TensorFlow

The Big Picture

Index

1. Objectives
2. Deep Learning
3. Understanding TensorFlow
4. ‘Hello World!’

https://github.com/gergf/TensorFlowBDSlab

Index

1. **Objectives**
2. Deep Learning
3. Understanding TensorFlow
4. ‘Hello World!’

Objectives

- We are going to address the next questions:
  - What kind of problems are we going to deal with?
  - Why do we need tools such a TF?
  - How to use TF? First steps...

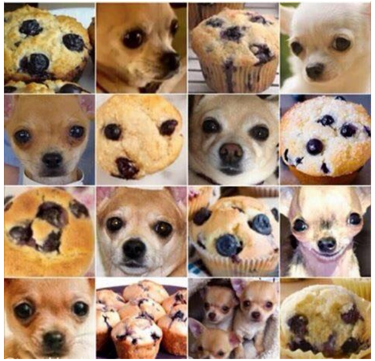
Index

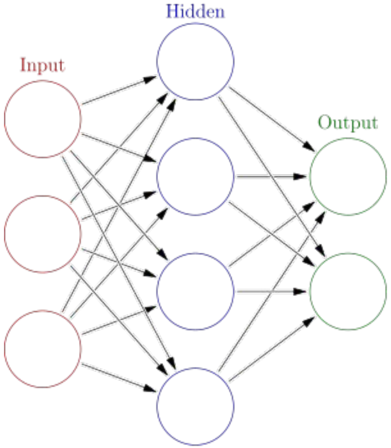
1. Objectives
2. **Deep Learning**
3. Understanding TensorFlow
4. ‘Hello World!’

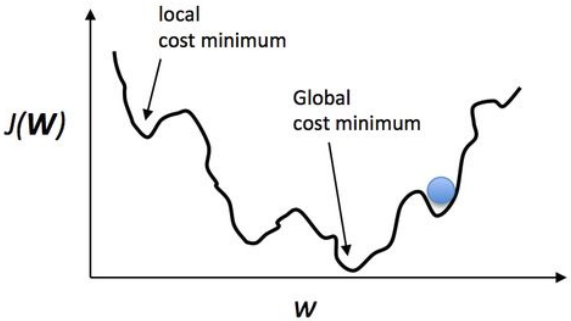
Deep Learning

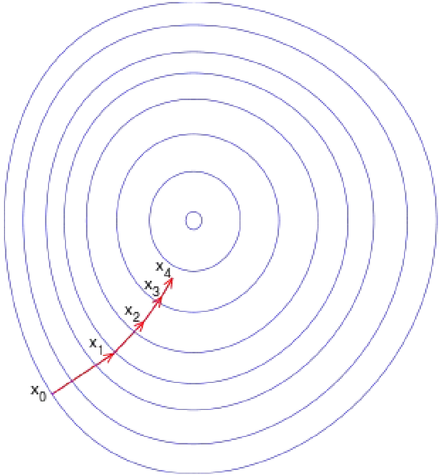
- Deep Learning algorithms can be described as an instance of a simple recipe:
  - Dataset (MNIST, CIFAR, BRATS…)
  - Model (ANN, ConvNN, RNN…)
  - Cost Function (MSE, Cross Entropy, DICE…)
  - Optimization Procedure (1st order GD/LM, 2nd order Newton…)

Deep Learning









X^m $\quad F : (X^m, W) \mapsto Y_p^n \quad J : (Y_t^n, Y_p^n) \mapsto \mathbb{R} \quad \frac{\partial J}{\partial W}$

Dataset → Model ⇄ Cost Function ⇄ Optimization

Deep Learning

X^m $\quad F : (X^m, W) \mapsto Y_p^n \quad J : (Y_t^n, Y_p^n) \mapsto \mathbb{R} \quad \frac{\partial J}{\partial W}$

¡Del orden de Millones! (con suerte)

¡¡Del orden de cientos de Millones!!

¡¡¡La derivada para cada uno de los pesos!!!

¡¡¡¡Y encima es un proceso ITERATIVO!!!!

