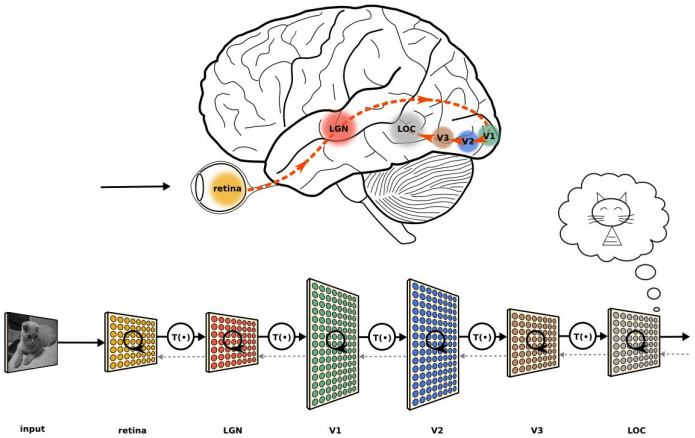


7



2. History Of Convolutional Neural Networks

Biological background



Research paper written by D.H. Hubel and T.N. Wiesel in 1962 on mammalian visual cortex.

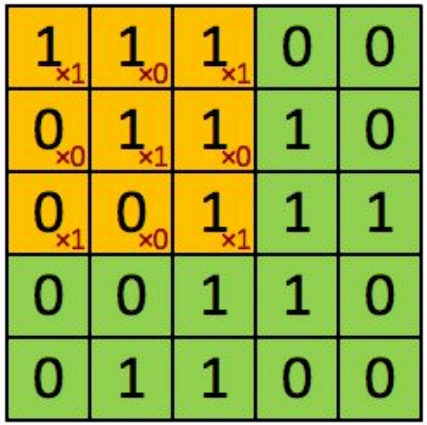
They proposed an explanation for the way in which mammals visually perceive the world around them using a layered architecture of neurons in the brain.

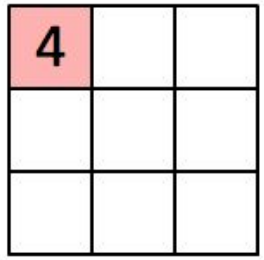
This inspired engineers to attempt to develop similar pattern recognition mechanisms in computer vision.

3. Convolutional neural networks

Let’s utilize the 2-D structure of images.

Convolution

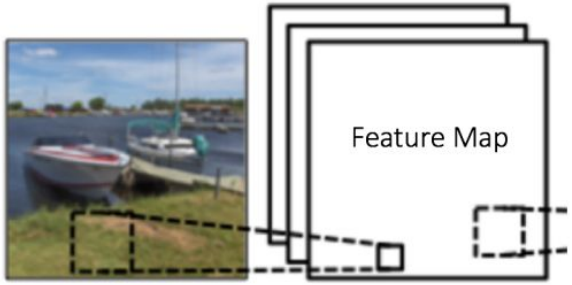




- Preserves the spatial relationship between pixels
- Learn features using small squares of input data

4. Add Layers- Degrees of Freedom

Add Layers



Feature Map having depth of 3 (since 3 filters have been used)

Convolution Operation

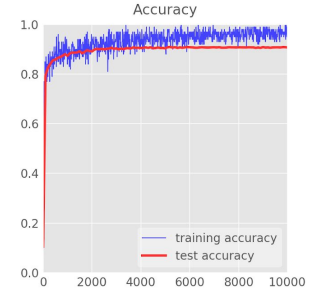
- **Depth** corresponds to the number of filters we use for the convolution operation.
- **Stride** is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time.

