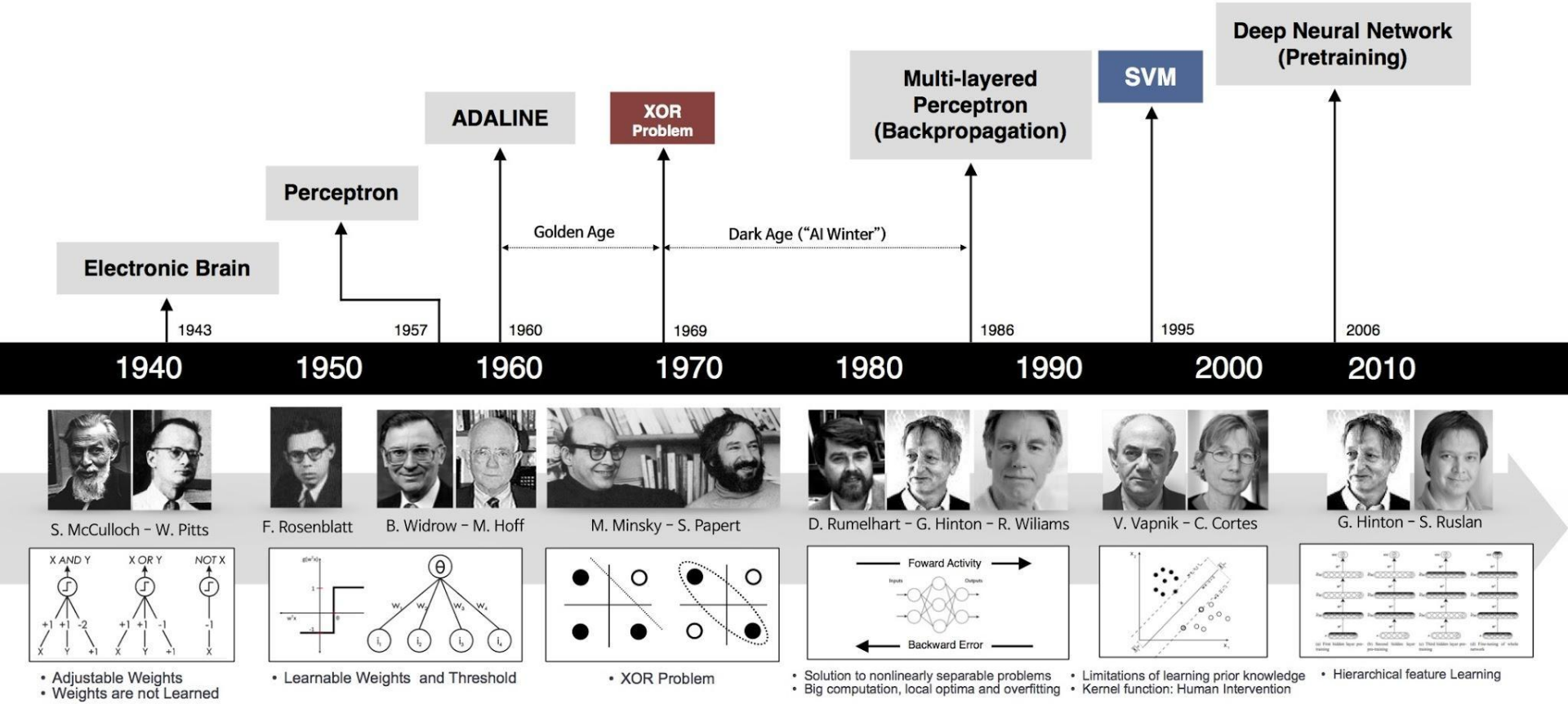
Нейронные сети

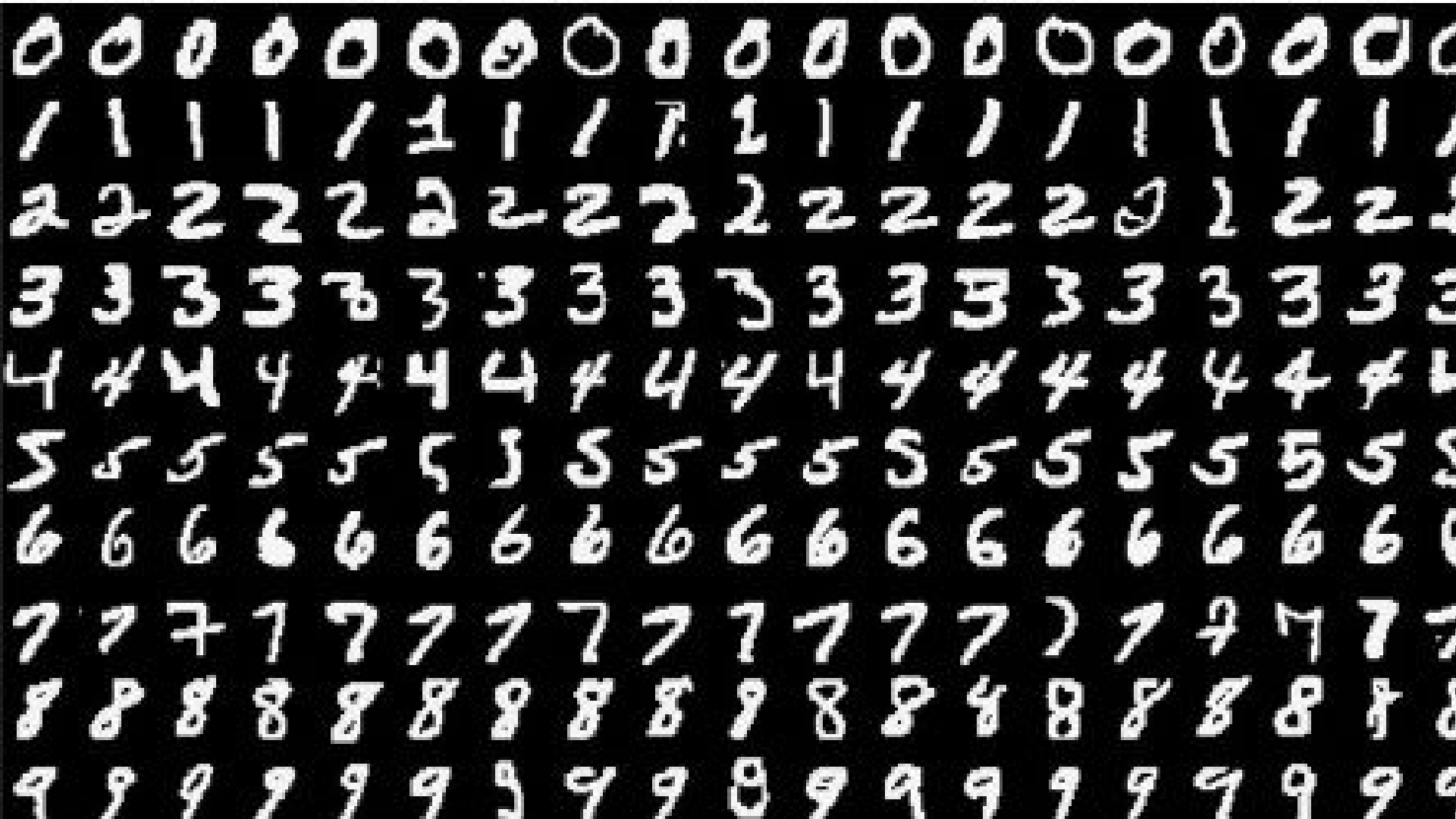
Лекция 7

История

1. 1940-ые – Идея перцептрона и искусственного мозга
2. 1950-ые – Мат. модель перцептрона
3. 1950-ые – Adaline модель
4. 1969 – Озвучена XOR проблема
5. 1986 – Многослойная модель, Backpropagation method
6. 1988 – Сверточные нейронные сети
7. 1990-ые – Рекуррентные нейронные сети
8. 2006 – Глубокие нейросети
9. 2009 – GPU для нейросетей
10. 2010-ые Image Net эпопея
11. ...