Dataset → Model ⇄ Cost Function ⇄ Optimization

Index

1. Objectives
2. Deep Learning
3. **Understanding TensorFlow**
4. ‘Hello World!’

Understanding TensorFlow

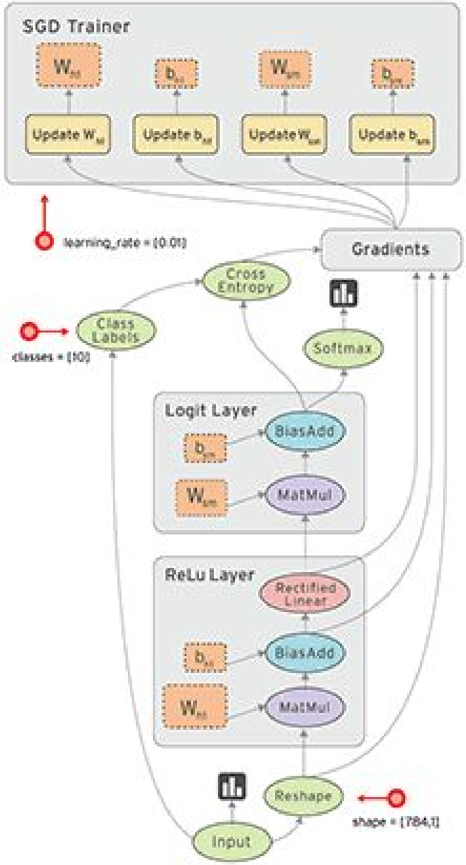
TensorFlow allows us to handle easily the computational cost and the calculations of the derivatives.

- TF works with Tensors, and all the tensors build a **computational graph.**
- TF uses **automatic differentiation** in order to calculate the derivatives of each parameter. Therefore, each node of the graph knows how to compute its own gradient.

Understanding TensorFlow

We might think TF programs as consisting of two sections:

1. Building the computational graph
2. Running the graph.



Understanding TensorFlow

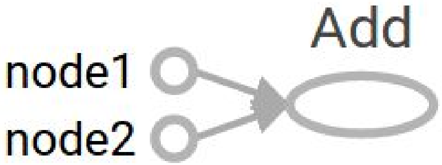
1_example.py

The Tensor primitive (Extension of *Eigen* Tensor class)

```
3 # a rank 0 tensor; this is a scalar with shape []
[1., 2., 3.] # a rank 1 tensor; this is 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]
```

Building the computational graph: Tensors and Operations

```
import tensorflow as tf

node1 = tf.constant(3.0, dtype=tf.float32, name='node1')
node2 = tf.constant(4.0, dtype=tf.float32, name='node2')
node3 = tf.add(node1, node2)

print(node1, node3)

```



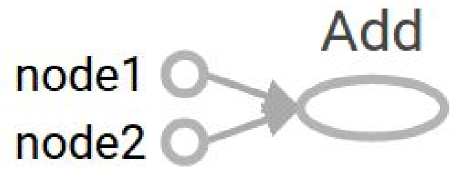

```
(ibime) ger@devon:~/Desktop/TF$ python example1.py
Tensor("node1:0", shape=(), dtype=float32) Tensor("Add:0", shape=(), dtype=float32)
```

Understanding TensorFlow

1_example.py

Running the computational graph

```python
import tensorflow as tf

node1 = tf.constant(3.0, dtype=tf.float32, name='node1')
node2 = tf.constant(4.0, dtype=tf.float32, name='node2')
node3 = tf.add(node1, node2)

print(node1, node3)

```

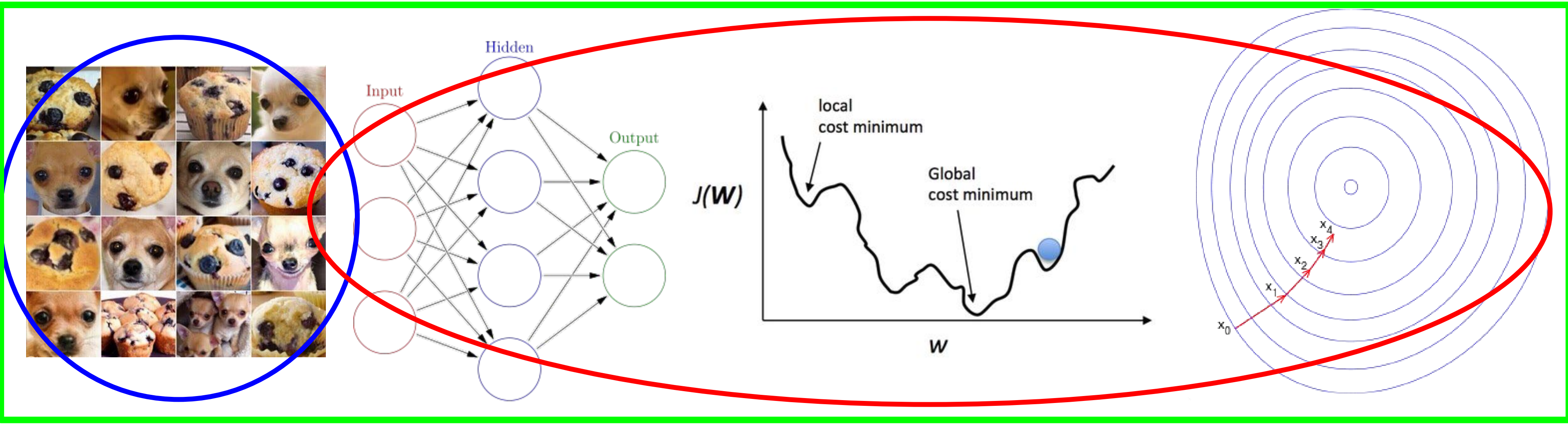



```python
sess = tf.Session()
result = sess.run([node3])
print('Result:', result)
```

```
Result: [7.0]
(ibime) ger@devon:~/Desktop/TF$
```

