tf2

June 14, 2018

```
In [1]: import sys
        !{sys.executable} -m pip install tensorflow
        # Just some imports
        from tensorflow.examples.tutorials.mnist import input_data
        import tensorflow as tf
        import random
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
Collecting tensorflow
  Using cached https://files.pythonhosted.org/packages/22/c6/d08f7c549330c2acc1b18b5c1f0f8d9d2a
Collecting grpcio>=1.8.6 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/1f/ea/664c589ec41b9e9ac6e20cc1fe9016f39
Collecting wheel>=0.26 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/81/30/e935244ca6165187ae8be876b6316ae20
Collecting gast>=0.2.0 (from tensorflow)
Collecting six>=1.10.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/67/4b/141a581104b1f6397bfa78ac9d43d8ad29
Collecting absl-py>=0.1.6 (from tensorflow)
Collecting astor>=0.6.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/b2/91/cc9805f1ff7b49f620136b3a7ca26f6a1
Collecting numpy>=1.13.3 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/4b/3d/9c0a34ad8544abef864714840fb8954d6
Collecting tensorboard<1.9.0,>=1.8.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/59/a6/0ae6092b7542cfedba6b2a1c9b8dceaf2
Collecting protobuf>=3.4.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/74/ad/ecd865eb1ba1ff7f6bd6bcb731a89d55b
Collecting termcolor>=1.1.0 (from tensorflow)
Collecting html5lib==0.9999999 (from tensorboard<1.9.0,>=1.8.0->tensorflow)
Collecting werkzeug>=0.11.10 (from tensorboard<1.9.0,>=1.8.0->tensorflow)
  Using cached https://files.pythonhosted.org/packages/20/c4/12e3e56473e52375aa29c4764e70d1b8f
Collecting markdown>=2.6.8 (from tensorboard<1.9.0,>=1.8.0->tensorflow)
  Using cached https://files.pythonhosted.org/packages/6d/7d/488b90f470b96531a3f5788cf12a93332
Collecting bleach==1.5.0 (from tensorboard<1.9.0,>=1.8.0->tensorflow)
  Using cached https://files.pythonhosted.org/packages/33/70/86c5fec937ea4964184d4d6c4f0b95515
Collecting setuptools (from protobuf>=3.4.0->tensorflow)
  Using cached https://files.pythonhosted.org/packages/7f/e1/820d941153923aac1d49d7fc37e17b6e7
```

```
Installing collected packages: six, grpcio, wheel, gast, absl-py, astor, numpy, html5lib, werkz
Successfully installed absl-py-0.2.2 astor-0.6.2 bleach-1.5.0 gast-0.2.0 grpcio-1.12.1 html5lil
```

1 MNIST dataset

MNIST is a large database of handwritten digits that is commonly used for training various image processing systems.

The MNIST database consists of 70000 grayscale digits, where each digit is an image with a size of 28x28 pixels.

The datased is splitted into three subsets:

1. Train set -- 55k images
2. Test set -- 10k images
3. Validation set -- 5k images

TensorFlow package has already built-in functions to deal with the MNIST dataset. Below one can find how to use it.

In [2]:
```
# Function for plotting the MNIST images
def plot(image):
    plt.axis('off')
    plt.imshow(image.reshape(28,28), cmap='gray')
    plt.show()
```

In [3]:
```
# Import data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
WARNING:tensorflow:From <ipython-input-3-978bdd810a8d>:2: read_data_sets (from tensorflow.cont
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
WARNING:tensorflow:From /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/contrib/l
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/contrib/l
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-images-idx3-ubyte.gz
WARNING:tensorflow:From /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/contrib/l
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/contrib/l
Instructions for updating:
Please use tf.one_hot on tensors.
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/contrib/l
```

```
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
```

Now we can easily deal with the MNIST data, by using:

1. mnist.train.images
2. mnist.test.images
3. mnist.validation.images

```
In [4]: # Print the shapes of images from each subset
        print ("IMAGES:")
        print (mnist.train.images.shape)
        print (mnist.test.images.shape)
        print (mnist.validation.images.shape)

        # Print the shapes of labels from each subset
        print ("LABELS:")
        print (mnist.train.labels.shape)
        print (mnist.test.labels.shape)
        print (mnist.validation.labels.shape)
```

```
IMAGES:
(55000, 784)
(10000, 784)
(5000, 784)
LABELS:
(55000, 10)
(10000, 10)
(5000, 10)
```

As we can see, every image is now a 784 dimensional vector, as by default it is reshaped from the matrix of a size 28x28 pixels.

Also we see that labels are 10 dimensional vectors, what is caused by the fact that they are in one-hot-encoding form.

```
In [5]: # Print the vector of sample training image
        mnist.train.images[0]
```

```
Out[5]: array([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.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
               0.        , 0.        , 0.        , 0.        , 0.        ,
```