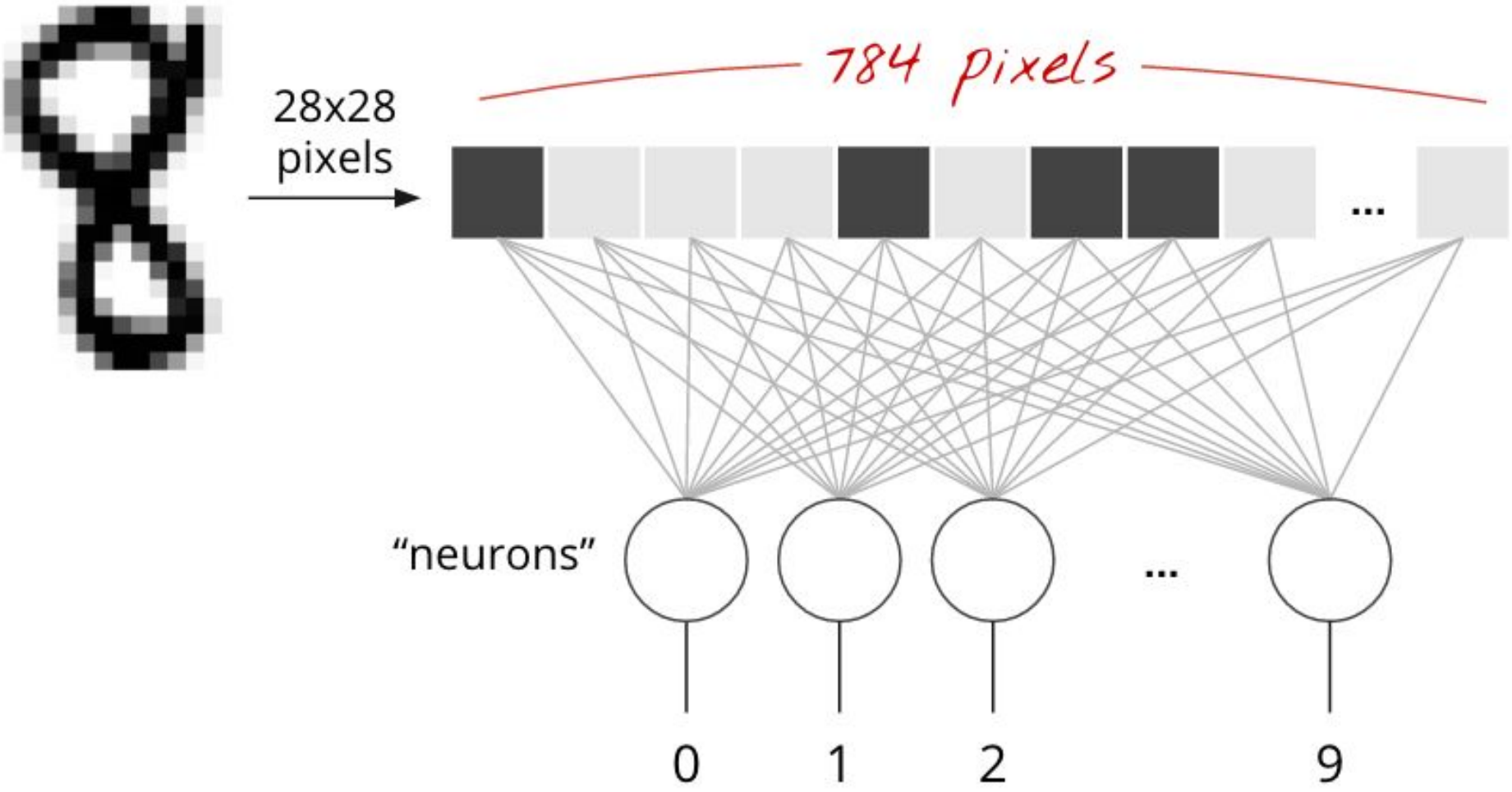
http://qingkaikong.blogspot.ru/2016/11/machine-learning-3-artificial-neural.html