QUIERO ESO

Understanding TensorFlow



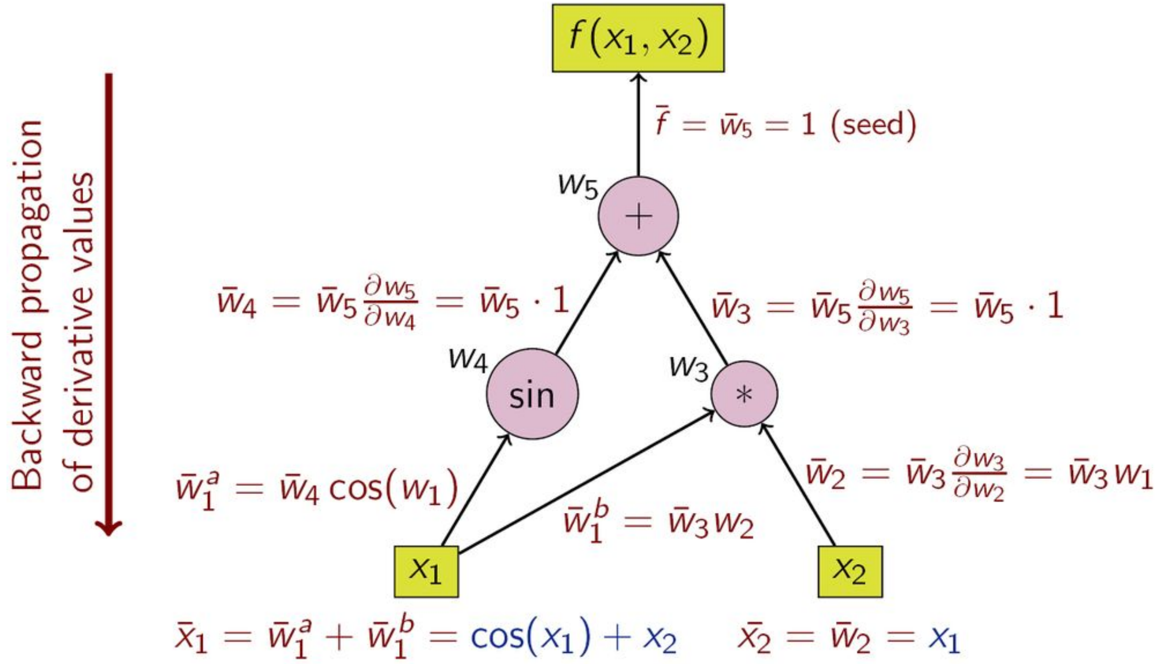
1) Load the data

2) Define the model Building the computational graph

3) Train the model running the graph (Session)

Understanding TensorFlow

What does it mean **automatic differentiation**?

