Introduction to

TensorFlow

What is tensorflow (tf)?

“ TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well”.

When should I care about tf?

- I need to speed up my math computations and I want to use GPU.
  - Use cuda if you are brave enough
- I am doing some optimization problem
  - Implement gradient computation and optimizers otherwise.
- I want to do big scale machine learning
  - For small scale use scikit-learn, R...

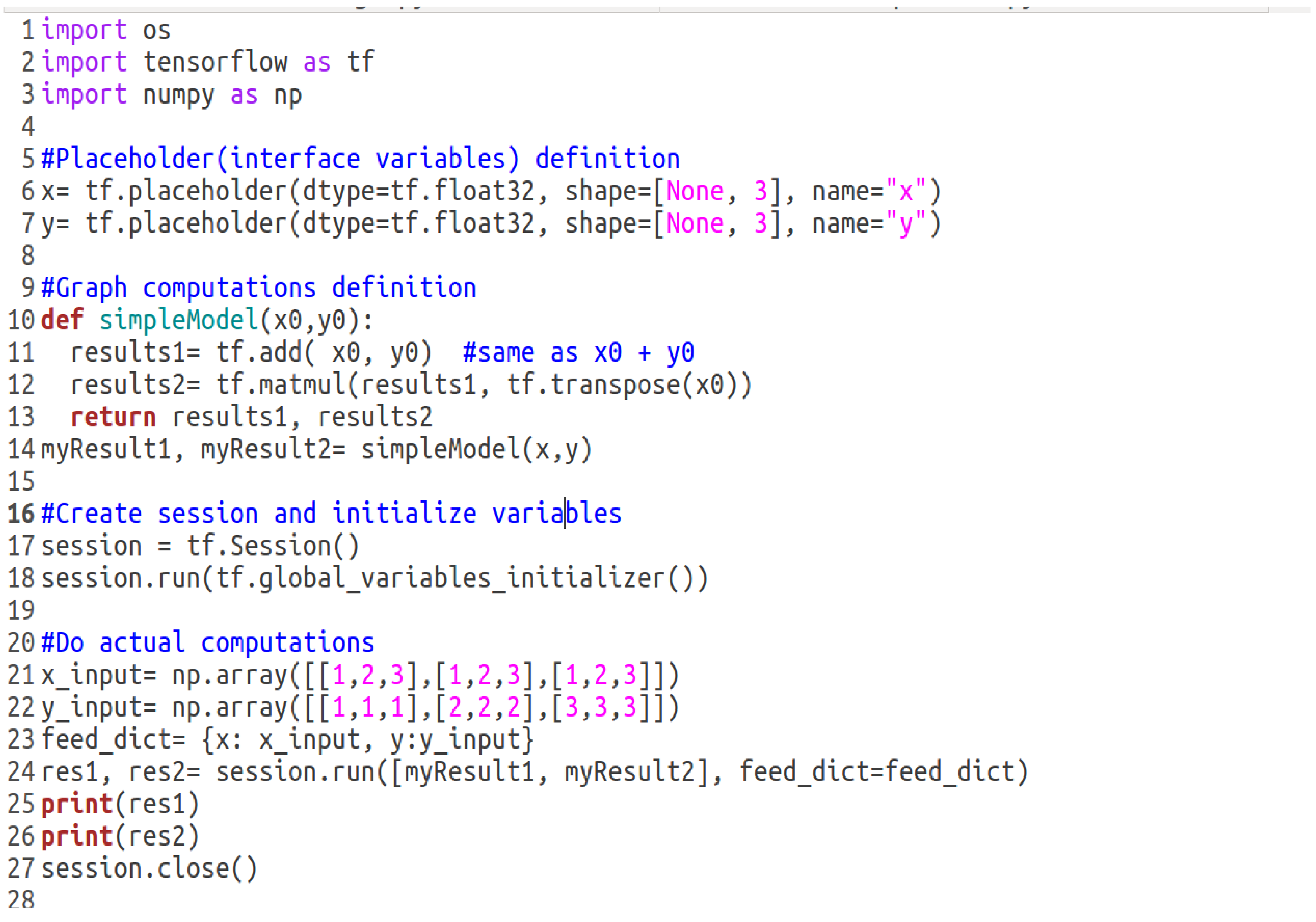
Why to choose tf?

- Multi-GPU support
- Cluster/cloud systems supported
- High performance, comparable to other popular frameworks (Torch, Theano)
- “Easy” to deploy for production
- Powerful visualization tool (tensorboard)
- Python and C++ API
- Supported by Google
- Tons of learning resources (Udacity course, playground, tutorials…)
- Easy installation (conda install, pip install...)

How does tf work?

1) You define desired operations (computation graph)
2) You create a session and use it to inject data and to grab results
3) Tf will compile your desired operations as soon as you execute your program
4) Tf will manage device communication and operations in a very efficient manner