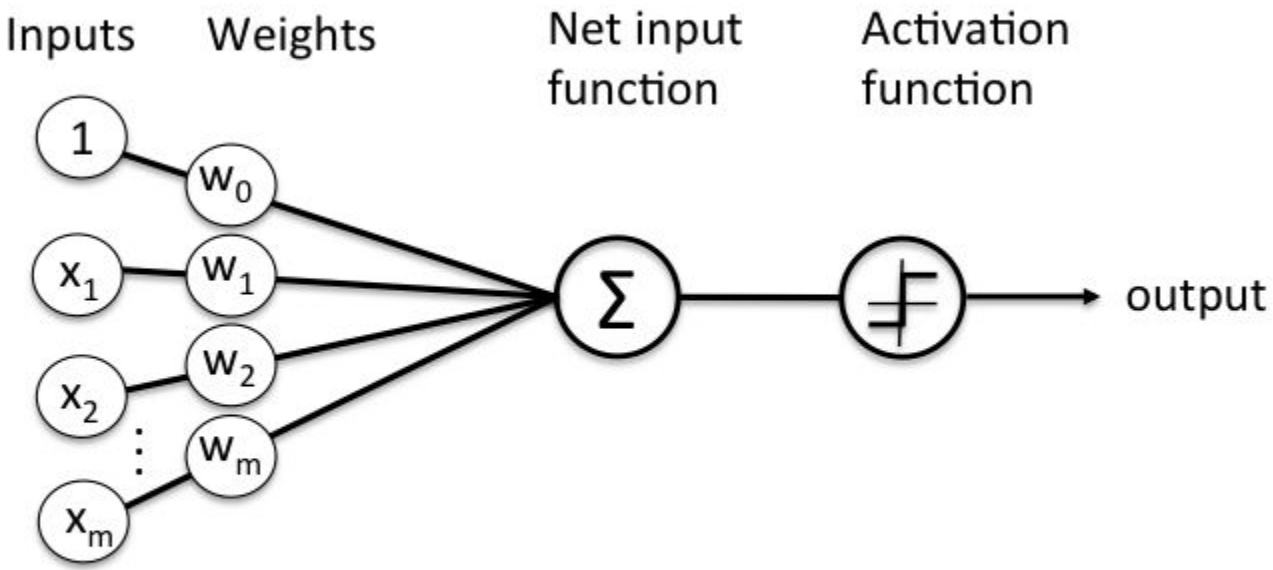


Задача MNIST

- 10 классов (0..9)
- 28x28 пикселя
- 60_000 тренировочная
- 10_000 тестовая

источник: http://yann.lecun.com/exdb/mnist/

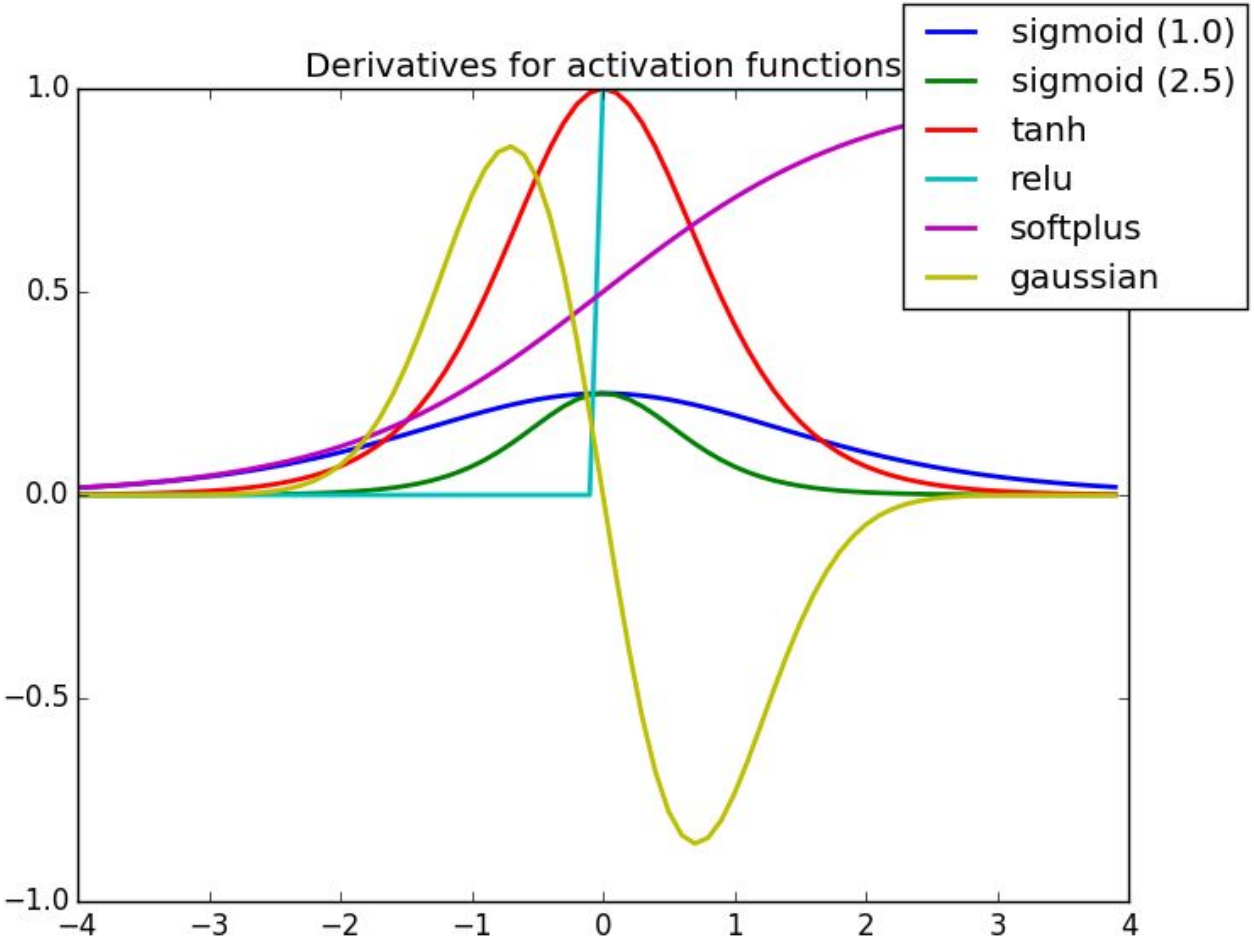

784 pixels
28x28
pixels
...
"neurons"
...
0
1
2
9

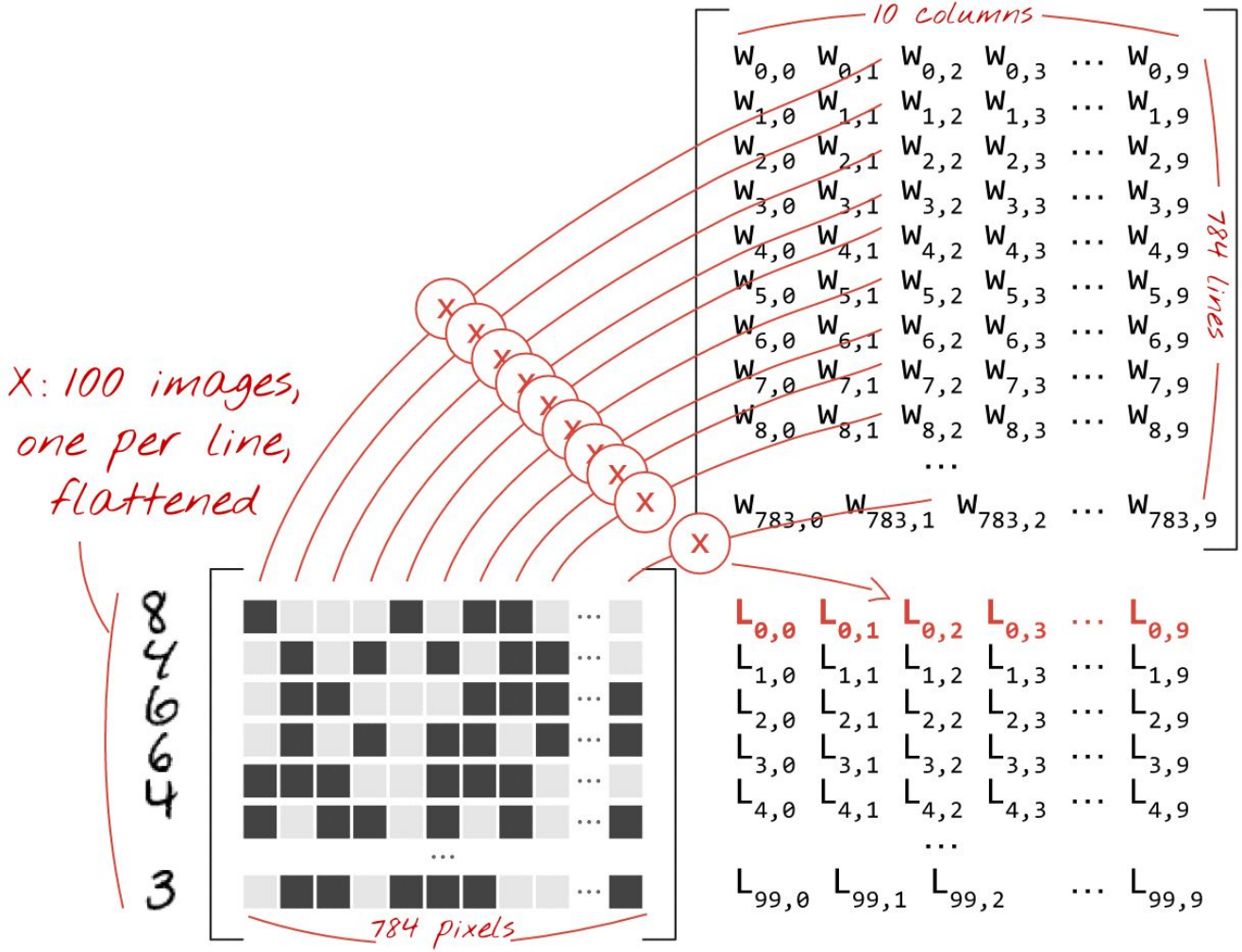

Inputs
Weights
Net input function
Activation function
1
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
output

