# TENSORFLOW

GABRIELA CZARSKA

CZYM JEST TENSORFLOW?

INSTALACJA

PODSTAWY

MNIST

# CZYM JEST TENSORFLOW

- Open sourcowa biblioteka do obliczeń numerycznych
- Stworzona przez programistów z Google Brain
- Pozwala na prowadzenie obliczeń zarówno na CPU jak i GPU
- Powstała głównie do Machine Learningu i Deep Learningu, ale pozwala na zastosowanie w wielu innych dziedzinach


airbnb


NVIDIA


UBER


SAP


kakao


DeepMind


Dropbox


ebay


Google


intel


Coca-Cola


ZTE


QUALCOMM

# INSTALACJA

(UŻYWAJĄC VIRTUALENVA)

- $ mkdir tensorflow
- $ virtualenv --system-site-packages tensorflow
- $ source tensorflow/bin/activate
- $ pip install --upgrade tensorflow-gpu

# SPRAWDŹ CZY DZIAŁA

```
# Python
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

# THE COMPUTATIONAL GRAPH

- Budowanie grafu obliczeń
- Odpalanie grafu obliczeń

# GRAF OBLICZEŃ

Ciąg operacji reprezentowany jako wierzchołki grafu. Każdy wierzchołek bierze jeden lub więcej tensorów na wejściu i zwraca tensor

# TENSOR

- Kluczowa jednostka w Tensorflow
- Zbiór wartości ułożonych w dowolnie wymiarową tablice
- *Rank - wymiar*

```
3 # a rank 0 tensor; a scalar with shape []
[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

```
[3.0, 4.0]
```

```
from __future__ import print_function
node3 = tf.add(node1, node2)
print("node3:", node3)
print("sess.run(node3):", sess.run(node3))
```

```
node3: Tensor("Add:0", shape=(), dtype=float32)
sess.run(node3): 7.0
```

# PLACEHOLDERY

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b  # + provides a shortcut for tf.add(a, b)
```

```
print(sess.run(adder_node, {a: 3, b: 4.5}))
print(sess.run(adder_node, {a: [1, 3], b: [2, 4]}))
```

```
7.5
[ 3.  7.]
```

```
add_and_triple = adder_node * 3.
print(sess.run(add_and_triple, {a: 3, b: 4.5}))
```

```
22.5
```

# VARIABLES

```
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W*x + b
```

```
init = tf.global_variables_initializer()
sess.run(init)
```

```
print(sess.run(linear_model, {x: [1, 2, 3, 4]}))
```

```
[ 0.          0.30000001  0.60000002  0.90000004]
```

# FUNKCJA STRATY

- Mierzy jak daleko od prawidłowej odpowiedzi jesteśmy
- Regresja liniowa
- Będziemy ją minimalizować

```
y = tf.placeholder(tf.float32)
squared_deltas = tf.square(linear_model - y)
loss = tf.reduce_sum(squared_deltas)
print(sess.run(loss, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]}))
```

```
23.66
```

# OPTIMIZER

GRADIENT DESCENT

- Będzie powoli modyfikował zmienne tak aby zminimalizować funkcje straty

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

```
sess.run(init) # reset values to incorrect defaults.
for i in range(1000):
  sess.run(train, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]})

print(sess.run([W, b]))
```

```
[array([-0.9999969], dtype=float32), array([ 0.99999082], dtype=float32)]
```

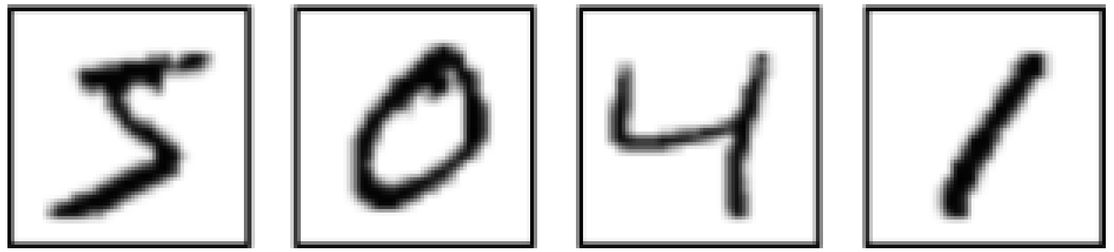
```
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W*x + b
y = tf.placeholder(tf.float32)

# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

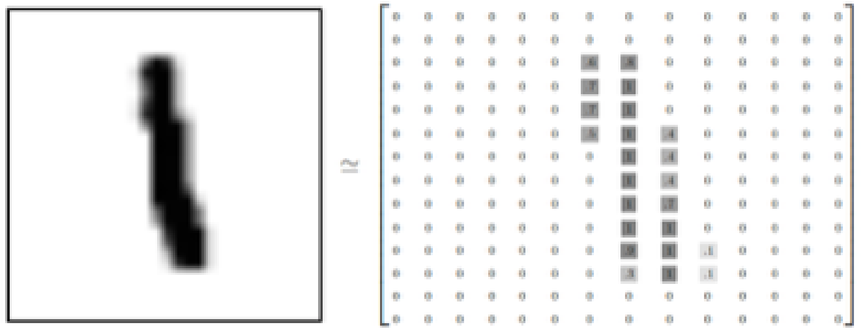
```
# training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
  sess.run(train, {x: x_train, y: y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x: x_train, y: y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```

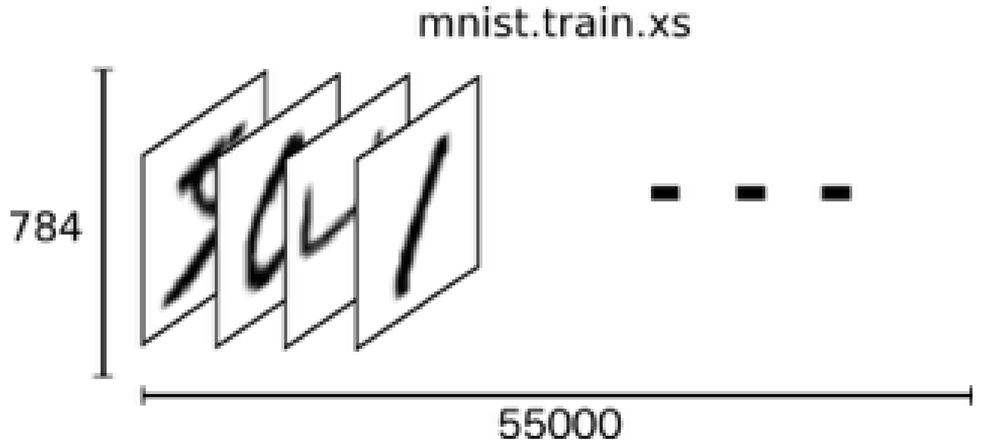
```
W: [-0.9999969] b: [ 0.99999082] loss: 5.69997e-11
```

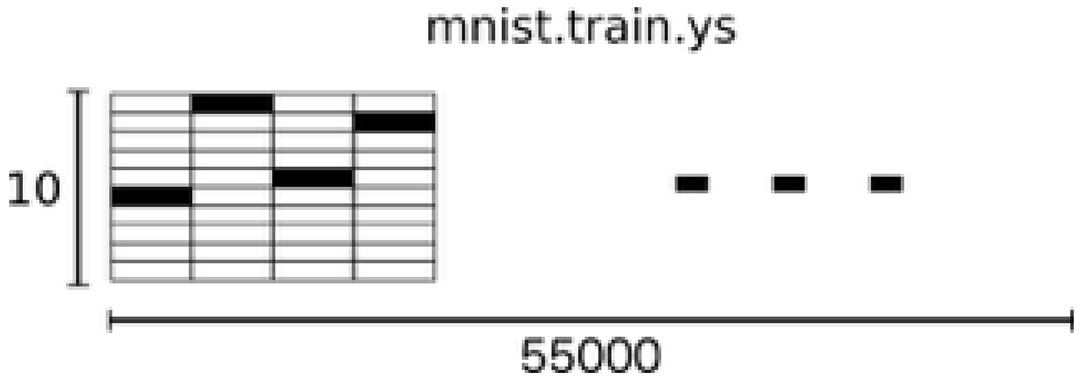
# MNIST

ROZPOZNAWANIE RĘCZNIE NAPISANYCH CYFR






mnist.train.xs
784
55000


mnist.train.ys
10
55000

```
"""A very simple MNIST classifier.

See extensive documentation at
https://www.tensorflow.org/get_started/mnist/beginners
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

FLAGS = None
```

```
def main(_):
  # Import data
  mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss and optimizer
  y_ = tf.placeholder(tf.float32, [None, 10])
```

```
# The raw formulation of cross-entropy,
#
#   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
#                                 reduction_indices=[1]))
#
# can be numerically unstable.
#
# So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
# outputs of 'y', and then average across the batch.
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

```
  # Test trained model
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  print(sess.run(accuracy, feed_dict={x: mnist.test.images,
                                      y_: mnist.test.labels}))

if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/mnist/input_data',
                      help='Directory for storing input data')
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

# LINKI

- https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/examples/tutorials/mnist/mnist_softmax.py
- https://www.tensorflow.org/get_started/mnist/beginners
- http://neuralnetworksanddeeplearning.com/chap3.html#softmax
- http://colah.github.io/posts/2015-08-Backprop/