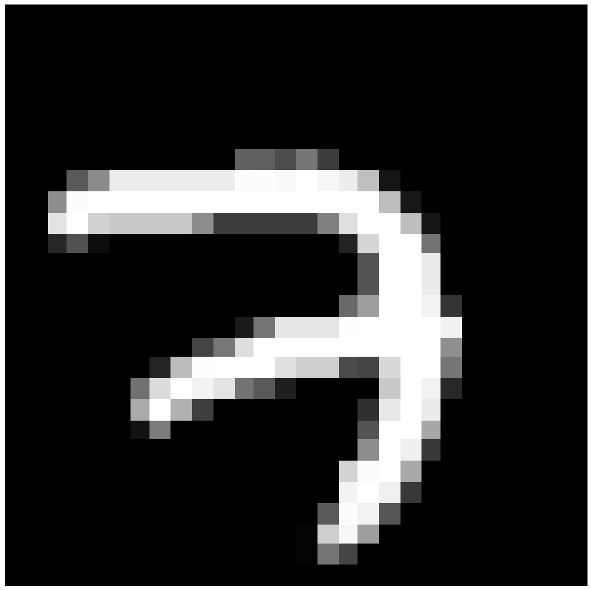
```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
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.        , 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.        , 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.        ,
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.        , 0.        ,
0.        , 0.        , 0.3803922 , 0.37647063, 0.3019608 ,
0.46274513, 0.2392157 , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.3529412 , 0.5411765 , 0.9215687 ,
0.9215687 , 0.9215687 , 0.9215687 , 0.9215687 , 0.9215687 ,
0.9843138 , 0.9843138 , 0.9725491 , 0.9960785 , 0.9607844 ,
0.9215687 , 0.74509805, 0.08235294, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.54901963,
0.9843138 , 0.9960785 , 0.9960785 , 0.9960785 , 0.9960785 ,
0.9960785 , 0.9960785 , 0.9960785 , 0.9960785 , 0.9960785 ,
0.9960785 , 0.9960785 , 0.9960785 , 0.9960785 , 0.9960785 ,
0.7411765 , 0.09019608, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.8862746 , 0.9960785 , 0.81568635,
```

```
0.7803922 , 0.7803922 , 0.7803922 , 0.7803922 , 0.54509807,
0.2392157 , 0.2392157 , 0.2392157 , 0.2392157 , 0.2392157 ,
0.5019608 , 0.8705883 , 0.9960785 , 0.9960785 , 0.7411765 ,
0.08235294, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.14901961, 0.32156864, 0.0509804 , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.13333334,
0.8352942 , 0.9960785 , 0.9960785 , 0.45098042, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.32941177, 0.9960785 ,
0.9960785 , 0.9176471 , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.32941177, 0.9960785 , 0.9960785 , 0.9176471 ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.4156863 , 0.6156863 ,
0.9960785 , 0.9960785 , 0.95294124, 0.20000002, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.09803922, 0.45882356, 0.8941177 , 0.8941177 ,
0.8941177 , 0.9921569 , 0.9960785 , 0.9960785 , 0.9960785 ,
0.9960785 , 0.94117653, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.26666668, 0.4666667 , 0.86274517,
0.9960785 , 0.9960785 , 0.9960785 , 0.9960785 , 0.9960785 ,
0.9960785 , 0.9960785 , 0.9960785 , 0.9960785 , 0.5568628 ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.14509805, 0.73333335,
0.9921569 , 0.9960785 , 0.9960785 , 0.9960785 , 0.8745099 ,
0.8078432 , 0.8078432 , 0.29411766, 0.26666668, 0.8431373 ,
0.9960785 , 0.9960785 , 0.45882356, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.4431373 , 0.8588236 , 0.9960785 , 0.9490197 , 0.89019614,
0.45098042, 0.34901962, 0.12156864, 0.        , 0.        ,
0.        , 0.        , 0.7843138 , 0.9960785 , 0.9450981 ,
```

```
0.16078432, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.6627451 , 0.9960785 ,
0.6901961 , 0.24313727, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.18823531,
0.9058824 , 0.9960785 , 0.9176471 , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.07058824, 0.48627454, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.32941177, 0.9960785 , 0.9960785 ,
0.6509804 , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.54509807, 0.9960785 , 0.9333334 , 0.22352943, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.8235295 , 0.9803922 , 0.9960785 ,
0.65882355, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.9490197 , 0.9960785 , 0.93725497, 0.22352943, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.34901962, 0.9843138 , 0.9450981 ,
0.3372549 , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.01960784,
0.8078432 , 0.96470594, 0.6156863 , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.01568628, 0.45882356, 0.27058825,
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.        , 0.        ,
0.        , 0.        , 0.        , 0.        ], dtype=float32)
```

In [6]:
```
# Plot the sample training image
plot(mnist.train.images[0])
```



In [7]:
```
# Print the label of sample training image
mnist.train.labels[0]
```

Out[7]:
```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.])
```

2 Introduction to TensorFlow

TensorFlow is popular open-source deep learning library developed by Google.

\

TensorFlow uses a dataflow graph to represent your computation in terms of the dependencies between individual operations.

A computational graph is a series of TensorFlow operations arranged into a graph. The graph is composed of two types of objects:

- Operations (or "ops"): The nodes of the graph. Operations describe calculations that consume and produce tensors.
- Tensors: The edges in the graph. These represent the values that will flow through the graph.

\
This leads to a low-level programming model in which you first define the dataflow graph, then create a TensorFlow session to run parts of the graph across a set of local and remote devices.
Dataflow has several advantages that TensorFlow leverages when executing your programs:

- Parallelism
- Distributed execution
- Compilation
- Portability

\
We highly recommend to read the resources from the following link, to understand the TF architecture.

2.1 Session initialization

To evaluate tensors, instantiate a tf.Session object, informally known as a session. A session encapsulates the state of the TensorFlow runtime, and runs TensorFlow operations. If a tf.Graph is like a .py file, a tf.Session is like the python executable.

```
In [8]: sess = tf.InteractiveSession()
```

2.2 Tensor

TensorFlow, is a framework to define and run computations involving tensors. A tensor is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes.
\
TensorFlow programs work by first building a graph of tf.Tensor objects, detailing how each tensor is computed based on the other available tensors and then by running parts of this graph to achieve the desired results.
\
tf.Tensor does not exist outside the context of a single session.run call.

2.2.1 Initialization

```
In [9]: node1 = tf.constant(3.0)
        node2 = tf.constant(4, dtype=tf.int32)
        node3 = tf.constant(0., name="Zero_tensor")
        node4 = tf.constant([1., 2., 3.])
        node5 = tf.constant([[1., 2., 3.], [4., 5., 6.]])
        node6 = tf.ones([3, 3, 3])
```

```
In [10]: print(node1)
         print(type(node1))
         print()

         print(node2.eval())
         print(sess.run(node2))
```

```
print()

print(node3.name)
print()

print(node4.shape)
print()

print(node5.eval())
print()

print(node6)
print(sess.run(node6))
```

```
Tensor("Const:0", shape=(), dtype=float32)
<class 'tensorflow.python.framework.ops.Tensor'>

4
4

Zero_tensor:0

(3,)

[[1. 2. 3.]
 [4. 5. 6.]]

Tensor("ones:0", shape=(3, 3, 3), dtype=float32)
[[[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]]
```

2.2.2 Operations

```
In [11]: x = tf.constant(2.)
         y = tf.constant(3.)

         add_1 = x + y
         add_2 = tf.add(x, y, name='our_add_node')
```

```
          print(add_1)
          print(add_1.eval())
          print()

          print(add_2)
          print(add_2.eval())
```

```
Tensor("add:0", shape=(), dtype=float32)
5.0

Tensor("our_add_node:0", shape=(), dtype=float32)
5.0
```

```
In [12]: x = tf.constant(2.)
         y = tf.constant([3., 4., 5.])

         add_3 = x + y

         print(add_3)
         print(add_3.eval())
```

```
Tensor("add_1:0", shape=(3,), dtype=float32)
[5. 6. 7.]
```

```
In [13]: xx = tf.random_normal(shape=(100, 10), name='xx')
         yy = tf.random_normal(shape=(10, 2), name='yy')

         xyxy = tf.matmul(xx, yy)

         print(xyxy)
```

```
Tensor("MatMul:0", shape=(100, 2), dtype=float32)
```

2.2.3 Tensor does not exist outside the context of a single session.run call

The result shows a different random value on each call to run, but a consistent value during a single run (out1 and out2 receive the same random input):

```
In [14]: vec = tf.random_uniform(shape=(3,))
         out1 = vec + 1
         out2 = vec + 2

         print(sess.run(vec))
         print(sess.run(vec))
         print(sess.run((out1, out2)))
```

```
[0.32856107 0.76389205 0.45400012]
[0.85200894 0.9873638  0.9982232 ]
(array([1.8682148, 1.9791001, 1.5944145], dtype=float32), array([2.8682148, 2.9791002, 2.59441
```

2.2.4 Placeholders

A placeholder represents an entry point for us to feed actual data values into tensors. It is not initialized and contains no data. A placeholder generates an error if it is executed without a feed.

