



**Barcelona  
Supercomputing  
Center**

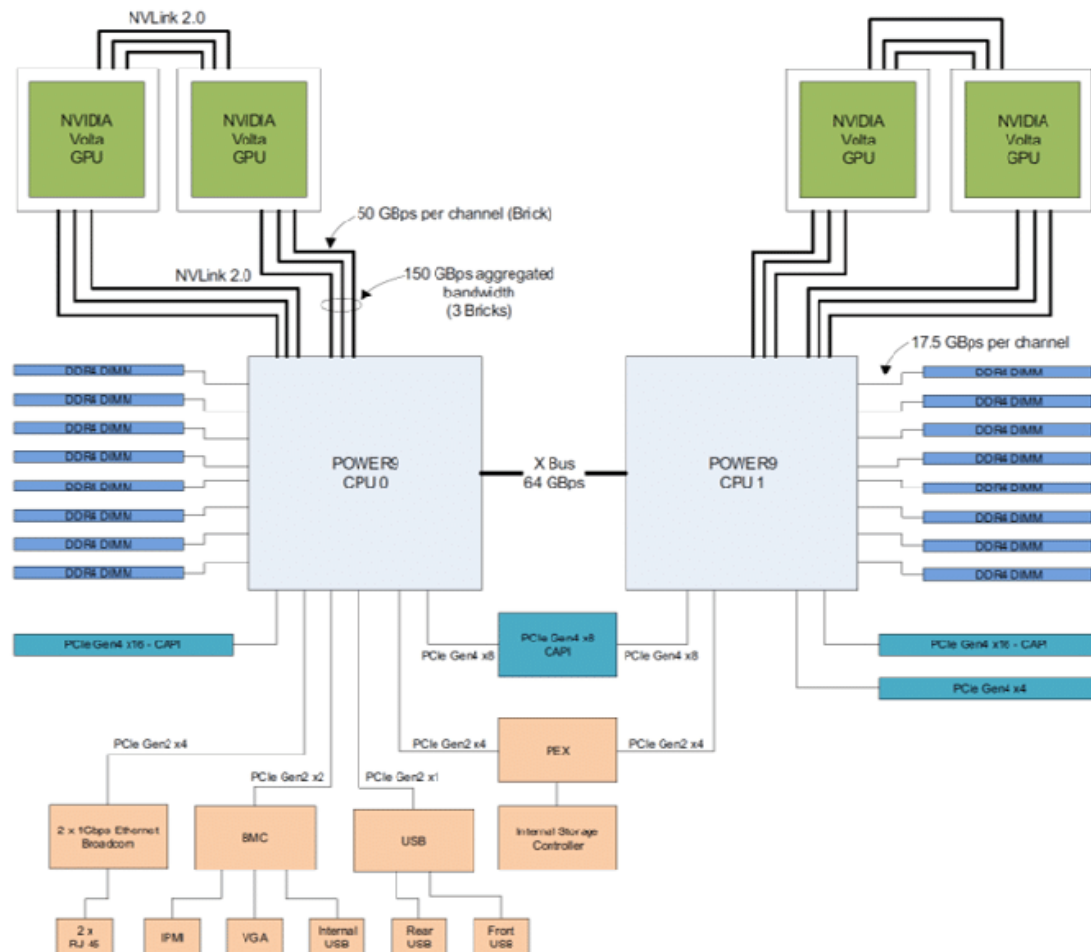
*Centro Nacional de Supercomputación*

# **Guided Laboratory on High Performance Computing Aspects of Deep Learning**

Marc Casas ([marc.casas@bsc.es](mailto:marc.casas@bsc.es))

# The Power9 Cluster

- It is a heterogeneous CPU-GPU cluster.
- Its main computational power is provided by NVIDIA GPUS.
  - 2 x IBM Power9 8335-GTH @ 2.4GHz
    - 3.0GHz on turbo,
    - 20 cores and 4 threads/core, total 160 threads per node.
  - 512GB of main memory distributed in 16 dimms x 32GB @ 2666MHz
  - 4 x GPU NVIDIA V100 (Volta) with 16GB HBM2.



# Setting Up the Environment

- ❧ ssh to Power9: “ssh plogin1.bsc.es”
- ❧ Pick up the files of this lab and put them in a folder
  - Decompress and untar the file.
- ❧ Use “sbatch script2launch.sh” to submit jobs to the queues.