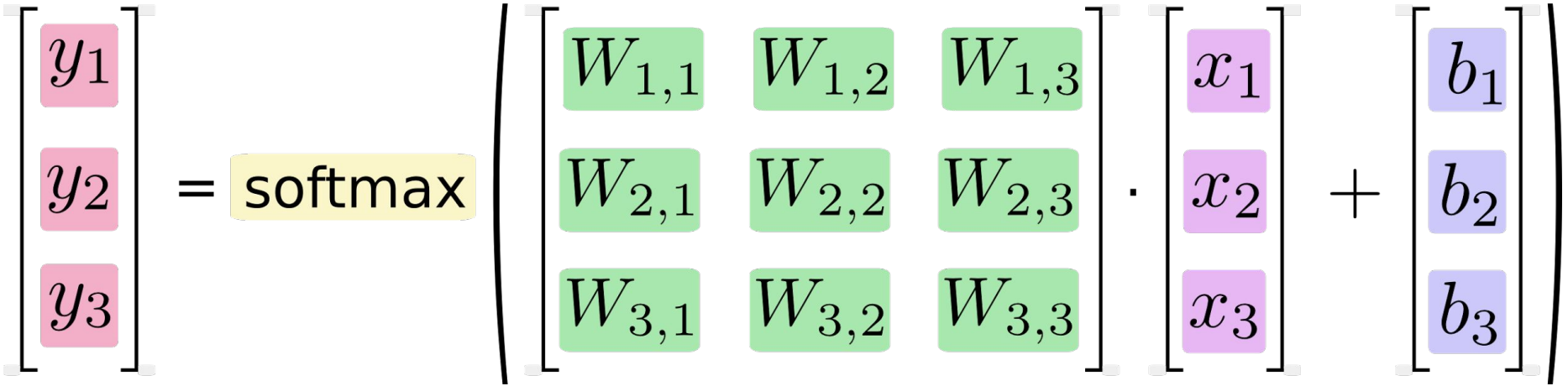
$$L_n = \sum_{j=1}^{M} x_{i,j} * w_{n,j} + b_n = w_n^T x_i$$

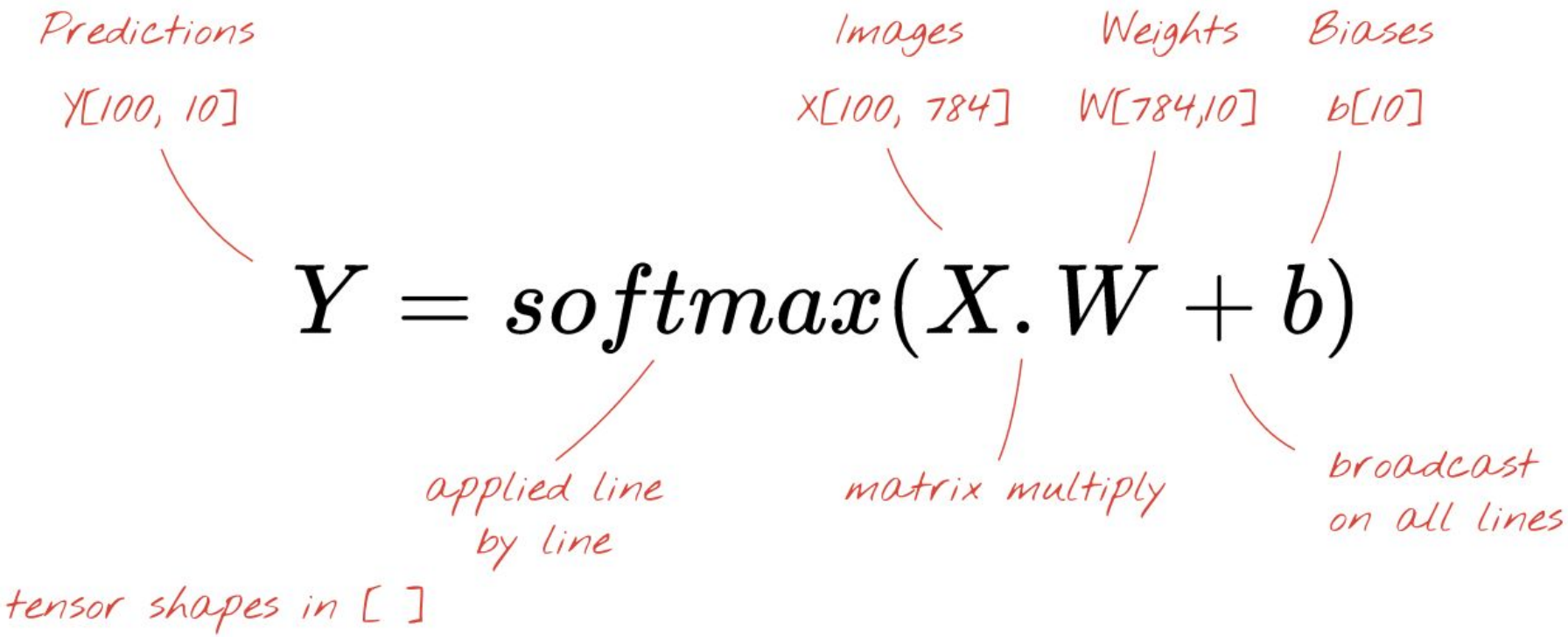
$$Softmax\left(L_n\right) = P\left(Y = n \middle| X = x_i\right)$$

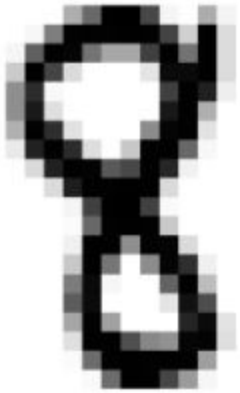
$$Softmax\left(L_n\right) = \frac{e_n^L}{||e^L||} = \frac{e_n^L}{\sum_{i=0}^{N} e_i^L}$$


Derivatives for activation functions
sigmoid (1.0)
sigmoid (2.5)
tanh
relu
softplus
gaussian
1.0
0.5
0.0
−0.5
−1.0
−4
−3
−2
−1
0
1
2
3
4


X: 100 images, one per line, flattened
10 columns
784 lines
784 pixels
8
4
6
6
4
3


$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$


Predictions
Y[100, 10]
Images
X[100, 784]
Weights
W[784,10]
Biases
b[10]
$Y = softmax(X.W + b)$
applied line by line
matrix multiply
broadcast on all lines
tensor shapes in [ ]

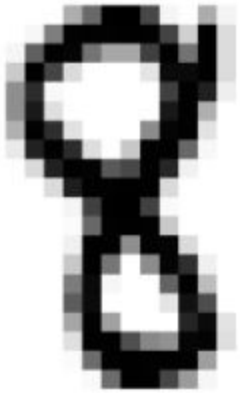


0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	1	0

$$Softmax\left(L_n\right)$$

0,1	0,2	0,1	0,5	0,4	0,1	0,3	0,2	0,1	0,6

На сколько хорошо?

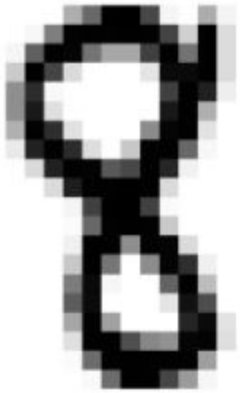


0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	1	0

$$Softmax\left(L_n\right)$$

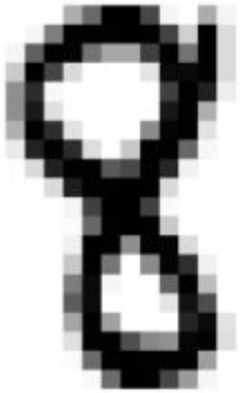
0,1	0,2	0,1	0,5	0,4	0,1	0,3	0,2	0,1	0,6

$$Loss\left(X_i\right) = -\sum Y_i' log\left(Y_i\right)$$



0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	1	0
0,1	0,2	0,1	0,5	0,4	0,1	0,3	0,2	0,1	0,6