Placeholder for a single number

```
In [15]: x = tf.placeholder(tf.float32, shape=[1,1])
         y = tf.matmul(x, x)
```

```
In [16]: """ A placeholder generates an error if it is executed without a feed """
         print(sess.run(y))  # ERROR: will fail because x was not fed.
```

```
        ---------------------------------------------------------------------------

        InvalidArgumentError                      Traceback (most recent call last)

        /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
       1321     try:
    -> 1322       return fn(*args)
       1323     except errors.OpError as e:


        /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
       1306       return self._call_tf_sessionrun(
    -> 1307           options, feed_dict, fetch_list, target_list, run_metadata)
       1308


        /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
       1408           self._session, options, feed_dict, fetch_list, target_list,
    -> 1409           run_metadata)
       1410     else:


        InvalidArgumentError: You must feed a value for placeholder tensor 'Placeholder' with d
             [[Node: Placeholder = Placeholder[dtype=DT_FLOAT, shape=[1,1], _device="/job:local


    During handling of the above exception, another exception occurred:
```

```
    InvalidArgumentError                      Traceback (most recent call last)

    <ipython-input-16-921d4d91d90a> in <module>()
      1 """ A placeholder generates an error if it is executed without a feed """
----> 2 print(sess.run(y))  # ERROR: will fail because x was not fed.


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
    898     try:
    899       result = self._run(None, fetches, feed_dict, options_ptr,
--> 900                          run_metadata_ptr)
    901       if run_metadata:
    902         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
   1133     if final_fetches or final_targets or (handle and feed_dict_tensor):
   1134       results = self._do_run(handle, final_targets, final_fetches,
-> 1135                              feed_dict_tensor, options, run_metadata)
   1136     else:
   1137       results = []


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
   1314     if handle is None:
   1315       return self._do_call(_run_fn, feeds, fetches, targets, options,
-> 1316                            run_metadata)
   1317     else:
   1318       return self._do_call(_prun_fn, handle, feeds, fetches)


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
   1333         except KeyError:
   1334           pass
-> 1335       raise type(e)(node_def, op, message)
   1336
   1337   def _extend_graph(self):


    InvalidArgumentError: You must feed a value for placeholder tensor 'Placeholder' with d
         [[Node: Placeholder = Placeholder[dtype=DT_FLOAT, shape=[1,1], _device="/job:local

Caused by op 'Placeholder', defined at:
  File "/usr/lib/python3.6/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "/usr/lib/python3.6/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/usr/lib/python3/dist-packages/ipykernel_launcher.py", line 16, in <module>
```

```
    app.launch_new_instance()
  File "/usr/lib/python3/dist-packages/traitlets/config/application.py", line 658, in launc
    app.start()
  File "/usr/lib/python3/dist-packages/ipykernel/kernelapp.py", line 486, in start
    self.io_loop.start()
  File "/usr/lib/python3/dist-packages/tornado/platform/asyncio.py", line 112, in start
    self.asyncio_loop.run_forever()
  File "/usr/lib/python3.6/asyncio/base_events.py", line 422, in run_forever
    self._run_once()
  File "/usr/lib/python3.6/asyncio/base_events.py", line 1432, in _run_once
    handle._run()
  File "/usr/lib/python3.6/asyncio/events.py", line 145, in _run
    self._callback(*self._args)
  File "/usr/lib/python3/dist-packages/tornado/ioloop.py", line 760, in _run_callback
    ret = callback()
  File "/usr/lib/python3/dist-packages/tornado/stack_context.py", line 276, in null_wrappe
    return fn(*args, **kwargs)
  File "/usr/lib/python3/dist-packages/zmq/eventloop/zmqstream.py", line 536, in <lambda>
    self.io_loop.add_callback(lambda : self._handle_events(self.socket, 0))
  File "/usr/lib/python3/dist-packages/zmq/eventloop/zmqstream.py", line 450, in _handle_e
    self._handle_recv()
  File "/usr/lib/python3/dist-packages/zmq/eventloop/zmqstream.py", line 480, in _handle_r
    self._run_callback(callback, msg)
  File "/usr/lib/python3/dist-packages/zmq/eventloop/zmqstream.py", line 432, in _run_call
    callback(*args, **kwargs)
  File "/usr/lib/python3/dist-packages/tornado/stack_context.py", line 276, in null_wrappe
    return fn(*args, **kwargs)
  File "/usr/lib/python3/dist-packages/ipykernel/kernelbase.py", line 283, in dispatcher
    return self.dispatch_shell(stream, msg)
  File "/usr/lib/python3/dist-packages/ipykernel/kernelbase.py", line 233, in dispatch_she
    handler(stream, idents, msg)
  File "/usr/lib/python3/dist-packages/ipykernel/kernelbase.py", line 399, in execute_requ
    user_expressions, allow_stdin)
  File "/usr/lib/python3/dist-packages/ipykernel/ipkernel.py", line 208, in do_execute
    res = shell.run_cell(code, store_history=store_history, silent=silent)
  File "/usr/lib/python3/dist-packages/ipykernel/zmqshell.py", line 537, in run_cell
    return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
  File "/usr/lib/python3/dist-packages/IPython/core/interactiveshell.py", line 2718, in ru
    interactivity=interactivity, compiler=compiler, result=result)
  File "/usr/lib/python3/dist-packages/IPython/core/interactiveshell.py", line 2822, in ru
    if self.run_code(code, result):
  File "/usr/lib/python3/dist-packages/IPython/core/interactiveshell.py", line 2882, in ru
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-15-ec0731bab3f8>", line 1, in <module>
    x = tf.placeholder(tf.float32, shape=[1,1])
  File "/home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/ops/array_ops
    return gen_array_ops.placeholder(dtype=dtype, shape=shape, name=name)
  File "/home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/ops/gen_array
```

```
    "Placeholder", dtype=dtype, shape=shape, name=name)
  File "/home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/framework/op_d
    op_def=op_def)
  File "/home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/framework/ops
    op_def=op_def)
  File "/home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/framework/ops
    self._traceback = self._graph._extract_stack()  # pylint: disable=protected-access

InvalidArgumentError (see above for traceback): You must feed a value for placeholder tenso
	 [[Node: Placeholder = Placeholder[dtype=DT_FLOAT, shape=[1,1], _device="/job:local
```

```
In [17]: number = [[3.]]
         print(sess.run(y, feed_dict={x: number}))  # Will succeed.
```

```
[[9.]]
```