# Setting Up the Environment

[illegible]

# TensorFlow Basics

- ⌘ The central unit of data in TensorFlow (TF) is the **tensor**
- ⌘ A tensor consists of a set of primitive values shaped into an array of any number of dimensions
- ⌘ A tensor's **rank** is its number of dimensions.

`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]`

- ⌘ TF codes are composed of operations on tensors
- ⌘ The sequences can be conceived as a computational graph
- ⌘ TF codes carry out two discrete operations:
  - Building the computational graph
  - Running the computational graph

# First TensorFlow example: Optimization Problem

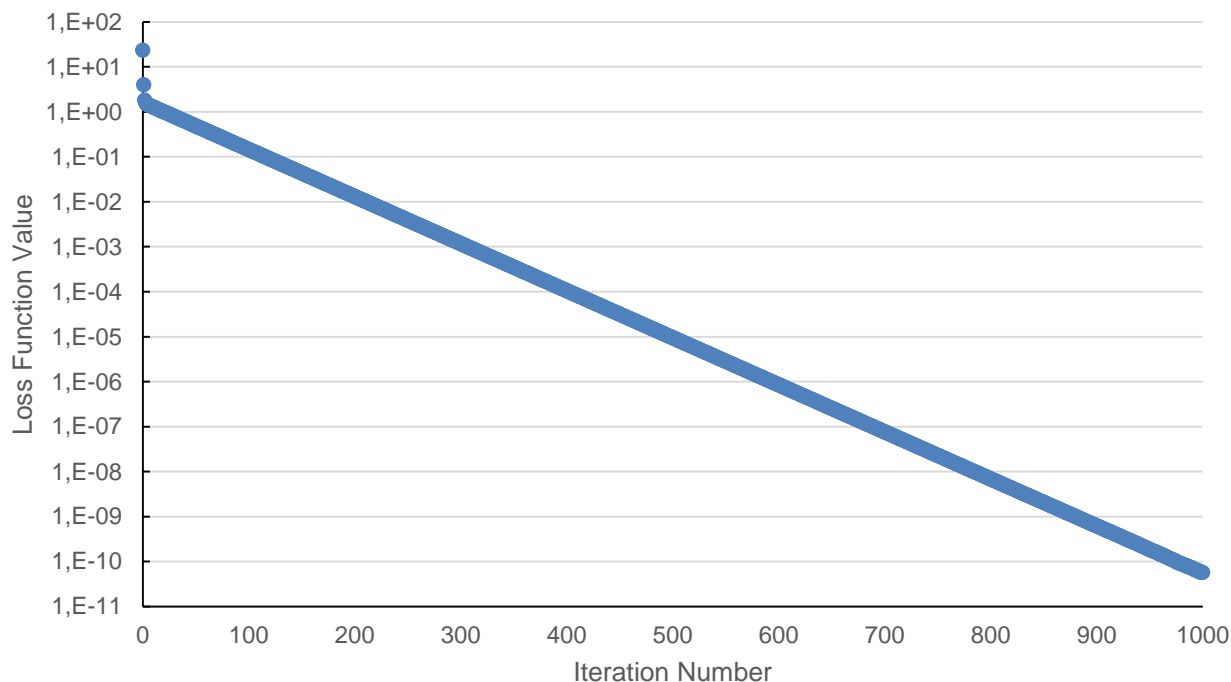
## Goal:

- Find the linear model  $y = Wx + b$  that better fits a set of points

## Optimization Method:

- GradientDescentOptimizer with a 0.01 step

## Result:



# First TensorFlow example: Optimization Problem

```
(bsc28069) mn1.bsc.es — Konsole
File Edit View Bookmarks Settings Help
#!/usr/bin/env python
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

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))

for i in range(1000):
    sess.run(train, {x: x_train, y: y_train})
    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))
```