$$Loss\,(X_i) = -log\,(0.1) = 2.30$$

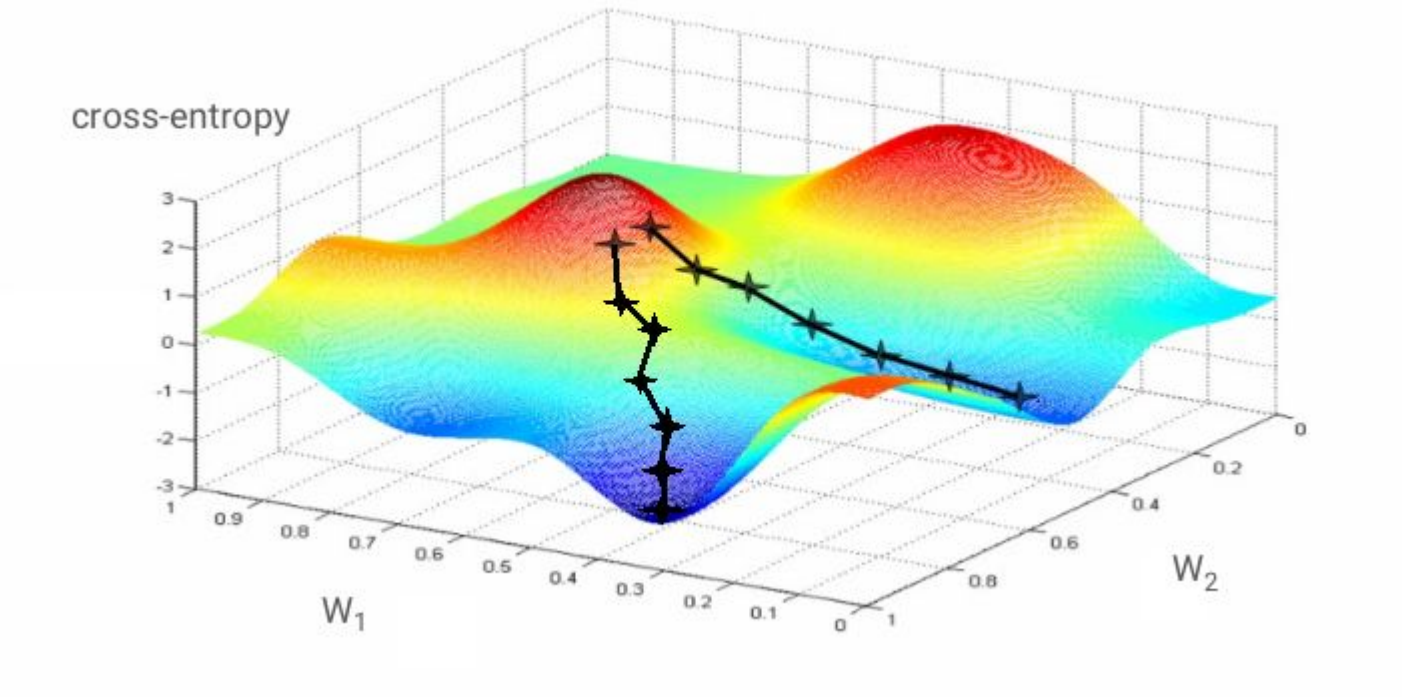


0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	1	0
0,1	0,2	0,1	0,5	0,4	0,1	0,3	0,2	0,1	0,6

$$Loss\,(X_i) = -log\,(0.7) = 0.36$$

Минимизация ошибки

Градиентный спуск



Оптимизаторы

- SGD
- Adagrad
- Adadelta
- Adam
- Adamax
- Nesterov Adam
- Rmsprop
- ….

- https://www.tensorflow.org/api_guides/python/train#Optimizers
- https://keras.io/optimizers/
- https://habrahabr.ru/post/318970/

Однослойная нейросеть на TF

```
X = tf.placeholder(tf.float32, [None, 784], name='X')
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```

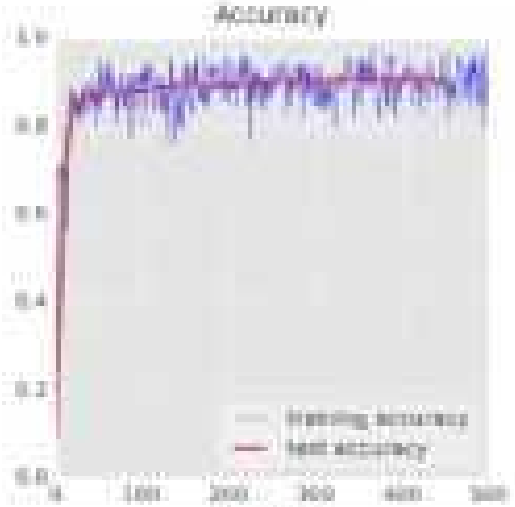
```
# Модель Y = softmax(X.W + b)
Y = tf.nn.softmax(tf.matmul(
    tf.reshape(X, [-1, 784]), W) + b)

# Подстановка для корректных значений входных данных
Y_ = tf.placeholder(tf.float32, [None, 10])
```

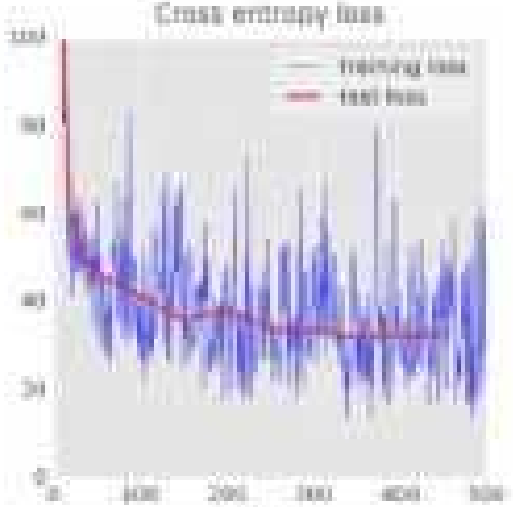
```
# Оптимизируем функцию потерь меотодом градиентного спуска
# 0.003 - это шаг градиента, гиперпараметр
optimizer = tf.train.GradientDescentOptimizer(0.003)

# Минимизируем потери
train_step = optimizer.minimize(cross_entropy)
```

```
# Функция потерь H = Sum(Y_ * log(Y))
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))
```

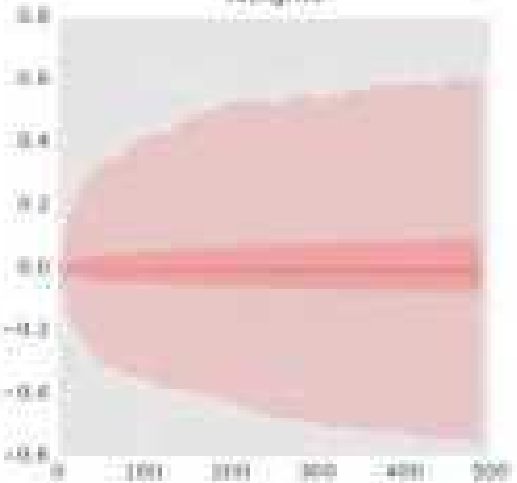
```
# доля корректных ответов
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
# точность предсказания
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

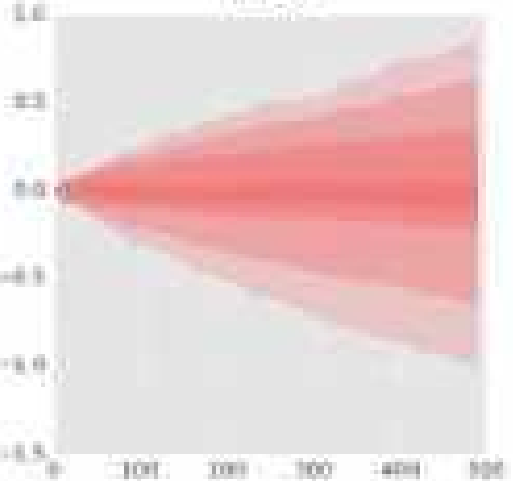
```
# инициализация сессии
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

acc, entropy = [], []  # переменные для рисования графиков

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)

    if i % 10 == 0:
        test_data={X: mnist.test.images, Y_: mnist.test.labels}
        a, c = sess.run([accuracy, cross_entropy], feed_dict=test_data)
        acc.append(a)
        entropy.append(c)
```