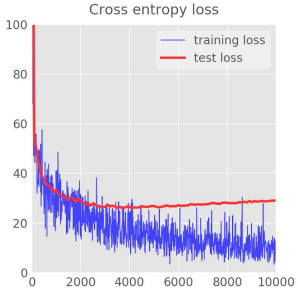
The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

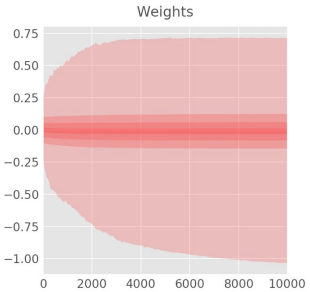
3 Convolutional Layers 2 fully connected

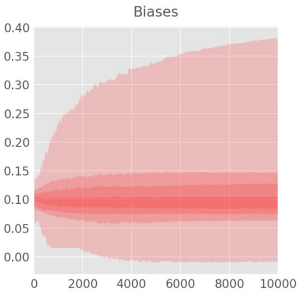
```
# The model
stride = 1  # output is 28x28
Y1 = tf.nn.relu(tf.nn.conv2d(X, W1, strides=[1, stride, stride, 1], padding='SAME') + B1)
stride = 2  # output is 14x14
Y2 = tf.nn.relu(tf.nn.conv2d(Y1, W2, strides=[1, stride, stride, 1], padding='SAME') + B2)
stride = 2  # output is 7x7
Y3 = tf.nn.relu(tf.nn.conv2d(Y2, W3, strides=[1, stride, stride, 1], padding='SAME') + B3)

# reshape the output from the third convolution for the fully connected layer
YY = tf.reshape(Y3, shape=[-1, 7 * 7 * M])

Y4 = tf.nn.relu(tf.matmul(YY, W4) + B4)
Ylogits = tf.matmul(Y4, W5) + B5
Y = tf.nn.softmax(Ylogits)
```

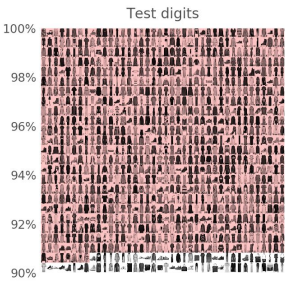
Results







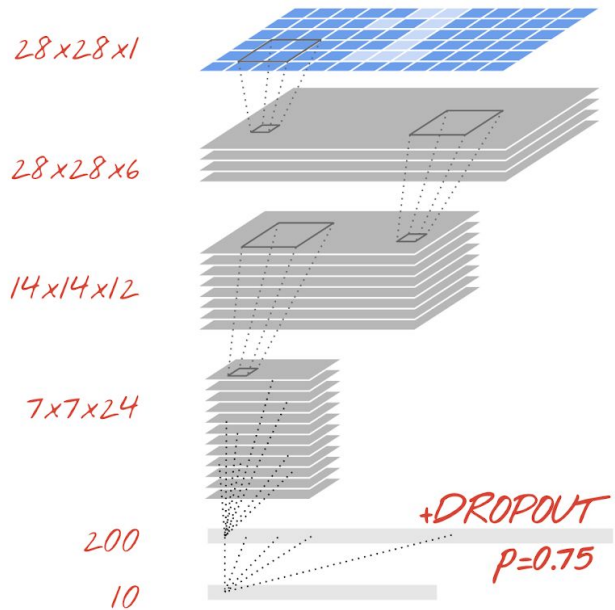




Model	Max Accuracy
Fully Connected 7 Layers + Relu+Dropout	88.91%
Convolutional Network	**90.91%**