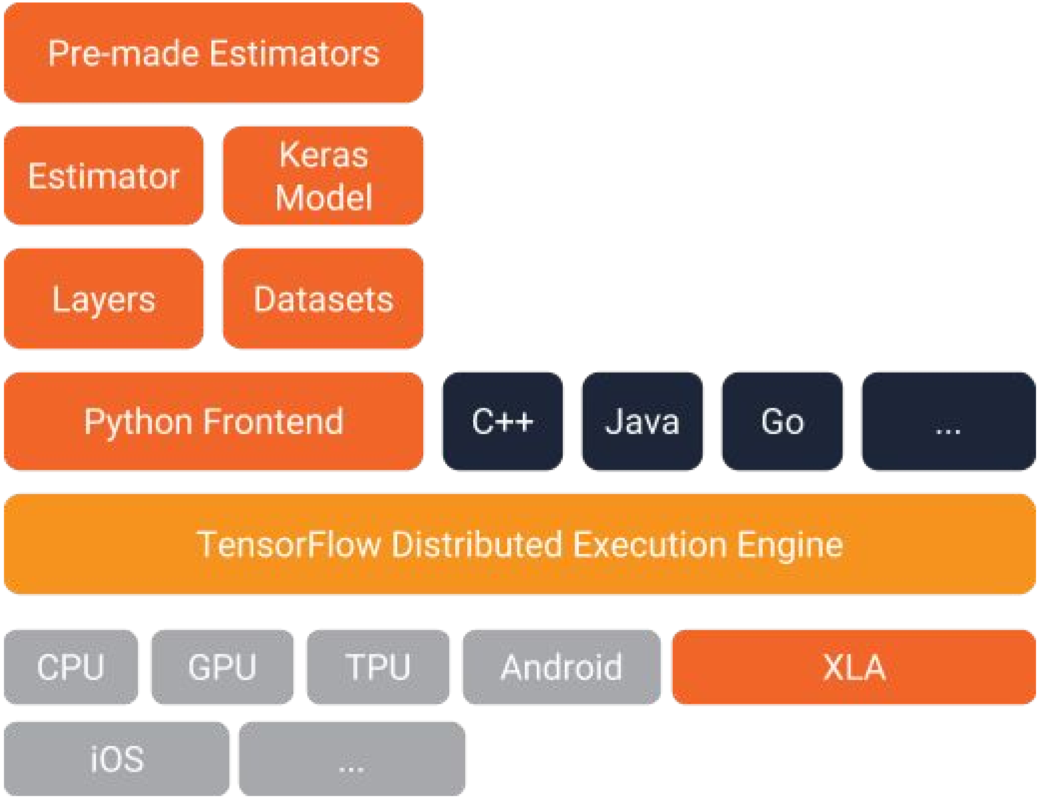


Contents

Understanding TensorFlow

Summary:

- TensorFlow is a low level library that allow us to implement a bunch of DeepLearning models efficiently.

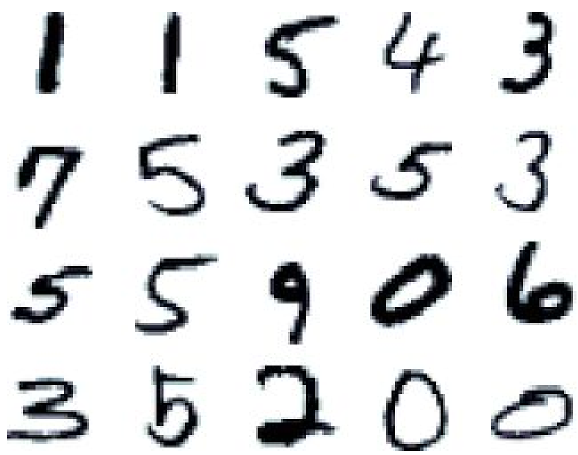


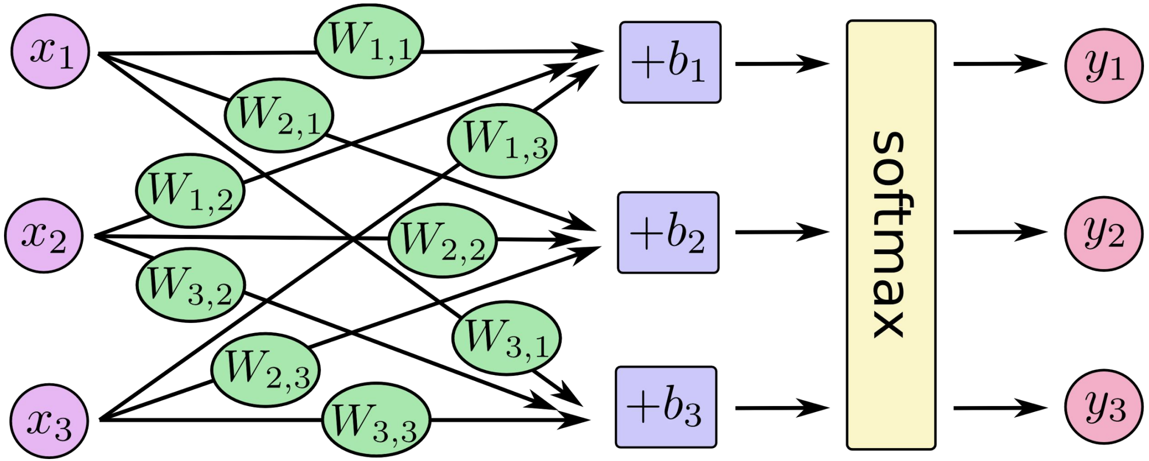
Index

1. Objectives
2. Deep Learning
3. Understanding TensorFlow
4. **'Hello World!'**

'Hello World!'

2_example.py

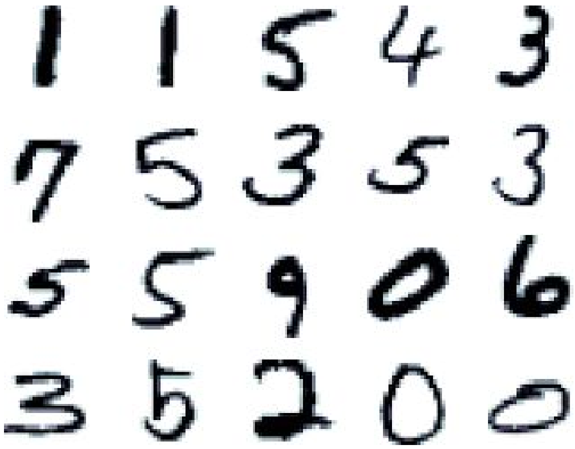




$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)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

‘Hello World!’