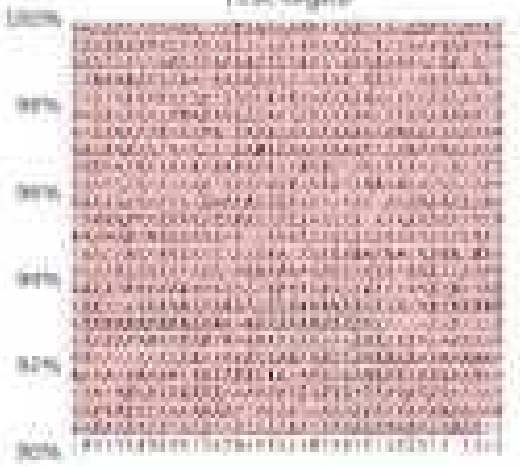

Accuracy
training accuracy
test accuracy


Cross entropy loss
training loss
test loss

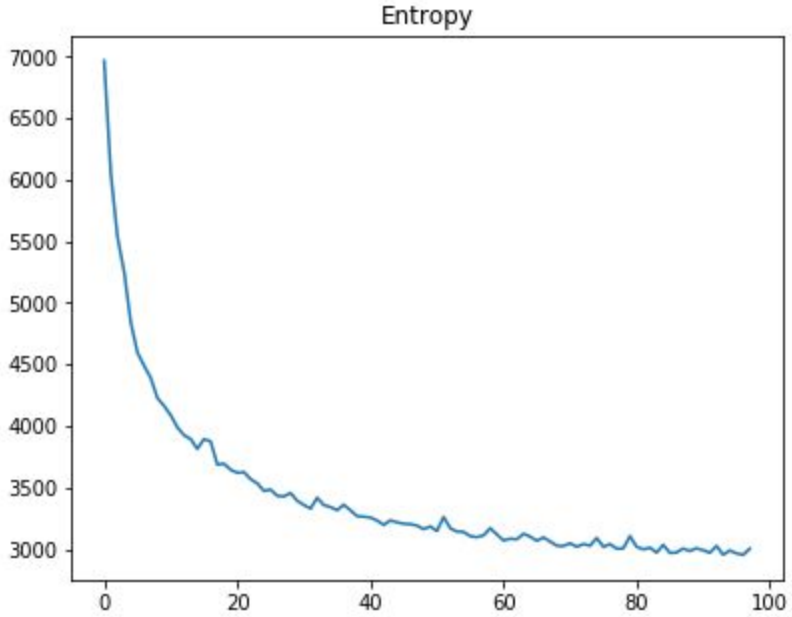

Training digits

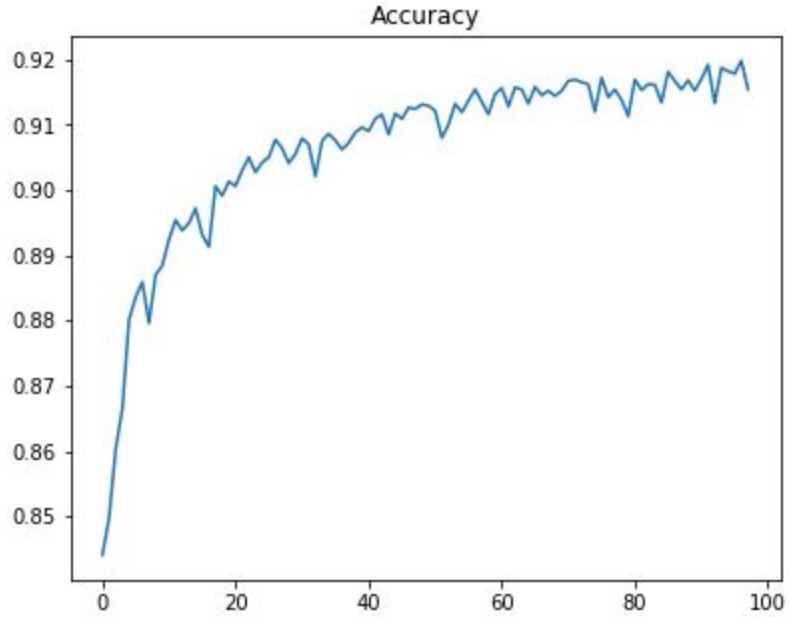

Weights


Biases


Test digits

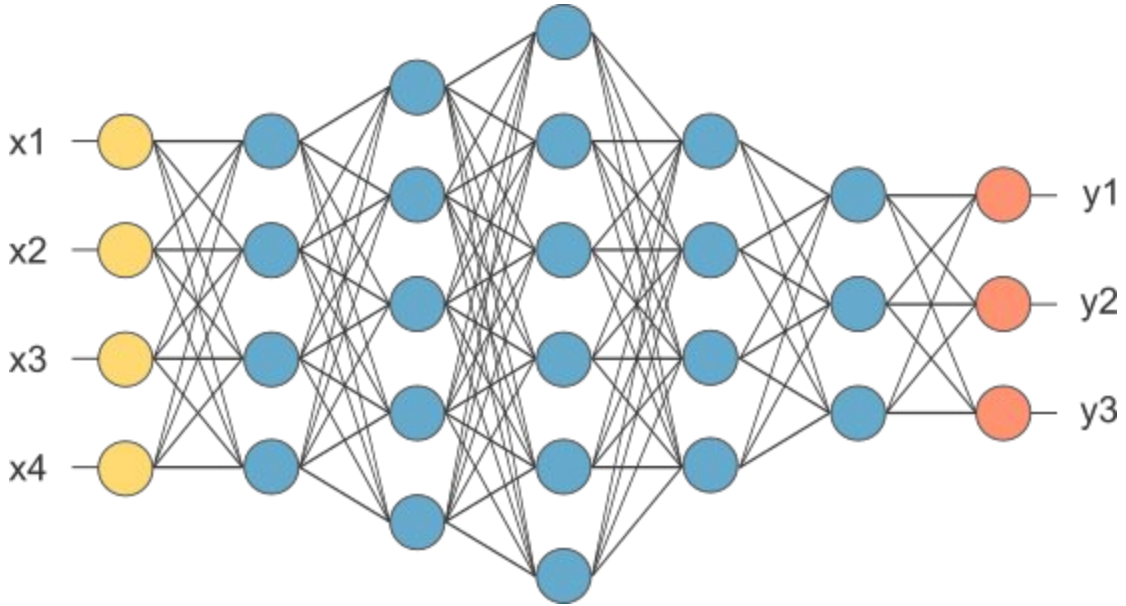
Результат однослойной сети (0.91)