5. Convolutional neural networks with Dropout

Fix Cross Entropy Loss



+ 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

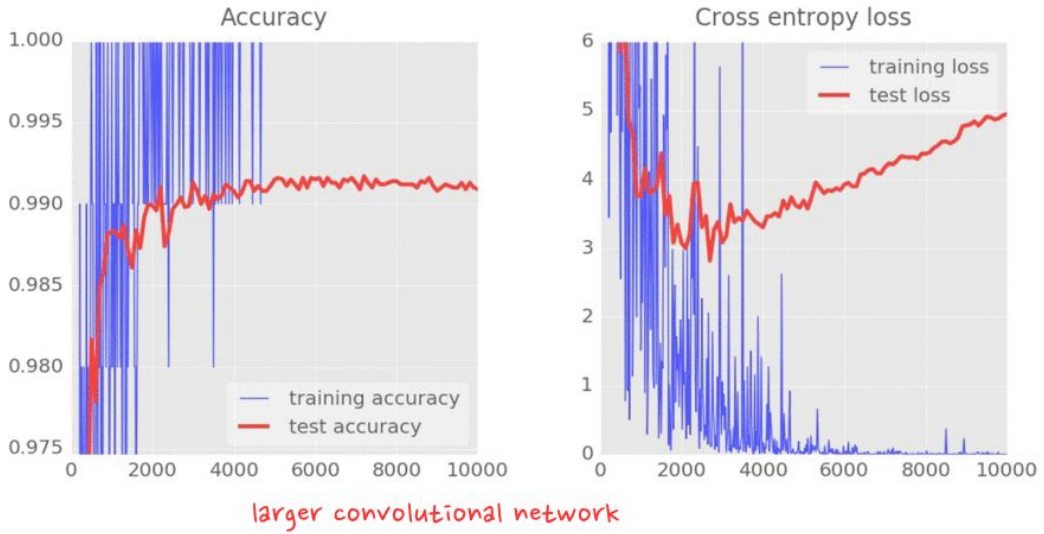
convolutional layer, 24 channels
W3[4, 4, 12, 24] stride 2

fully connected layer
W4[7x7x24, 200]

softmax readout layer
W5[200, 10]

Increase number of channels to increase degrees of freedom but also add dropout on fully connected layer so increase in degrees of freedom doesn't result in overfitting.

Results

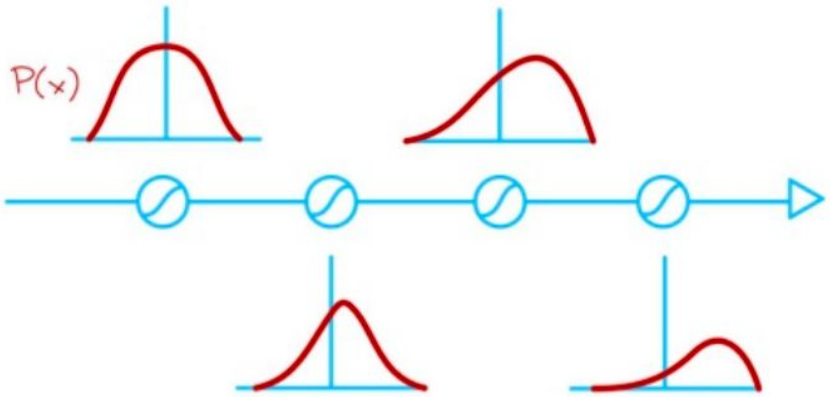


Model	Max Accuracy
Convolutional Network	90.91%
Convolutional Network + Dropout	**91.16%**

6. Batch normalization

Speed up training!

Internal Covariate shift

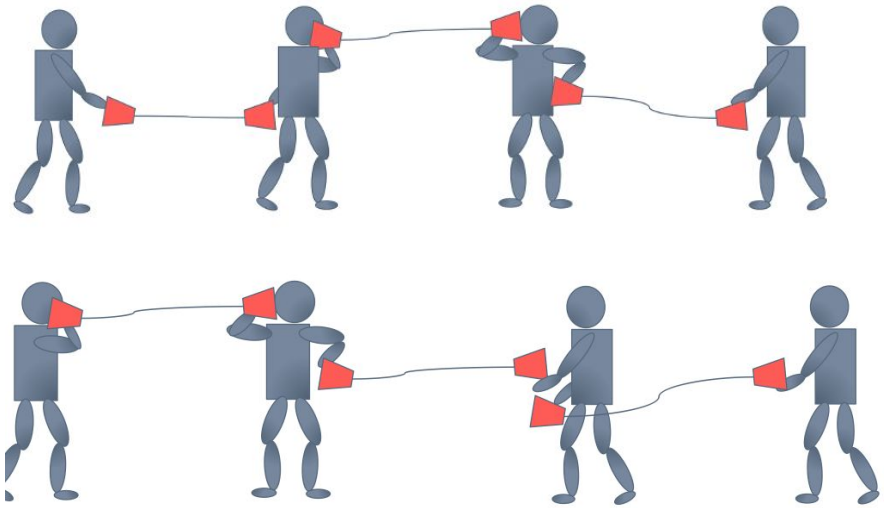


As learning progresses, the distribution of layer inputs changes due to parameter updates.