Placeholder for a tensor with undefined length

```
In [18]: x = tf.placeholder(tf.float32, shape=[None, 5])
         y = x * 2

In [19]: tensor = np.ones((1, 5))
         print(tensor)
         print()
         print(sess.run(y, feed_dict={x: tensor}))
```

```
[[1. 1. 1. 1. 1.]]

[[2. 2. 2. 2. 2.]]
```

Operations on placeholders

```
In [20]: x = tf.placeholder(tf.float32, shape=[1, None])
         y = tf.placeholder(tf.float32, shape=[1, None])

         z_1 = x + y
         z_2 = tf.matmul(x, tf.transpose(y))

In [21]: x_tensor = [[1., 2., 3.]]
         y_tensor = [[11., 12., 13.]]
         print(sess.run(z_1, feed_dict={x: x_tensor, y: y_tensor}))
         print()
         print(sess.run(z_2, feed_dict={x: x_tensor, y: y_tensor}))
```

```
[[12. 14. 16.]]

[[74.]]
```

2.3 Variable

A TensorFlow variable is the best way to represent the state manipulated by your program. A tf.Variable represents a tensor whose value can be changed by running ops on it. Internally, a tf.Variable stores a tensor. Specific ops allow you to read and modify the values of this tensor.
\
Unlike tf.Tensor objects, a tf.Variable exists outside the context of a single session.run call.

```
In [22]: var_1 = tf.get_variable("var_1", shape=[2, 3]) # default type tf.float32, default ini
         var_2 = tf.get_variable("var_2", shape=[5], initializer=tf.constant_initializer(1000.
         var_3 = tf.get_variable("var_3", shape=[3, 3, 3], initializer=tf.initializers.random_

         var_4 = tf.Variable(tf.constant(3., shape=[1, 2]))
         var_5 = tf.Variable(tf.random_normal([2, 1]))
         var_6 = tf.Variable(tf.random_uniform([1, 1]), name="var_6")
```

```
In [23]: """ Before you can use a variable, it must be initialized!!! """
         print(var_1)
         print(var_1.eval()) # ERROR: will fail because variables are not initialized!
```

```
<tf.Variable 'var_1:0' shape=(2, 3) dtype=float32_ref>



        ---------------------------------------------------------------------------

        FailedPreconditionError                   Traceback (most recent call last)

        /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
       1321     try:
    -> 1322       return fn(*args)
       1323     except errors.OpError as e:


        /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
       1306       return self._call_tf_sessionrun(
    -> 1307           options, feed_dict, fetch_list, target_list, run_metadata)
       1308


        /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
       1408           self._session, options, feed_dict, fetch_list, target_list,
    -> 1409           run_metadata)
       1410     else:


        FailedPreconditionError: Attempting to use uninitialized value var_1
             [[Node: _retval_var_1_0_0 = _Retval[T=DT_FLOAT, index=0, _device="/job:localhost/
```

```
During handling of the above exception, another exception occurred:


    FailedPreconditionError                   Traceback (most recent call last)

    <ipython-input-23-d637e93c1edd> in <module>()
      1 """ Before you can use a variable, it must be initialized!!! """
      2 print(var_1)
----> 3 print(var_1.eval()) # ERROR: will fail because variables are not initialized!


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/ops/variables.py i
    550       A numpy `ndarray` with a copy of the value of this variable.
    551     """
--> 552     return self._variable.eval(session=session)
    553
    554   def initialized_value(self):


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/framework/ops.py i
    708
    709     """
--> 710     return _eval_using_default_session(self, feed_dict, self.graph, session)
    711
    712


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/framework/ops.py i
   5178                        "the tensor's graph is different from the session's "
   5179                        "graph.")
-> 5180   return session.run(tensors, feed_dict)
   5181
   5182


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
    898     try:
    899       result = self._run(None, fetches, feed_dict, options_ptr,
--> 900                          run_metadata_ptr)
    901       if run_metadata:
    902         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
   1133     if final_fetches or final_targets or (handle and feed_dict_tensor):
   1134       results = self._do_run(handle, final_targets, final_fetches,
-> 1135                              feed_dict_tensor, options, run_metadata)
```

```
    1136     else:
    1137       results = []


 /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
    1314     if handle is None:
    1315       return self._do_call(_run_fn, feeds, fetches, targets, options,
 -> 1316                            run_metadata)
    1317     else:
    1318       return self._do_call(_prun_fn, handle, feeds, fetches)


 /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/client/session.py
    1333         except KeyError:
    1334           pass
 -> 1335       raise type(e)(node_def, op, message)
    1336
    1337   def _extend_graph(self):


 FailedPreconditionError: Attempting to use uninitialized value var_1
      [[Node: _retval_var_1_0_0 = _Retval[T=DT_FLOAT, index=0, _device="/job:localhost/
```

In [24]:
```
""" Before you can use a variable, it must be initialized!!! """
sess.run(tf.global_variables_initializer())
```

In [25]:
```
print(var_1)
print(var_1.eval())
print()

print(var_2)
print(var_2.eval())
print()

print(var_3)
print(sess.run(var_3))
print()

print(var_4)
print(sess.run(var_4))
print()

print(var_5)
print(sess.run(var_5))
print()

print(var_6)
```

```
        print(var_6.eval())
        print()
```

```
<tf.Variable 'var_1:0' shape=(2, 3) dtype=float32_ref>
[[ 0.6339016  -0.85074687 -0.40628666]
 [-0.13980776  1.0913789   0.1443994 ]]

<tf.Variable 'var_2:0' shape=(5,) dtype=float32_ref>
[1000. 1000. 1000. 1000. 1000.]

<tf.Variable 'var_3:0' shape=(3, 3, 3) dtype=float32_ref>
[[[ 0.173098   -0.7522116   0.53066134]
  [ 0.36987022  0.94048494 -0.71320784]
  [ 1.9105393   0.7954687  -0.44640672]]

 [[-0.9223671  -0.8734648   0.36734757]
  [-0.68438435  0.19258565 -2.2917182 ]
  [-0.02112285  1.7471564  -0.9517774 ]]

 [[ 2.4070487  -0.33092928 -0.82059413]
  [ 0.40101963 -1.0287039  -0.24965385]
  [-0.12046738  0.33668953 -1.3680288 ]]]

<tf.Variable 'Variable:0' shape=(1, 2) dtype=float32_ref>
[[3. 3.]]

<tf.Variable 'Variable_1:0' shape=(2, 1) dtype=float32_ref>
[[-2.6204815 ]
 [-0.51063406]]

<tf.Variable 'var_6:0' shape=(1, 1) dtype=float32_ref>
[[0.98330724]]
```