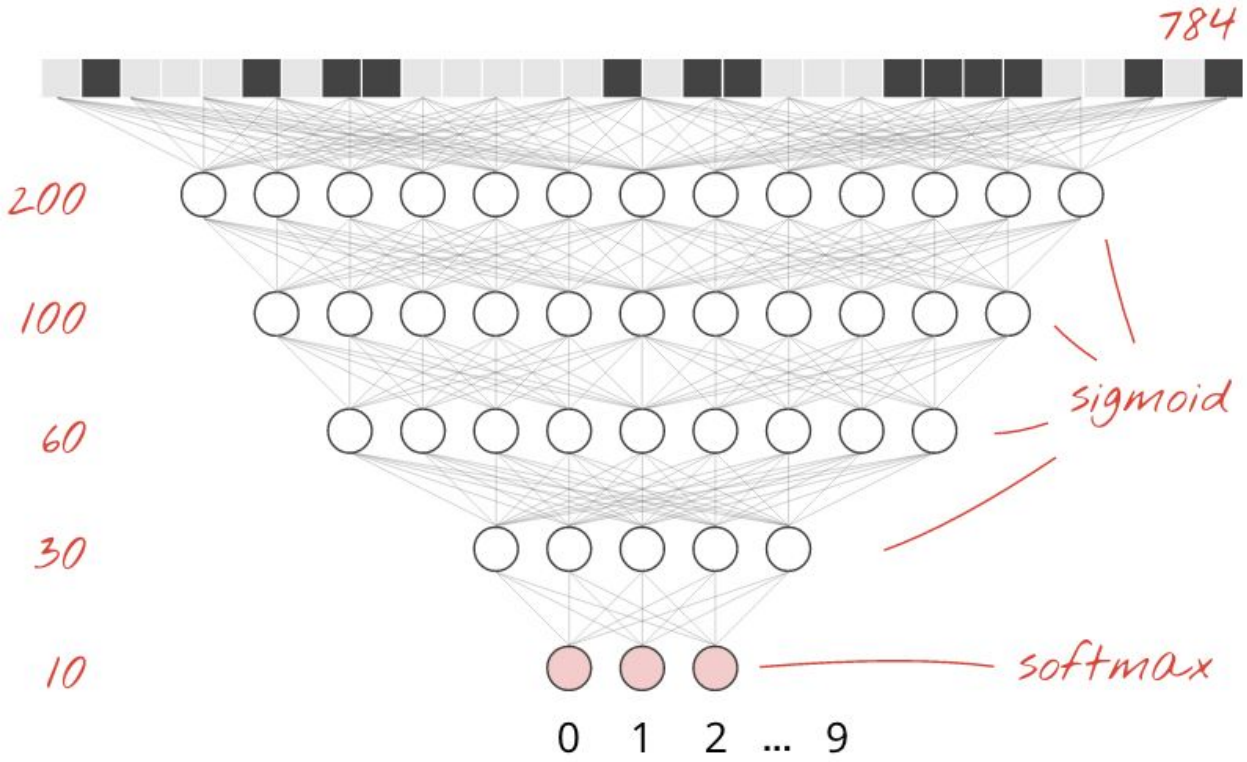




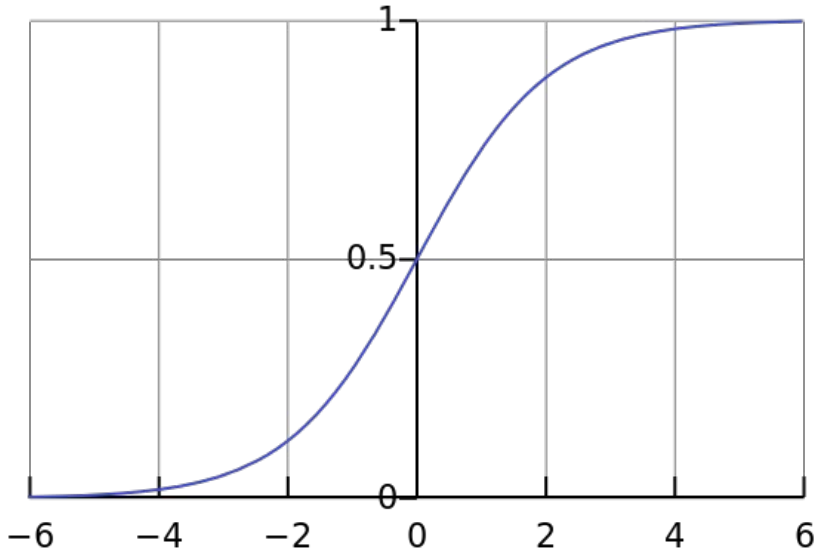

WE NEED TO GO
DEEPER

Многослойная нейронная сеть




784
200
100
60
30
10
sigmoid
softmax
0 1 2 ... 9

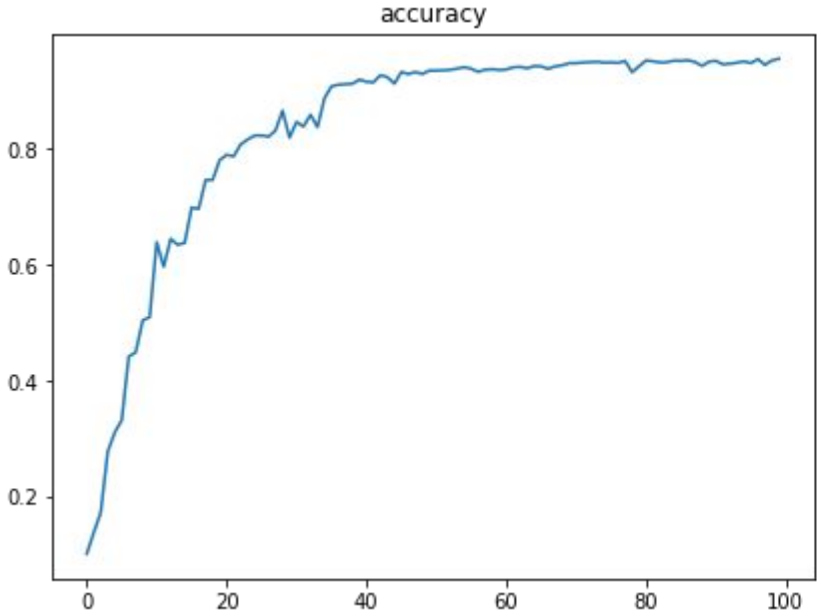
$$Sigmoid = \frac{1}{1 + e^{-x}}$$

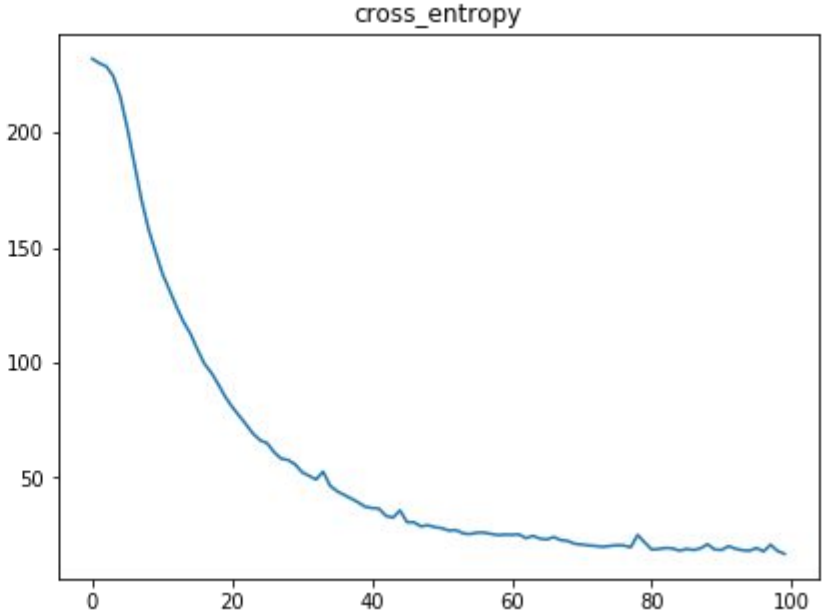


Объявляем веса

```
layer_sizes = [28*28, 200, 100, 60, 30, 10]

W1 = tf.Variable(tf.truncated_normal([layer_sizes[0], layer_sizes[1]], stddev=0.1))
B1 = tf.Variable(tf.zeros(layer_sizes[1])) # 200

W2 = tf.Variable(tf.truncated_normal([layer_sizes[1], layer_sizes[2]], stddev=0.1))
B2 = tf.Variable(tf.zeros(layer_sizes[2])) # 100

W3 = tf.Variable(tf.truncated_normal([layer_sizes[2], layer_sizes[3]], stddev=0.1))
B3 = tf.Variable(tf.zeros(layer_sizes[3])) # 60

W4 = tf.Variable(tf.truncated_normal([layer_sizes[3], layer_sizes[4]], stddev=0.1))
B4 = tf.Variable(tf.zeros(layer_sizes[4])) # 30

W5 = tf.Variable(tf.truncated_normal([layer_sizes[4], layer_sizes[5]], stddev=0.1))
B5 = tf.Variable(tf.zeros(layer_sizes[5])) # 10
```