The weights in our model become updated over each epoch during training.

What if during training one of the weights ends up becoming drastically larger than the other weights?

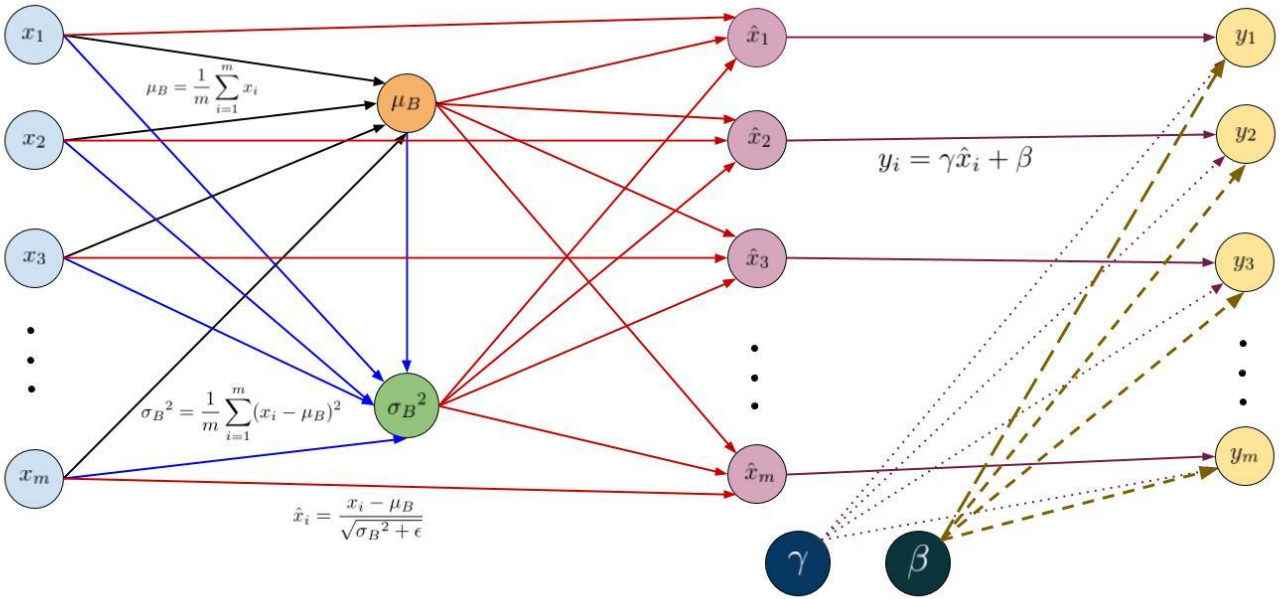
Problem: Internal Covariate shift



Well this large weight will cause the output from its corresponding neuron to be extremely large.

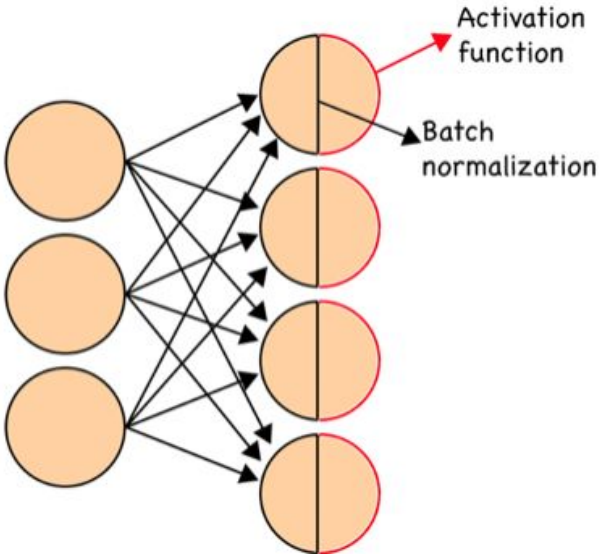
The imbalance will cascade through the neural network causing instability that slows down convergence.