2_example.py



$$y_p = softmax(W \cdot x + b)$$

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

```python
# Load the data #
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
x_train, y_train = mnist.train.next_batch(batch_size)
```

```python
# Declare parameters #
weights = {
    'w1': tf.Variable(tf.random_normal([784, 10]))
    }
biases = {
    'b1': tf.Variable(tf.random_normal([10]))
    }
```

‘Hello World!’

2_example.py

Cost Function

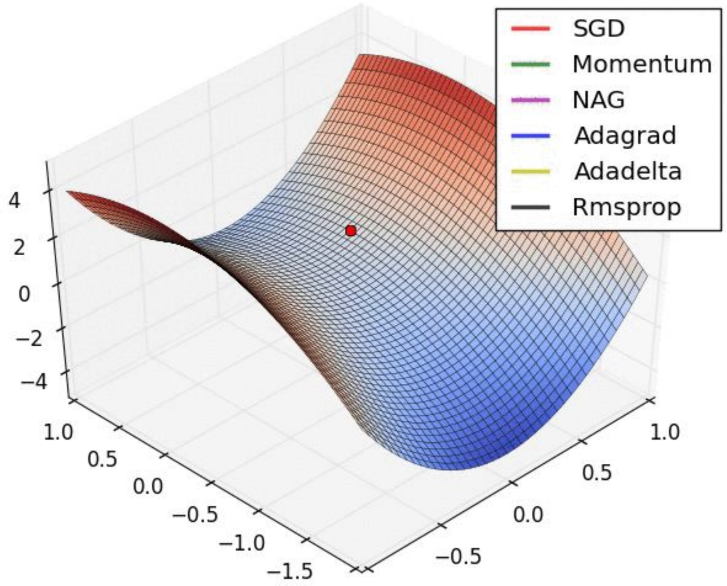
$$\begin{array}{ccc}\hat{\mathbf{y}} & & \mathbf{y}\\ \begin{bmatrix}0.1\\0.5\\0.4\end{bmatrix} & D(\hat{\mathbf{y}},\mathbf{y}) = -\sum_j y_j \ln \hat{y}_j & \begin{bmatrix}0\\1\\0\end{bmatrix}\end{array}$$

```
# Cost Function. Compare predictions with true labels
cross_entropy = tf.reduce_mean(- y_pred * tf.log(y_in))
```

‘Hello World!’

2_example.py

Optimization Procedure




```python
# Optimization
optimizer  = tf.train.GradientDescentOptimizer(learning_rate=0.5)
gradients  = optimizer.compute_gradients(cross_entropy)
train_step = optimizer.apply_gradients(gradients)
```

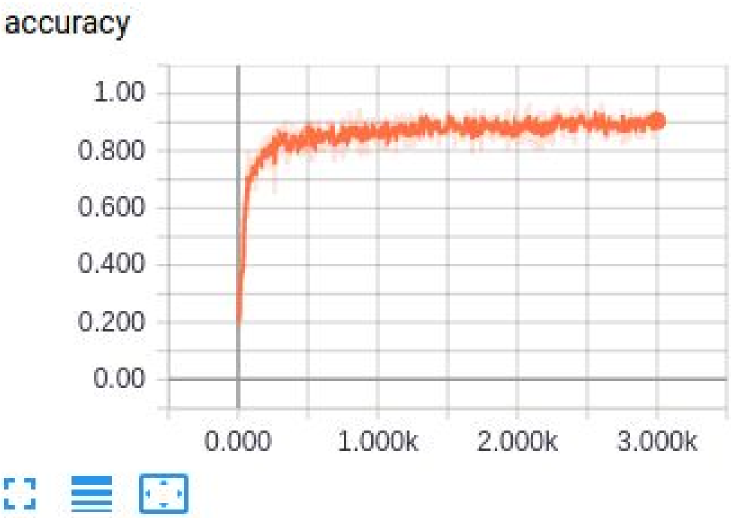
‘Hello World!’