2.3.1 Variable exists outside the context of a single session.run call

```
In [26]: vec = tf.get_variable("vec", shape=(3,), initializer=tf.initializers.random_normal())
         sess.run(tf.global_variables_initializer())
         out1 = vec + 1
         out2 = vec + 2

         print(sess.run(vec))
         print(sess.run(vec))
         print(sess.run((out1, out2)))
```

```
[ 0.9746403  0.8640773 -1.0308193]
[ 0.9746403  0.8640773 -1.0308193]
```

```
(array([ 1.9746404,  1.8640773, -0.0308193], dtype=float32), array([2.9746404, 2.8640773, 0.969
```

2.3.2 Computing gradients

```
In [27]: var_7 = tf.matmul(var_4, var_5)

In [28]: g = tf.gradients(var_7, [var_4, var_5])
         print(g)
         print(g[0].eval())
         print(g[1].eval())
```

```
[<tf.Tensor 'gradients/MatMul_3_grad/MatMul:0' shape=(1, 2) dtype=float32>, <tf.Tensor 'gradien
[[1.0435681 0.6223902]]
[[3.]
 [3.]]
```

```
In [29]: x_1 = tf.constant(2., shape=[2,1])
         x_2 = tf.constant(3., shape=[1,2])
         y = tf.matmul(x_1, x_2)

         g = tf.gradients(y, [x_1, x_2])
         print(g)
         print(g[0].eval())
         print(g[1].eval())
```

```
[<tf.Tensor 'gradients_1/MatMul_4_grad/MatMul:0' shape=(2, 1) dtype=float32>, <tf.Tensor 'grad
[[6.]
 [6.]]
[[4. 4.]]
```

2.4 Optimizers

In TensorFlow one can use fast, efficient gradient optimizers to minimize the given function. \ In the following example we will show how to use optimizers in TF, by minimizing x^2 function, with using of the Gradient Decent Optimizer.

Defining the starting point x and the function y = x^2 Notice, that the starting point should be initialized as a variable, not a tensor, as the optimizer have to change its value, by repeatedly subtracting the gradient of function in order to minimize y value.

```
In [30]: x = tf.get_variable("opt_x", dtype=tf.float32, initializer=tf.constant_initializer(10
         y = tf.pow(x, [2.])
```

Defining the Gradient Descent Optimizer

```
In [31]: train_step = tf.train.GradientDescentOptimizer(0.1).minimize(y)
```

Iterate to minimize function

In [32]:
```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print("x = ", x.eval())
    print("y = ", y.eval())
    for i in range(200):
        train_step.run()
        if i % 10 == 0:
          print(x.eval())
```

```
x =  [[1000.]]
y =  [[1000000.]]
[[800.]]
[[85.89934]]
[[9.2233715]]
[[0.99035203]]
[[0.10633824]]
[[0.01141798]]
[[0.001226]]
[[0.00013164]]
[[1.4134782e-05]]
[[1.5177106e-06]]
[[1.6296293e-07]]
[[1.7498012e-08]]
[[1.8788346e-09]]
[[2.0173833e-10]]
[[2.1661489e-11]]
[[2.3258847e-12]]
[[2.4973995e-13]]
[[2.6815621e-14]]
[[2.8793051e-15]]
[[3.0916303e-16]]
```

In [33]:
```
""" Optimizer needs variables that can be mutated in order to minimize a function! ""
x = tf.constant(1000.)
y = tf.pow(x, [2.])

train_step = tf.train.GradientDescentOptimizer(0.1).minimize(y) # ERROR: there are no
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-33-3091d36536e0> in <module>()
  3 y = tf.pow(x, [2.])
  4
```

```
----> 5 train_step = tf.train.GradientDescentOptimizer(0.1).minimize(y) # ERROR: there are


    /home/karolinka/.local/lib/python3.6/site-packages/tensorflow/python/training/optimize
    419             "No gradients provided for any variable, check your graph for ops"
    420             " that do not support gradients, between variables %s and loss %s." %
--> 421             ([str(v) for _, v in grads_and_vars], loss))
    422
    423       return self.apply_gradients(grads_and_vars, global_step=global_step,


    ValueError: No gradients provided for any variable, check your graph for ops that do no
```

3 Softmax regression for MNIST classification

Example of the softmax regression applied to MNIST digits classification.

3.1 Building the softmax architecture

Create placeholders for training data. Remember about a propper shape of training images (in mnist.train.images every digit is a 784D vector) and labels (In training dataset labels are in one-hot-encoding form).

```
In [34]: x = tf.placeholder(tf.float32, [None, 784])
         y_labels = tf.placeholder(tf.float32, [None,10])
```

Initialize weight matrix and biases vector

```
In [35]: W = tf.get_variable("softmax_W", dtype=tf.float32, initializer=tf.initializers.trunca
         b = tf.get_variable("softmax_b",dtype=tf.float32,initializer=tf.constant_initializer(
         #W = tf.get_variable("softmax_W", dtype=tf.float32, initializer=tf.initializers.trunc
```

Define classificator Multiply input image by a weighr matrix, them add biases and apply softmax function.

```
In [36]: y = tf.nn.softmax(tf.matmul(x, W) + b)
```

3.2 Softmax training

We will now train the softmax classifier. For this purpose we have to define the loss function, cross entropy in this example and optimizer, we will use Adam.

Define the loss function Here we define the cross entropy loss function by hand, but one should notice, that better option is to use the tf.nn.softmax_cross_entropy_with_logits_v2 function, as this is more numerically stable solution (as we are taking the log of softmax).

```
In [37]: cross_entropy = -tf.reduce_sum(input_tensor=y_labels*tf.log(y),reduction_indices=[1])
         #cross_entropy=-tf.reduce_sum(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_lab
         cross_entropy = tf.reduce_mean(cross_entropy)
```

Define the optimizer

```
In [38]: train_step = tf.train.GradientDescentOptimizer(0.1).minimize(cross_entropy)
```

Check whether softmax classifier returns correct predictions and calculate the accuracy

```
In [39]: """ Create a vector that tells us, whether the predictions from our net - y_conv
             are equal to the correct digit labels - y. """
         correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_labels, 1))
         correct_prediction = tf.cast(correct_prediction, tf.float32)

         """ Calculate the accurracy of correct predictions """
         accuracy = tf.reduce_mean(correct_prediction)
```