Solution: Batch normalization



1. Compute average & variance on mini-batch

Batch normalization



$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

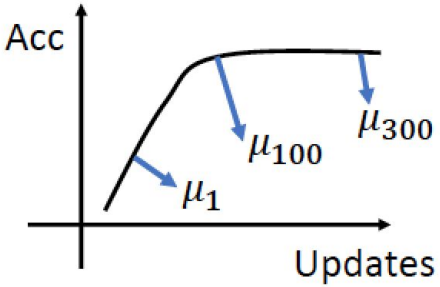
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

2. Center & rescale logits before the activation function
3. Add learnable scale & offset

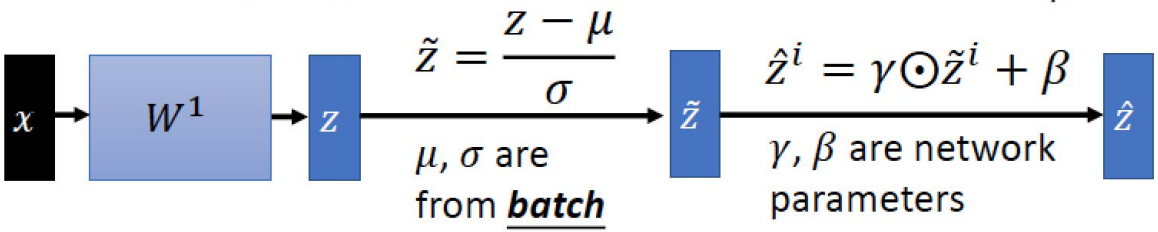
Why use Batch normalization?

- Helps maximize ability to take advantage of activation function
- Makes training faster
- Unbalanced non-normalized data can make it harder to train data
- Regularization is a side-benefit!

Batch normalization



- At testing stage:



We do not have ***batch*** at testing stage.

Ideal solution:

Computing μ and σ using the whole training dataset.

Practical solution:

Computing the moving average of μ and σ of the batches during training.

Code: batch normalization

```
def batchnorm(Ylogits, is_test, iteration, offset, convolutional=False):
    # adding the iteration prevents from averaging across non-existing iterations
    exp_moving_avg = tf.train.ExponentialMovingAverage(0.999, iteration)
    bnepsilon = 1e-5
    if convolutional:
        mean, variance = tf.nn.moments(Ylogits, [0, 1, 2])
    else:
        mean, variance = tf.nn.moments(Ylogits, [0])
    update_moving_averages = exp_moving_avg.apply([mean, variance])
    m = tf.cond(is_test, lambda: exp_moving_avg.average(mean), lambda: mean)
    v = tf.cond(is_test, lambda: exp_moving_avg.average(variance), lambda: variance)
    Ybn = tf.nn.batch_normalization(Ylogits, m, v, offset, None, bnepsilon)
    return Ybn, update_moving_averages
```