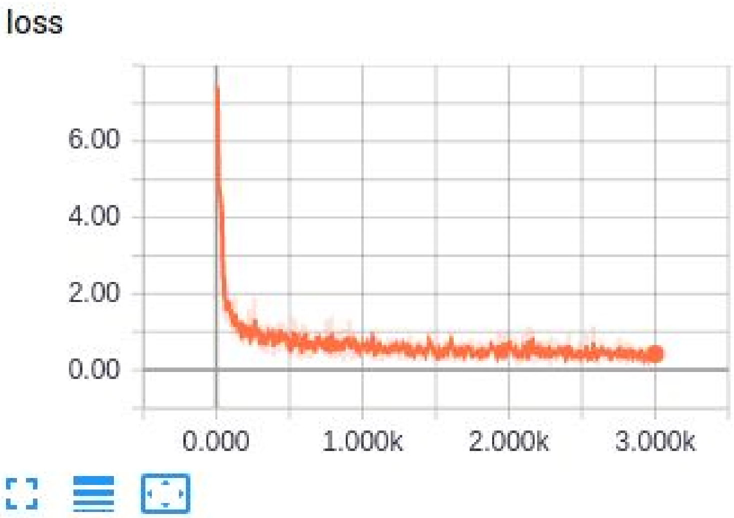
Running computational graph

```
#######################################################################
############### RUNNING THE COMPUTATIONAL GRAPH ###########
#######################################################################
with tf.Session() as sess:
    writer = tf.summary.FileWriter('./2e', sess.graph)  # TensorBoard writer
    # We need to initialize the variables
    tf.global_variables_initializer().run()

    # Train the model
    for i in range(max_iterations):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        _, loss, summ = sess.run([train_step, cross_entropy, merged],
                                feed_dict={x_in: batch_xs, y_in: batch_ys})
        writer.add_summary(summ, i)
```

‘Hello World!’

2_example.py



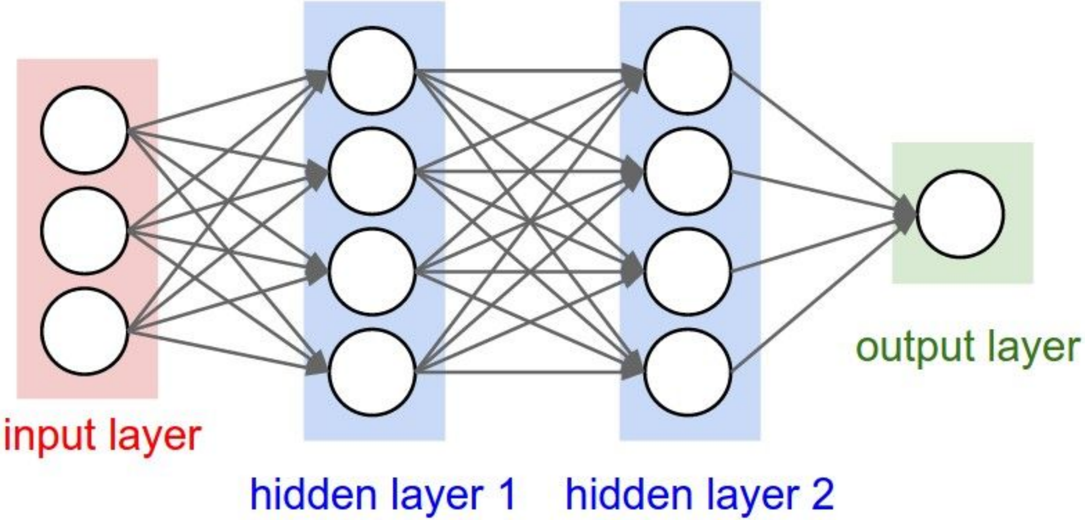


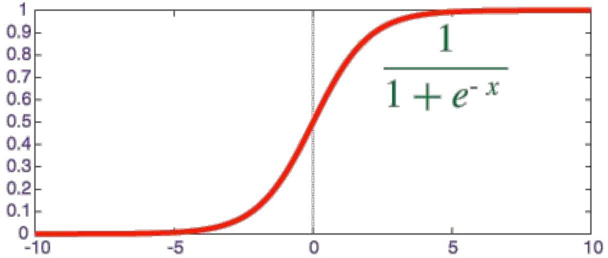
ACCURACY: 89,73%

‘Hello World!’

3_example.py

Let’s go deeper...





‘Hello World!’