Training the network The following code will train our network with using of our predefined SGD optimizer, based on the batch size equal to 64 and with 2000 steps.

```
In [40]: with tf.Session() as sess:
           sess.run(tf.global_variables_initializer())
           for i in range(2000):
             # Get the batch with 64 images from the MNIST training set
             batch = mnist.train.next_batch(64)

             # Fill data into placeholders and run the training step
             train_step.run(feed_dict={x: batch[0], y_labels: batch[1]})

             # Print the validation accuracy every 100 steps
             if i % 100 == 0:
               validation_accuracy = accuracy.eval(feed_dict={
                   x: mnist.validation.images, y_labels: mnist.validation.labels})
               print('step: {}, validation accuracy: {}'.format(i, round(validation_accuracy, 


           # Print the test set accuracy
           print('test accuracy: {}'.format(round(accuracy.eval(feed_dict={
               x: mnist.test.images, y_labels: mnist.test.labels}), 3)))
```

```
step: 0, validation accuracy: 0.11299999803304672
step: 100, validation accuracy: 0.8519999980926514
step: 200, validation accuracy: 0.875
step: 300, validation accuracy: 0.8859999775886536
step: 400, validation accuracy: 0.890999972820282
step: 500, validation accuracy: 0.8970000147819519
step: 600, validation accuracy: 0.902999997138977
step: 700, validation accuracy: 0.902999997138977
step: 800, validation accuracy: 0.9049999713897705
step: 900, validation accuracy: 0.9079999923706055
step: 1000, validation accuracy: 0.9100000262260437
```

```
step: 1100, validation accuracy: 0.9079999923706055
step: 1200, validation accuracy: 0.9110000133514404
step: 1300, validation accuracy: 0.9129999876022339
step: 1400, validation accuracy: 0.9120000004768372
step: 1500, validation accuracy: 0.9120000004768372
step: 1600, validation accuracy: 0.9150000214576721
step: 1700, validation accuracy: 0.9160000085830688
step: 1800, validation accuracy: 0.9160000085830688
step: 1900, validation accuracy: 0.9160000085830688
test accuracy: 0.9150000214576721
```

4 Simple Feed Forward Network for MNIST classification

4.1 Building the network

Exercise:

Create the Fully Connected Feed Forward Network for digits classification, with the following architecture:

1. First layer, that maps one grayscale image (of size 28x28 = 784 pixels) to 100 hidden neurons.
2. Second layer, that maps 100 features into another 100 hidden neurons.
3. Third layer, that maps 100 features into another 100 hidden neurons.
4. Last layer, that maps 100 features to 10 classes, one for each digit.

In every hidden layer, we should use the ReLU activation function. Softmax should be applied on the last layer of our network.

Create placeholders for training data. Remember about a propper shape of training images (in mnist.train.images every digit is a 784D vector) and labels (In training dataset labels are in one-hot-encoding form).

```
In [41]: """ Create a placeholders for MNIST images and their labels """
         x = tf.placeholder(tf.float32, [None, 784])
         y_labels = tf.placeholder(tf.float32, [None, 10])
```

Initialize weight matrices and biases for all layers of our network

```
In [42]: """ Initialize weights matrix W and biases b for the first hidden layer (matrix shoul
         W_1 = tf.get_variable("softmax_W1", dtype=tf.float32, initializer=tf.initializers.trun
         b_1 = tf.get_variable("softmax_b1", dtype=tf.float32, initializer=tf.constant_initiali

         """ Initialize weights matrix W and biases b for the second hidden layer (matrix shou
         W_2 = tf.get_variable("softmax_W2", dtype=tf.float32, initializer=tf.initializers.trun
         b_2 = tf.get_variable("softmax_b2", dtype=tf.float32, initializer=tf.constant_initiali

         """ Initialize weights matrix W and biases b for the third hidden layer (matrix shoul
         W_3 = tf.get_variable("softmax_W3", dtype=tf.float32, initializer=tf.initializers.trun
```

```
b_3 = tf.get_variable("softmax_b3", dtype=tf.float32, initializer=tf.constant_initial

""" Initialize weights matrix W and biases b for the output layer (matrix should map
W_4 = tf.get_variable("softmax_W4", dtype=tf.float32, initializer=tf.initializers.tru
b_4 = tf.get_variable("softmax_b4", dtype=tf.float32, initializer=tf.constant_initial
```

Define network operations In every hidden layer, we should use the ReLU activation function (tf.nn.relu). \ Softmax should be applied on the last layer of our network (tf.nn.softmax).

```
In [43]: """ Define output from the first layer (using W_1 and b_1) """
         y = tf.nn.relu(tf.matmul(x, W_1) + b_1)

         """ Define output from the second layer (using W_2 and b_2) """
         y =  tf.nn.relu(tf.matmul(y, W_2) + b_2)

         """ Define output from the third layer (using W_3 and b_3) """
         y = tf.nn.relu(tf.matmul(y,W_3) + b_3)

         """ Define output from the last layer (using W_4 and b_4) --> this will give us predi
         y = tf.nn.softmax(tf.matmul(y, W_4) + b_4)
```

4.2 Network training

We will now try to train our network. For this purpose we have to define the loss function, cross entropy in this example and optimizer, we will use Adam.

Define the loss function and optimizer This time try to use numerically stable version of softmax, given by a function 'tf.nn.softmax_cross_entropy_with_logits_v2', instead of writing it by hand

```
In [44]: """ Define the cross entropy error between true and predicted labels """
         cross_entropy=tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_labels, logits=y)
         #cross_entropy=-tf.reduce_sum(input_tensor=y_labels*tf.log(y),reduction_indices=[1])
         cross_entropy = tf.reduce_mean(cross_entropy)

In [45]: train_step = tf.train.AdamOptimizer(1e-3).minimize(cross_entropy)
```

Check whether our network returns correct predictions and calculate the accuracy

```
In [46]: """ Create a vector that tells us, whether the predictions from our net - y_conv
             are equal to the correct digit labels - y. """
         correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_labels, 1))
         correct_prediction = tf.cast(correct_prediction, tf.float32)

         """ Calculate the accurracy of correct predictions """
         accuracy = tf.reduce_mean(correct_prediction)
```