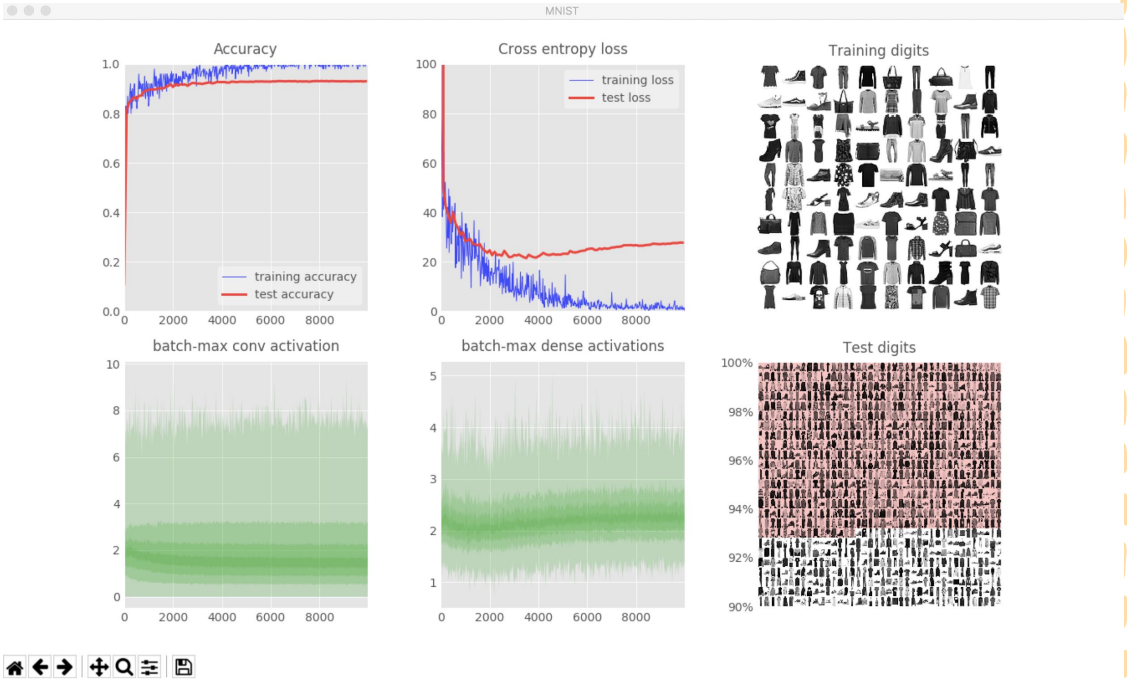
BATCH NORMALIZATION MODEL

```
# batch norm offsets are used instead of biases
stride = 1  # output is 28x28

Y1l = tf.nn.conv2d(X, W1, strides=[1, stride, stride, 1], padding='SAME')
Y1bn, update_ema1 = batchnorm(Y1l, tst, iter, B1, convolutional=True)
Y1r = tf.nn.relu(Y1bn)
Y1 = tf.nn.dropout(Y1r, pkeep_conv, compatible_convolutional_noise_shape(Y1r))

stride = 2  # output is 14x14
Y2l = tf.nn.conv2d(Y1, W2, strides=[1, stride, stride, 1], padding='SAME')
Y2bn, update_ema2 = batchnorm(Y2l, tst, iter, B2, convolutional=True)
Y2r = tf.nn.relu(Y2bn)
Y2 = tf.nn.dropout(Y2r, pkeep_conv, compatible_convolutional_noise_shape(Y2r))

stride = 2  # output is 7x7
Y3l = tf.nn.conv2d(Y2, W3, strides=[1, stride, stride, 1], padding='SAME')
Y3bn, update_ema3 = batchnorm(Y3l, tst, iter, B3, convolutional=True)
Y3r = tf.nn.relu(Y3bn)
Y3 = tf.nn.dropout(Y3r, pkeep_conv, compatible_convolutional_noise_shape(Y3r))
```

5 layer CNN with BATCH NORMALIZATION Results

	Dropout	Batch Normalization
Max test accuracy	91.16%	**93.14%**

Cons	Pros
Noisy accuracy curve	Faster training
Noisy cross entropy loss	Improved accuracy



Comparing accuracy rates of models we tested

No Dropout	**Dropout (p_keep)**	**No Batch Normalization**	**Batch Normalization**
	0.6		92% Entropy loss improved
	0.75	91.16%	93.14%
90.91%			

How do our models compare to fashion-mnist benchmarks?

Model	Accuracy Rate	Author
CNN w/ Batch Norm	93.14%	Stephanie & Fanny
Xgboost	95.3%	@anktplwl91
Resnet18	94.9%	Kyriakos Efthymiadis
Alexnet with triple loss	89.99%	@Cenk Bircanoğlu

THANKS! QUESTIONS?

SOURCES

- https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners
- http://colah.github.io/posts/2014-10-Visualizing-MNIST/
- https://www.youtube.com/watch?v=qyvlt7kiQoI&feature=youtu.be
- https://cloud.google.com/blog/big-data/2017/01/learn-tensorflow-and-deep-learning-without-a-phd