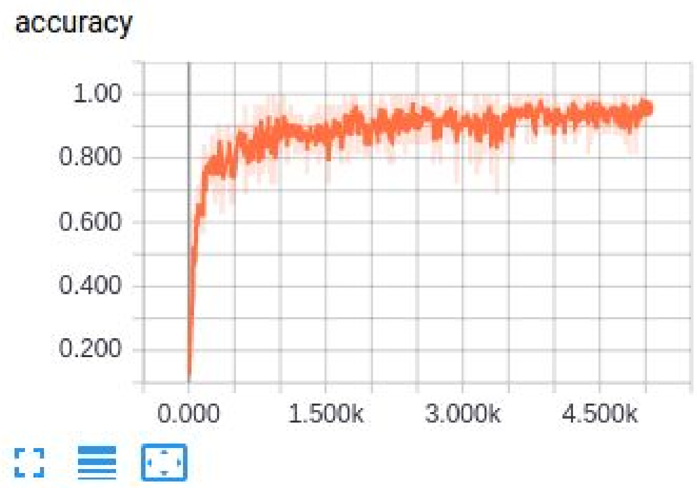
3_example.py

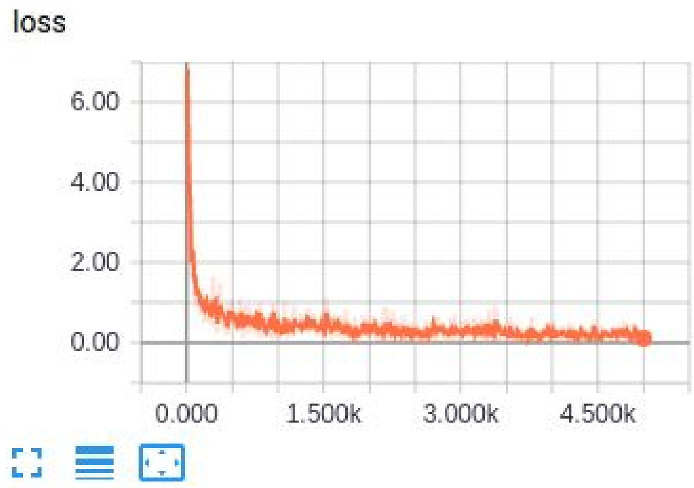
```
# Declare parameters #
weights = {
    'w1': tf.Variable(tf.random_normal([784, 256])),
    'w2': tf.Variable(tf.random_normal([256, 128])),
    'w3': tf.Variable(tf.random_normal([128, 10]))
    }
biases = {
    'b1': tf.Variable(tf.random_normal([256])),
    'b2': tf.Variable(tf.random_normal([128])),
    'b3': tf.Variable(tf.random_normal([10]))
    }

```

```
# Inference. Compute everything until the predictions
y = tf.nn.sigmoid(tf.matmul(x_in, weights['w1']) + biases['b1']) # INPUT LAYER
y = tf.nn.sigmoid(tf.matmul(y, weights['w2']) + biases['b2'])    # HIDDEN LAYER 1
y = tf.matmul(y, weights['w3']) + biases['b3']                   # HIDDEN LAYER 2
y_pred = tf.nn.softmax(y)                                        # PREDICTION
```

‘Hello World!’

3_example.py



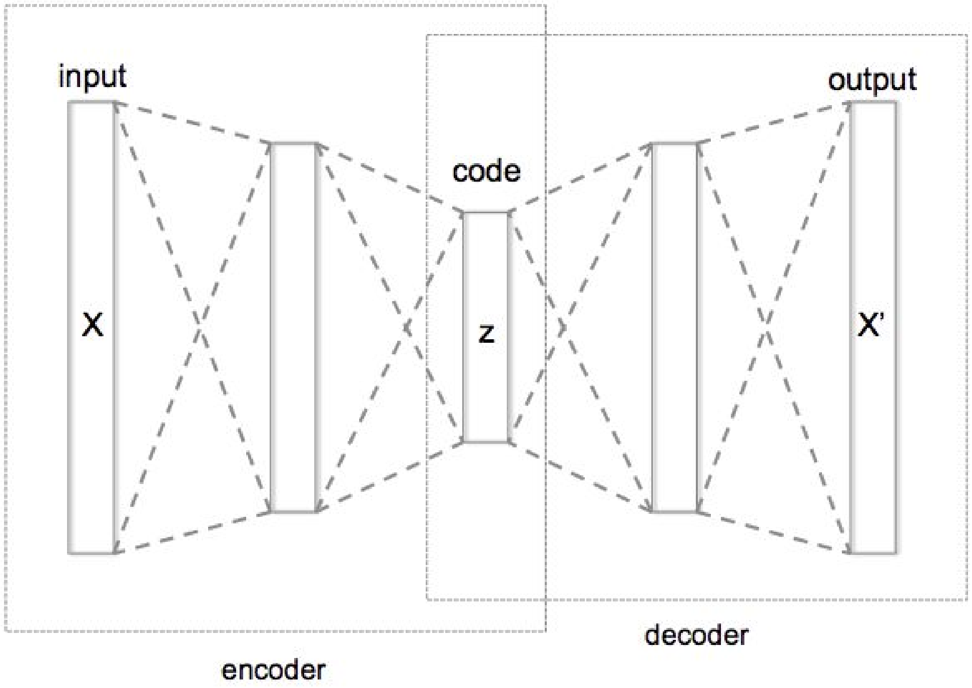


ACCURACY: 92,90%

‘Hello World!’