Training the network The following code will train our network with using of our predefined Adam optimizer, based on the batch size equal to 64 and with 5000 steps.

```
In [47]: with tf.Session() as sess:
             sess.run(tf.global_variables_initializer())
             for i in range(5000):
               # Get the batch with 64 images from the MNIST training set
               batch = mnist.train.next_batch(64)
               #print(batch_x)
               # Fill data into placeholders
               train_step.run(feed_dict={x: batch[0], y_labels:batch[1]})

               # Print the validation accuracy every 100 steps
               if i % 100 == 0:
                 validation_accuracy = accuracy.eval(feed_dict={
                     x: mnist.validation.images, y_labels: mnist.validation.labels})
                 print('step: {}, validation accuracy: {}'.format(i, round(validation_accuracy, 3


             # Print the test set accuracy
             print('test accuracy: {}'.format(round(accuracy.eval(feed_dict={
                 x: mnist.test.images, y_labels: mnist.test.labels}), 3)))
```

```
step: 0, validation accuracy: 0.10000000149011612
step: 100, validation accuracy: 0.8349999785423279
step: 200, validation accuracy: 0.8920000195503235
step: 300, validation accuracy: 0.8930000066757202
step: 400, validation accuracy: 0.9020000100135803
step: 500, validation accuracy: 0.9049999713897705
step: 600, validation accuracy: 0.925000011920929
step: 700, validation accuracy: 0.9259999990463257
step: 800, validation accuracy: 0.9300000071525574
step: 900, validation accuracy: 0.9330000281333923
step: 1000, validation accuracy: 0.9359999895095825
step: 1100, validation accuracy: 0.9330000281333923
step: 1200, validation accuracy: 0.9300000071525574
step: 1300, validation accuracy: 0.9409999847412109
step: 1400, validation accuracy: 0.9490000009536743
step: 1500, validation accuracy: 0.949999988079071
step: 1600, validation accuracy: 0.9449999928474426
step: 1700, validation accuracy: 0.9490000009536743
step: 1800, validation accuracy: 0.9490000009536743
step: 1900, validation accuracy: 0.9409999847412109
step: 2000, validation accuracy: 0.9549999833106995
step: 2100, validation accuracy: 0.9559999704360962
step: 2200, validation accuracy: 0.9559999704360962
step: 2300, validation accuracy: 0.9430000185966492
step: 2400, validation accuracy: 0.9559999704360962
```

```
step: 2500, validation accuracy: 0.953000009059906
step: 2600, validation accuracy: 0.9559999704360962
step: 2700, validation accuracy: 0.9559999704360962
step: 2800, validation accuracy: 0.9580000042915344
step: 2900, validation accuracy: 0.9599999785423279
step: 3000, validation accuracy: 0.9589999914169312
step: 3100, validation accuracy: 0.9549999833106995
step: 3200, validation accuracy: 0.9589999914169312
step: 3300, validation accuracy: 0.9559999704360962
step: 3400, validation accuracy: 0.9470000267028809
step: 3500, validation accuracy: 0.9549999833106995
step: 3600, validation accuracy: 0.9629999995231628
step: 3700, validation accuracy: 0.9629999995231628
step: 3800, validation accuracy: 0.9639999866485596
step: 3900, validation accuracy: 0.9589999914169312
step: 4000, validation accuracy: 0.9599999785423279
step: 4100, validation accuracy: 0.9570000171661377
step: 4200, validation accuracy: 0.9670000076293945
step: 4300, validation accuracy: 0.9570000171661377
step: 4400, validation accuracy: 0.9679999947547913
step: 4500, validation accuracy: 0.9649999737739563
step: 4600, validation accuracy: 0.9639999866485596
step: 4700, validation accuracy: 0.9649999737739563
step: 4800, validation accuracy: 0.9589999914169312
step: 4900, validation accuracy: 0.9649999737739563
test accuracy: 0.9610000252723694
```

5 Convolutional Neural Network for MNIST classification

5.1 Building the network

Exercise:

Create the CNN for digits classification, with the following architecture:

1. First convolutional layer, that maps one grayscale image to 32 feature maps.
2. Second convolutional layer, that maps 32 feature maps to 64 feature maps.
3. Fully connected layer 1, that maps our 64 feature maps into one layer, with 1024 features.
4. Fully connected layer 2, that maps the 1024 features to 10 classes, one for each digit

Use filters with width and height equal to 5.

At the beggining we have to create some auxiliary functions.

```
In [10]: def get_weight_variable(shape):
             """
             Write a function, that will return the tf.Variable of specified shape, with
             coefficients initialized by random, sampled from a normal distribution with
```

```
    mean = 0 and sd = 0.02
    """
    init = tf.random_normal(shape,mean=0, stddev=0.02)
    return tf.Variable(init)



def conv2d(x, W):
    """
    Write a function, that will return the result of a convolution between
    a tensor x and a weight vector W. We recommend using strides equal to 1
    in every direction and use SAME padding.
    """
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')



def max_pool_2x2(x):
    """
    Write a function, that will return the result of max pooling operation done
    on a tensor x, that will reduce the size of inner image.
    The length and width of a pooling layer window should be equal to 2.
    """
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

Create placeholders for training data. Remember about a propper shape of training images (in mnist.train.images every digit is a 784D vector) and labels (In training dataset labels are in one-hot-encoding form).

```
In [11]: x = tf.placeholder(tf.float32, [None, 784])
         y_l = tf.placeholder(tf.float32, [None, 10])
```

Reshape the x vector into a rank 4 tensor with shapes: [batch_size, rows, columnss, colors/filters]. Keep in mind, that we should have 28x28 image, with only one color (as the image is in grayscale) and that batch size will be given later.

```
In [12]: x_image = tf.reshape(x, [-1, 28, 28, 1]) # [b_size, rows, cols, colors/filters]
```

Create first convolutional layer, that will map one grayscale image into 32 feature maps. To do so, we will use 32 convolutional filters with sizes 5x5x1.

```
In [13]: """ Initialize weights matrix W """
         W_conv1 = get_weight_variable([5, 5, 1, 32]) # 32 filters with size 5x5x1

         """ Initialize biases vector """
         b_conv1 = get_weight_variable([32])
```

```
In [14]: """ Apply convolution operation between image and weights, then add bias and apply re
         y = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1) # [batch_size, 28, 28, 32]
         print(x_image.shape)
         print(W_conv1.shape)
         """ Apply max pooling operation on h_conv1 """
         y = max_pool_2x2(y) # [batch_size, 14, 14, 32]
```

```
(?, 28, 28, 1)
(5, 5, 1, 32)
```

Create second convolutional layer, that will map resulted 32 feature maps into 64 features maps. To do so, we will use 64 convolutional filters with sizes 5x5x32.

```
In [15]: """ Initialize weights matrix W """
         W_conv2 =get_weight_variable([5, 5, 32, 64]) ### 64 filters with size 5x5x32

         """ Initialize biases vector b """
         b_conv2 = get_weight_variable([64]) ###
```

```
In [16]: """ Apply convolution operation between image and weights, then add bias and apply re
         y = tf.nn.relu(conv2d(y, W_conv2) + b_conv2) ###

         """ Apply max pooling operation on h_conv1 """
         y = max_pool_2x2(y) ### [batch_size, 7, 7, 64]
```

Create first fully connected layer -- after 2 rounds of downsampling, our 28x28 image is down to 7x7x64 feature maps -- now map this to 1024 features, with using of fully connected layer, with ReLU activation function and dropout.