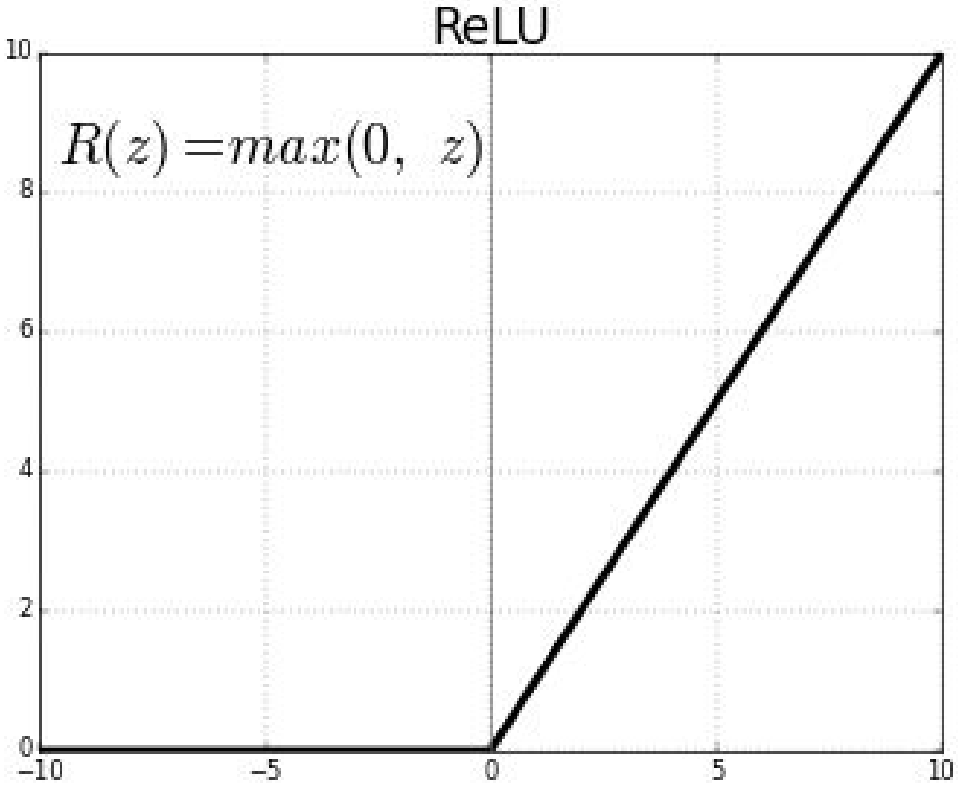
```
X =  tf.placeholder(tf.float32, [None, 784])

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)

Ylogits = tf.matmul(Y4, W5) + B5
Y = tf.nn.softmax(Ylogits)
Y_ = tf.placeholder(tf.float32, [None, 10])
```

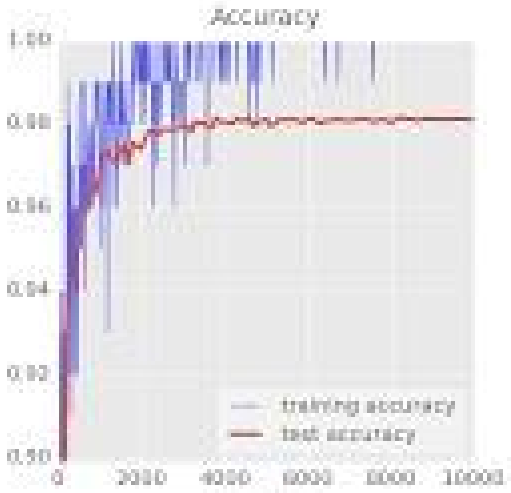


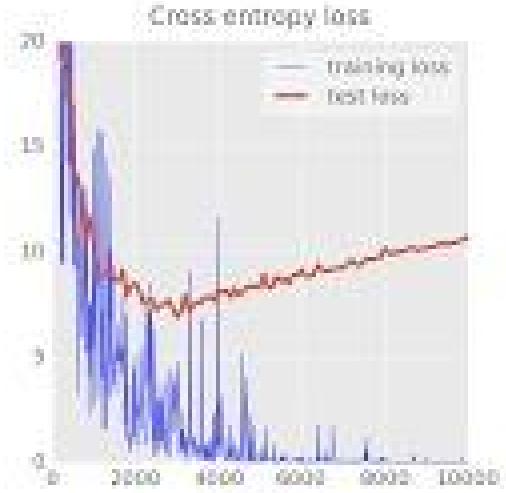


Медленное обучение
Нестабильность

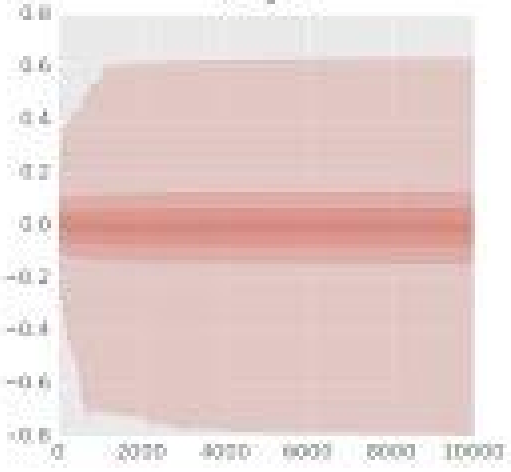
Relu = Rectified Linear Unit

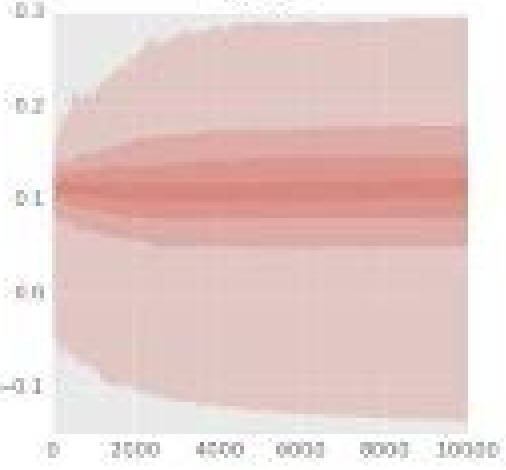



Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0
2000
4000
6000
8000
10000
training accuracy
test accuracy


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


Training digits


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


Biases
0.3
0.2
0.1
0.0
−0.1
0
2500
5000
7500
10000


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

Далее

- Learning rate decay
- Dropout for regularization
- Convolutional NN
- Deep conv NN

https://nbviewer.jupyter.org/github/mlomsk/ML_MNIST_Workshop_2017-09-28/blob/master/main.ipynb

Литература

- http://cs231n.github.io/
- https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/
- http://neuralnetworksanddeeplearning.com/
- http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html
- http://www.deeplearningbook.org/
- https://medium.com/@ageitgey/machine-learning-is-fun-part-2-a26a10b68df3
- https://keras.io/
- https://www.tensorflow.org/