4_example.py

We can build autoencoders too!



‘Hello World!’

4_example.py

```
# Declare parameters #
weights = {
        'encoder_w1': tf.Variable(tf.random_normal([784, 256])),
        'encoder_w2': tf.Variable(tf.random_normal([256, 64])),
        'decoder_w1': tf.Variable(tf.random_normal([64, 256])),
        'decoder_w2': tf.Variable(tf.random_normal([256, 784]))
        }
biases = {
        'encoder_b1': tf.Variable(tf.random_normal([256])),
        'encoder_b2': tf.Variable(tf.random_normal([64])),
        'decoder_b1': tf.Variable(tf.random_normal([256])),
        'decoder_b2': tf.Variable(tf.random_normal([784]))
        }
```

‘Hello World!’

4_example.py

```python
# Building the encoder
def encoder(x):
    # Encoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_w1']),
                                   biases['encoder_b1']))
    # Encoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_w2']),
                                   biases['encoder_b2']))
    return layer_2


# Building the decoder
def decoder(x):
    # Decoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_w1']),
                                   biases['decoder_b1']))
    # Decoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_w2']),
                                   biases['decoder_b2']))
    return layer_2
```

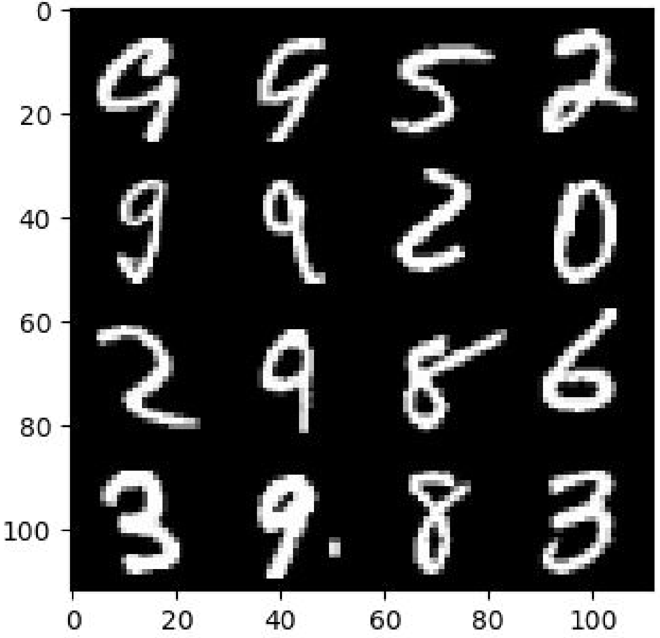
‘Hello World!’

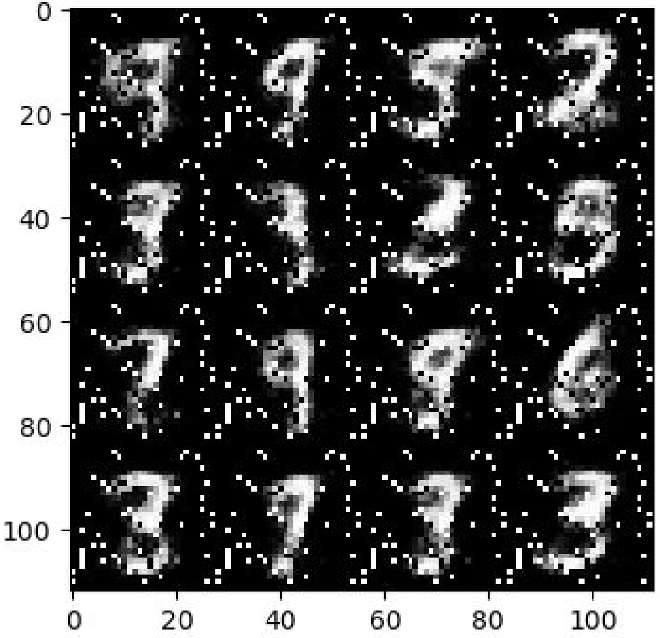
4_example.py

```
# Inputs
x_in = tf.placeholder(tf.float32, [None, 784])

# Inference. Compute everything until the predictions
encoder_op = encoder(x_in)
decoder_op = decoder(encoder_op)
y_pred     = decoder_op

# Cost Function. Compare predictions with true labels
# In this case with use the L2 distance
loss = tf.reduce_mean(tf.pow(x_in - y_pred, 2))

# Optimization
optimizer  = tf.train.RMSPropOptimizer(learning_rate=0.01)
gradients  = optimizer.compute_gradients(loss)
train_step = optimizer.apply_gradients(gradients)

```

‘Hello World!’

4_example.py





Done!

Caffe

theano

PYTORCH

TensorFlow™

DL4J

DEEPLEARNING4J