1,1 Top

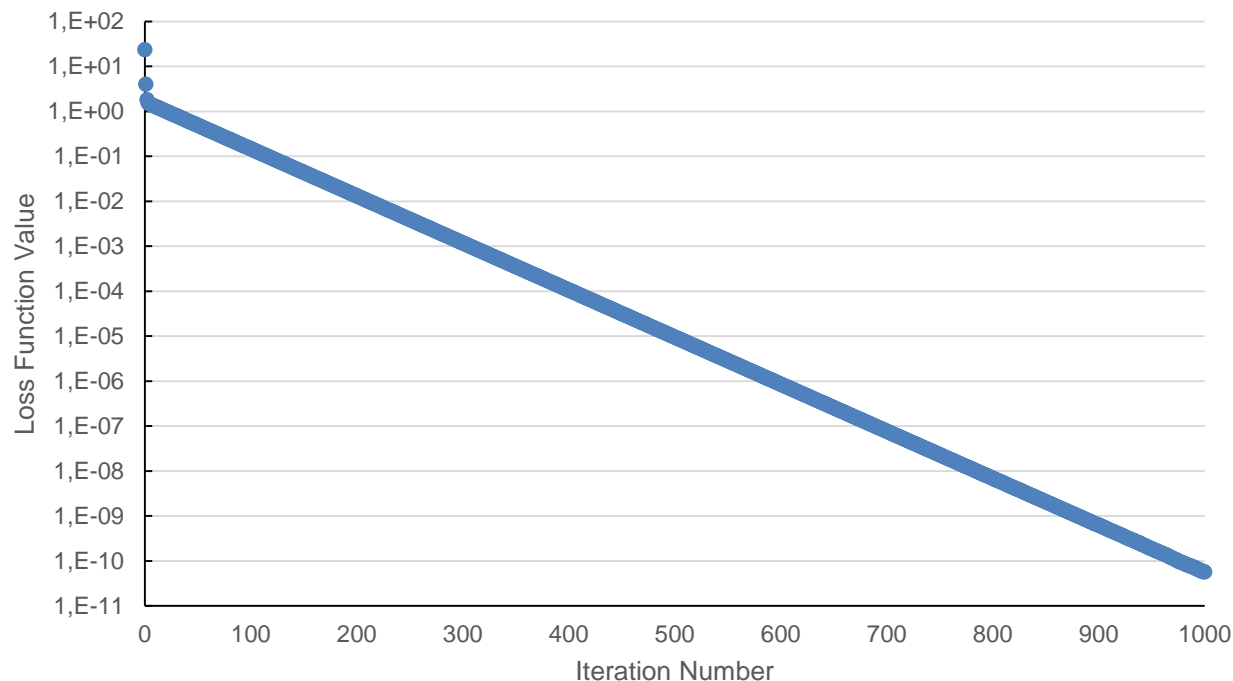
(bsc28069) mn1.bsc.es



# First TensorFlow example: Optimization Problem

## Exercise 1:

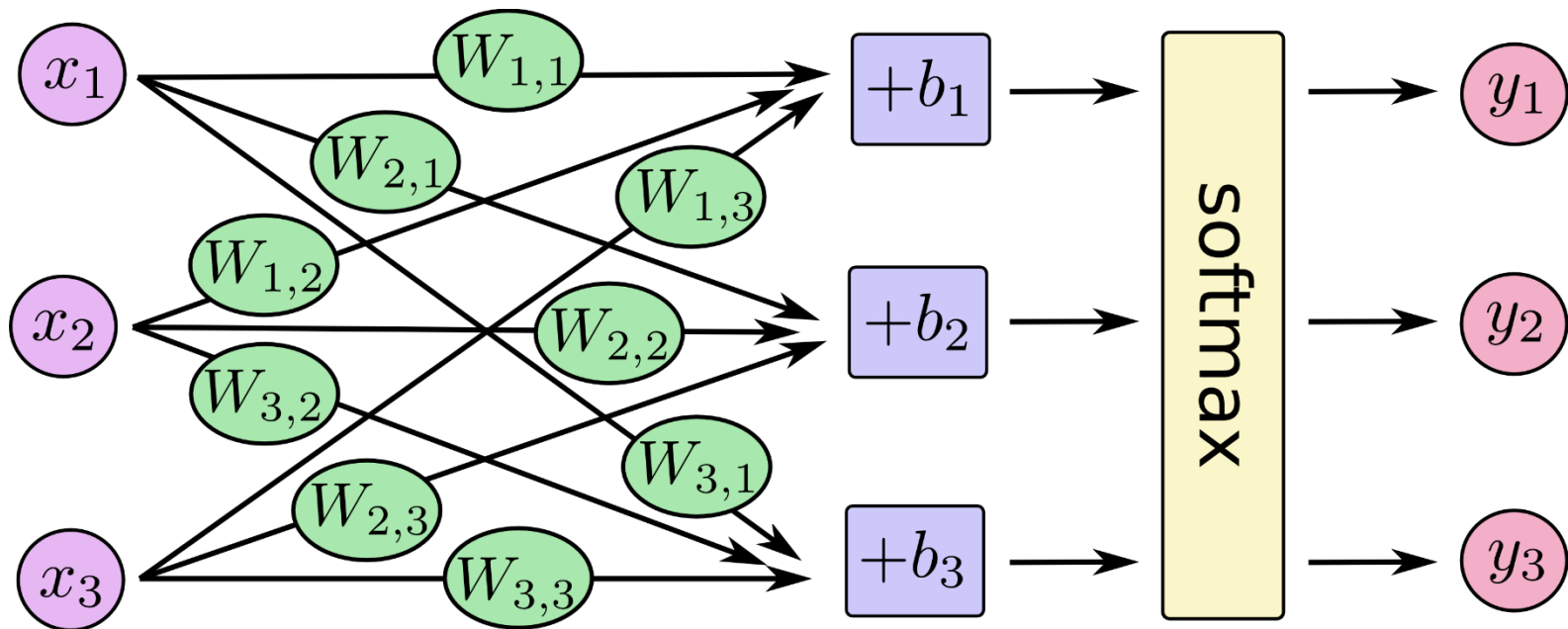
- Try GradientDescentOptimizer with different learning rates
- Check out other descent methods (look for options online)
- Represent these extra experiments plus the Gradient Descent with 0.01 learning rate