```
In [17]: """ Initialize weights matrix W, which maps 7x7x64 feature maps into one layer with 1
         W_fc1 = get_weight_variable([7*7*64, 1024])

         """ Initialize biases vector b """
         b_fc1 = get_weight_variable([1024])

         """ Reshape the result from the last convolutional layer from [batch_size, 7, 7, 64]
             as this is the shape that is expected by the weight matrix W_fc1. """
         print(y)
         y_flat = tf.reshape(y, [-1,7*7*64])
         print(y_flat)
```

```
Tensor("MaxPool_9:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("Reshape_9:0", shape=(?, 3136), dtype=float32)
```

```
In [18]: """ Apply matrix multiplication between image and weights, then add bias and apply re
         h_fc1 = tf.nn.relu(tf.matmul(y_flat, W_fc1) + b_fc1)
```

```
In [19]: """ Create placeholder for the dropout probability """
         keep_prob = tf.placeholder(tf.float32)

         """ Apply dropout with probability = keep_prob """
         h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

Create the second convolutional layer, that maps 1024 features from the last layer into 10 classes, one for each digit. The softmax function applied on the result of this layer *y_conv* will give us the probabilities for every class that our convolutional neural network gives for the given image. Although we won't use this function at the moment, as this could be numerically unstable. We will handle this problem during the training step. At the moment we want to keep only logits.

```
In [20]: """ Initialize weights matrix W, which maps 1024 features into one layer with 10 feat
         W_fc2 = get_weight_variable([1024, 10])
         print(W_fc2)

         """ Initialize biases vector b """
         b_fc2 = get_weight_variable([10])
         print(b_fc2)

<tf.Variable 'Variable_38:0' shape=(1024, 10) dtype=float32_ref>
<tf.Variable 'Variable_39:0' shape=(10,) dtype=float32_ref>
```

```
In [21]: """ Apply matrix multiplication between image and weights, then add bias to get logit
         print(h_fc1_drop)
         print(W_fc2)
         y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

Tensor("dropout_4/mul:0", shape=(?, 1024), dtype=float32)
<tf.Variable 'Variable_38:0' shape=(1024, 10) dtype=float32_ref>
```

5.2 Network training

We will now try to train our network. For this purpose we have to define the loss function, cross entropy in this example and optimizer, we will use Adam.

Define the loss function and optimizer

```
In [28]: """ Define the cross entropy loss function
             Remember that in the return from our network - 'y_conv', we didn't use the softma
             with numerical stability, that's why we should use now the function named softmax
             is more numerical stable. """
         #cross_entropy=-tf.reduce_sum(input_tensor=y_l*tf.log(y_conv),reduction_indices=[1])
         cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_l, logits=y_conv)
         cross_entropy = tf.reduce_mean(cross_entropy)

In [29]: """ Define the Adam optimizer with parameter equal to 1e-3, that will minimize our cr
         train_step = tf.train.AdamOptimizer(1e-3).minimize(cross_entropy)
```

Check whether our network returns correct predictions and calculate the accuracy

```
In [30]: """ Create a vector that tells us, whether the predictions from our net - y_conv
             are equal to the correct digit labels - y. """
         correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_l, 1))
         correct_prediction = tf.cast(correct_prediction, tf.float32)

         """ Calculate the accurracy of correct predictions """
         accuracy = tf.reduce_mean(correct_prediction)
```

Training the network The following code will train our network with using of our predefined Adam optimizer, based on the batch size equal to 64 and with 5000 steps.

```
In [36]: with tf.Session() as sess:
           sess.run(tf.global_variables_initializer())
           for i in range(5000):
             batch = mnist.train.next_batch(64)
             if i % 100 == 0:
                 validation_accuracy = accuracy.eval(feed_dict={
                   x: mnist.validation.images, y_l: mnist.validation.labels, keep_prob: 1.0})
                 print('step: {}, validation accuracy: {}'.format(i, round(validation_accuracy
                 train_step.run(feed_dict={x: batch[0], y_l: batch[1], keep_prob: 0.5})
           print('test accuracy: {}'.format(round(accuracy.eval(feed_dict={
               x: mnist.test.images, y_l: mnist.test.labels, keep_prob: 1.0}),3)))
```

```
step: 0, validation accuracy: 0.0989999994635582
step: 100, validation accuracy: 0.10999999940395355
step: 200, validation accuracy: 0.16599999368190765
step: 300, validation accuracy: 0.1809999942779541
step: 400, validation accuracy: 0.15700000524520874
step: 500, validation accuracy: 0.19900000095367432
step: 600, validation accuracy: 0.21799999475479126
step: 700, validation accuracy: 0.2800000011920929
step: 800, validation accuracy: 0.25
step: 900, validation accuracy: 0.40400001406669617
step: 1000, validation accuracy: 0.4300000071525574
step: 1100, validation accuracy: 0.41100001335144043
step: 1200, validation accuracy: 0.44600000977516174
step: 1300, validation accuracy: 0.503000020980835
step: 1400, validation accuracy: 0.4930000071525574
step: 1500, validation accuracy: 0.5070000290870667
step: 1600, validation accuracy: 0.5189999938011169
step: 1700, validation accuracy: 0.5509999990463257
step: 1800, validation accuracy: 0.5849999785423279
step: 1900, validation accuracy: 0.6079999804496765
step: 2000, validation accuracy: 0.574999988079071
step: 2100, validation accuracy: 0.5879999995231628
step: 2200, validation accuracy: 0.5979999899864197
```

```
step: 2300, validation accuracy: 0.6110000014305115
step: 2400, validation accuracy: 0.6309999823570251
step: 2500, validation accuracy: 0.6370000243186951
step: 2600, validation accuracy: 0.628000020980835
step: 2700, validation accuracy: 0.6340000033378601
step: 2800, validation accuracy: 0.6389999985694885
step: 2900, validation accuracy: 0.6439999938011169
step: 3000, validation accuracy: 0.640999972820282
step: 3100, validation accuracy: 0.6259999871253967
step: 3200, validation accuracy: 0.6150000095367432
step: 3300, validation accuracy: 0.6060000061988831
step: 3400, validation accuracy: 0.6129999756813049
step: 3500, validation accuracy: 0.628000020980835
step: 3600, validation accuracy: 0.6539999842643738
step: 3700, validation accuracy: 0.6800000071525574
step: 3800, validation accuracy: 0.6869999766349792
step: 3900, validation accuracy: 0.6930000185966492
step: 4000, validation accuracy: 0.6959999799728394
step: 4100, validation accuracy: 0.7039999961853027
step: 4200, validation accuracy: 0.7049999833106995
step: 4300, validation accuracy: 0.7279999852180481
step: 4400, validation accuracy: 0.7480000257492065
step: 4500, validation accuracy: 0.7269999980926514
step: 4600, validation accuracy: 0.7120000123977661
step: 4700, validation accuracy: 0.7120000123977661
step: 4800, validation accuracy: 0.7329999804496765
step: 4900, validation accuracy: 0.7480000257492065
test accuracy: 0.7490000128746033
```