# Second TensorFlow Example: Single Layer Network

- « Data Set: MNIST, which is composed of 28x28 pixel images representing a number between 0 and 9. These 28x28 pixel images can be represented by 784 floating point values
- « In this example, we implement a very simple single-layer experiment



# Second TensorFlow Example: Single Layer Network

- « Data Set: MNIST, which is composed of 28x28 pixel images representing a number between 0 and 9. These 28x28 pixel images can be represented by 784 floating point values
- « In this example, we implement a very simple single-layer experiment

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

# Second TensorFlow Example: Single Layer Network

- « Data Set: MNIST, which is composed of 28x28 pixel images representing a number between 0 and 9. These 28x28 pixel images can be represented by 784 floating point values
- « In this example, we implement a very simple single-layer experiment

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

- « The provided implementation obtains a 90.79% accuracy

# Second TensorFlow Example: Single Layer Network

```
(bsc28069) mn1.bsc.es — Konsole
File Edit View Bookmarks Settings Help
#!/usr/bin/env python
import tensorflow as tf
import read_inputs
import numpy as N

#read data from file
data_input = read_inputs.load_data_mnist('MNIST_data/mnist.pkl.gz')
#FYI data = [(train_set_x, train_set_y), (valid_set_x, valid_set_y), (test_set_x, test_set_y)]
data = data_input[0]
#print ( N.shape(data[0][0])[0] )
#print ( N.shape(data[0][1])[0] )

#data layout changes since output should an array of 10 with probabilities
real_output = N.zeros( (N.shape(data[0][1])[0] , 10), dtype=N.float )
for i in range ( N.shape(data[0][1])[0] ):
    real_output[i][data[0][1][i]] = 1.0

#data layout changes since output should an array of 10 with probabilities
real_check = N.zeros( (N.shape(data[2][1])[0] , 10), dtype=N.float )
for i in range ( N.shape(data[2][1])[0] ):
    real_check[i][data[2][1][i]] = 1.0

#set up the computation. Definition of the variables.
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

#TRAINING PHASE
print("TRAINING")

for i in range(500):
    batch_xs = data[0][0][100*i:100*i+100]
    batch_ys = real_output[100*i:100*i+100]
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

#CHECKING THE ERROR
print("ERROR CHECK")

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: data[2][0], y_: real_check}))

"singlelayer.py" 54L, 1748C
Top
(bsc28069) mn1.bsc.es
```

# Second TensorFlow Example: Single Layer Network

- « Data Set: MNIST, which is composed of 28x28 pixel images representing a number between 0 and 9. These 28x28 pixel images can be represented by 784 floating point values
- « In this example, we implement a very simple single-layer experiment

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

## « Exercise 2:

- Plot convergence rates in a similar way as the previous exercise
- Consider different optimizers and learning rates

# Third TensorFlow Example: Multiple Layer Network

« Data Set: MNIST

« We consider a deep neural network with:

- First convolutional layer: 32 features per each 5x5 patch
- Second convolutional layer: 64 features for each 5x5 patch
- Densely connected layer: Processes 64 7x7 images with 1024 neurons
- Dropout rate: 50%

« Current version achieves 95.97% accuracy

« **Exercise 3:** Increase accuracy rate as much as possible.

- HINT: It is possible to reach accuracies above 99% by just re-arranging the way batches are defined.

« **Exercise 4:** All 3 examples use a single GPU.

- Reduce example 3 training time by using all the 4 GPU devices on either a node of the P9 cluster or some other multi-GPU system.
- Provide training time using 1, 2 and 4 GPUs. Discuss implementation.

# Third TensorFlow Example: Multiple Layer Network

```
(bsc28069) mn1.bsc.es — Konsole
File Edit View Bookmarks Settings Help

#!/usr/bin/env python
import tensorflow as tf
import read_inputs
import numpy as N

#read data from file
data_input = read_inputs.load_data_mnist('MNIST_data/mnist.pkl.gz')
#FYI data = [(train_set_x, train_set_y), (valid_set_x, valid_set_y), (test_set_x, test_set_y)]
data = data_input[0]
#print ( N.shape(data[0][0])[0] )
#print ( N.shape(data[0][1])[0] )

#data layout changes since output should an array of 10 with probabilities
real_output = N.zeros( (N.shape(data[0][1])[0] , 10), dtype=N.float )
for i in range ( N.shape(data[0][1])[0] ):
    real_output[i][data[0][1][i]] = 1.0

#data layout changes since output should an array of 10 with probabilities
real_check = N.zeros( (N.shape(data[2][1])[0] , 10), dtype=N.float )
for i in range ( N.shape(data[2][1])[0] ):
    real_check[i][data[2][1][i]] = 1.0

#set up the computation. Definition of the variables.
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
#b = tf.Variable(tf.zeros([10]))
#y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder(tf.float32, [None, 10])

#declare weights and biases
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

#convolution and pooling
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
        strides=[1, 2, 2, 1], padding='SAME')

#First convolutional layer: 32 features per each 5x5 patch
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

#Reshape x to a 4d tensor, with the second and third dimensions corresponding to image width and
#height.
#28x28 = 784
#The final dimension corresponding to the number of color channels.
x_image = tf.reshape(x, [-1, 28, 28, 1])
```



# Third TensorFlow Example: Multiple Layer Network

```
(bsc28069) mn1.bsc.es — Konsole
File Edit View Bookmarks Settings Help
x_image = tf.reshape(x, [-1, 28, 28, 1])

#We convolve x_image with the weight tensor, add the bias, apply the ReLU function, and finally
#max pool.
#The max_pool_2x2 method will reduce the image size to 14x14.
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

#Second convolutional layer: 64 features for each 5x5 patch.
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

#Densely connected layer: Processes the 64 7x7 images with 1024 neurons
#Reshape the tensor from the pooling layer into a batch of vectors,
#multiply by a weight matrix, add a bias, and apply a ReLU.
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

#drop_out
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

#Readout Layer
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

#Crossentropy
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))

train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    #TRAIN
    print("TRAINING")

    for i in range(1000):

        #until 1000 96,35%
        batch_ini = 50*i
        batch_end = 50*i+50
```

76,1 75%

(bsc28069) mn1.bsc.es

# Third TensorFlow Example: Multiple Layer Network

```
(bsc28069) mn1.bsc.es — Konsole
File Edit View Bookmarks Settings Help

W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

#drop_out
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

#Readout Layer
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

#Crossentropy
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))

train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    #TRAIN
    print("TRAINING")
    for i in range(1000):
        #until 1000 96.35%
        batch_ini = 50*i
        batch_end = 50*i+50

        batch_xs = data[0][0][batch_ini:batch_end]
        batch_ys = real_output[batch_ini:batch_end]

        if i % 10 == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch_xs, y_: batch_ys, keep_prob: 1.0})
            print('step %d, training accuracy %g Batch [%d,%d]' % (i, train_accuracy, batch_ini, batch_end))

        train_step.run(feed_dict={x: batch_xs, y_: batch_ys, keep_prob: 0.5})

    #TEST
    print("TESTING")

    train_accuracy = accuracy.eval(feed_dict={x: data[2][0], y_: real_check, keep_prob: 1.0})
    print('test accuracy %g' % (train_accuracy))

127,0-1 Bot
```

# Third TensorFlow Example: Multiple Layer Network

- « The lab is due on 12/01/23.
- « This practical exercise is **individual**.

# Research Opportunities

- « Are you interested in working in topics involving AI, HPC and computer architecture?
  - Contact [marc.casas@bsc.es](mailto:marc.casas@bsc.es)
- « We have ongoing research projects with top-level IT multinational companies, as well as collaborations with US universities.
- « Possibilities for
  - Master Projects
  - PhD
  - Others...



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*