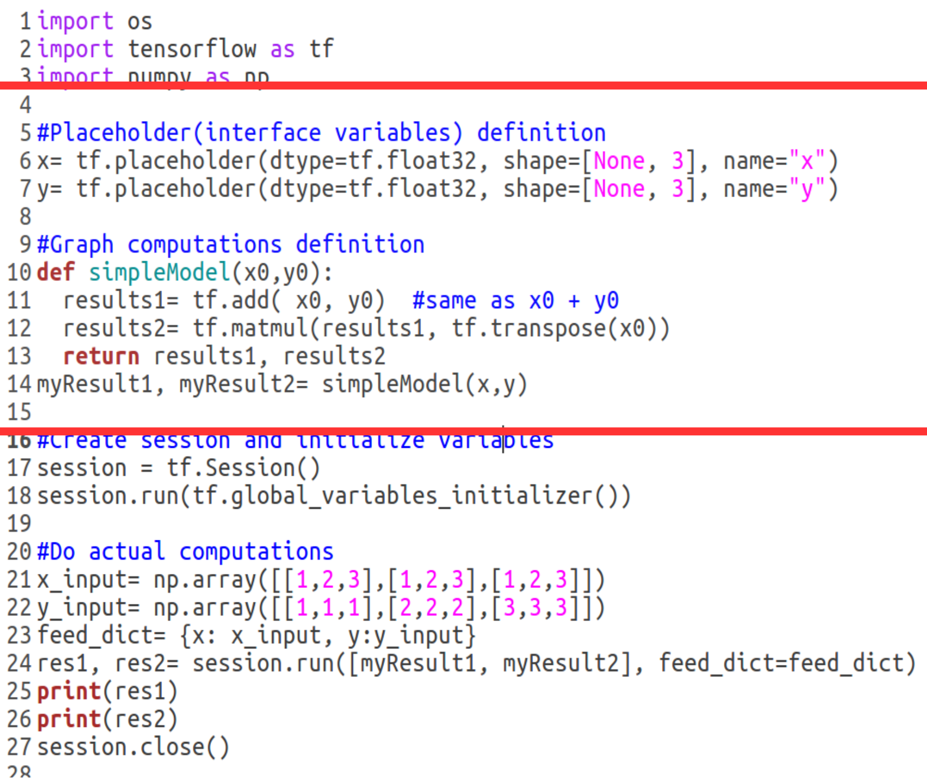
tf program structure



```python
import os
import tensorflow as tf
import numpy as np

#Placeholder(interface variables) definition
x= tf.placeholder(dtype=tf.float32, shape=[None, 3], name="x")
y= tf.placeholder(dtype=tf.float32, shape=[None, 3], name="y")

#Graph computations definition
def simpleModel(x0,y0):
  results1= tf.add( x0, y0)  #same as x0 + y0
  results2= tf.matmul(results1, tf.transpose(x0))
  return results1, results2
myResult1, myResult2= simpleModel(x,y)

#Create session and initialize variables
session = tf.Session()
session.run(tf.global_variables_initializer())

#Do actual computations
x_input= np.array([[1,2,3],[1,2,3],[1,2,3]])
y_input= np.array([[1,1,1],[2,2,2],[3,3,3]])
feed_dict= {x: x_input, y:y_input}
res1, res2= session.run([myResult1, myResult2], feed_dict=feed_dict)
print(res1)
print(res2)
session.close()

```

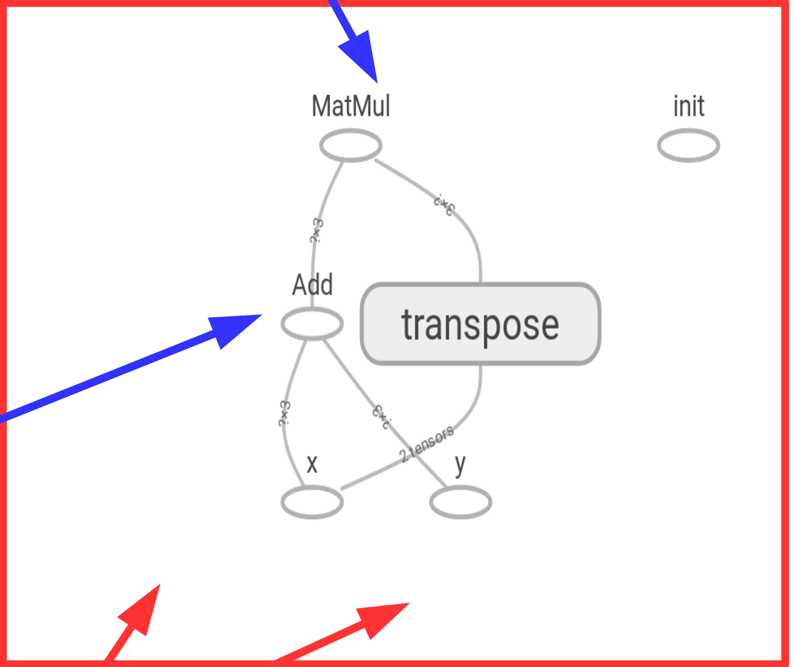
./1_tf_intro/examples/e1_smallProg.py

Computation graph definition



```
import os
import tensorflow as tf
import numpy as np

#Placeholder(interface variables) definition
x= tf.placeholder(dtype=tf.float32, shape=[None, 3], name="x")
y= tf.placeholder(dtype=tf.float32, shape=[None, 3], name="y")

#Graph computations definition
def simpleModel(x0,y0):
  results1= tf.add( x0, y0)  #same as x0 + y0
  results2= tf.matmul(results1, tf.transpose(x0))
  return results1, results2
myResult1, myResult2= simpleModel(x,y)

#Create session and initialize variables
session = tf.Session()
session.run(tf.global_variables_initializer())

#Do actual computations
x_input= np.array([[1,2,3],[1,2,3],[1,2,3]])
y_input= np.array([[1,1,1],[2,2,2],[3,3,3]])
feed_dict= {x: x_input, y:y_input}
res1, res2= session.run([myResult1, myResult2], feed_dict=feed_dict)
print(res1)
print(res2)
session.close()
```

myResult2



myResult1

Data will be injected here

Computation graph definition (2)

- Placeholders (for data injection)

  tf.placeholder(...)

- Constants

  tf.constant(…), tf.zeros(…), tf.ones(…) ...

- Variables (model parameters, several initialization schemes)

  tf.get_variable(…)

- Operations (can be applied to Variables, Constants and Placeholders)

  tf.add(...), tf.matmul(…), tf.exp(…), tf.diag_part(…) ...

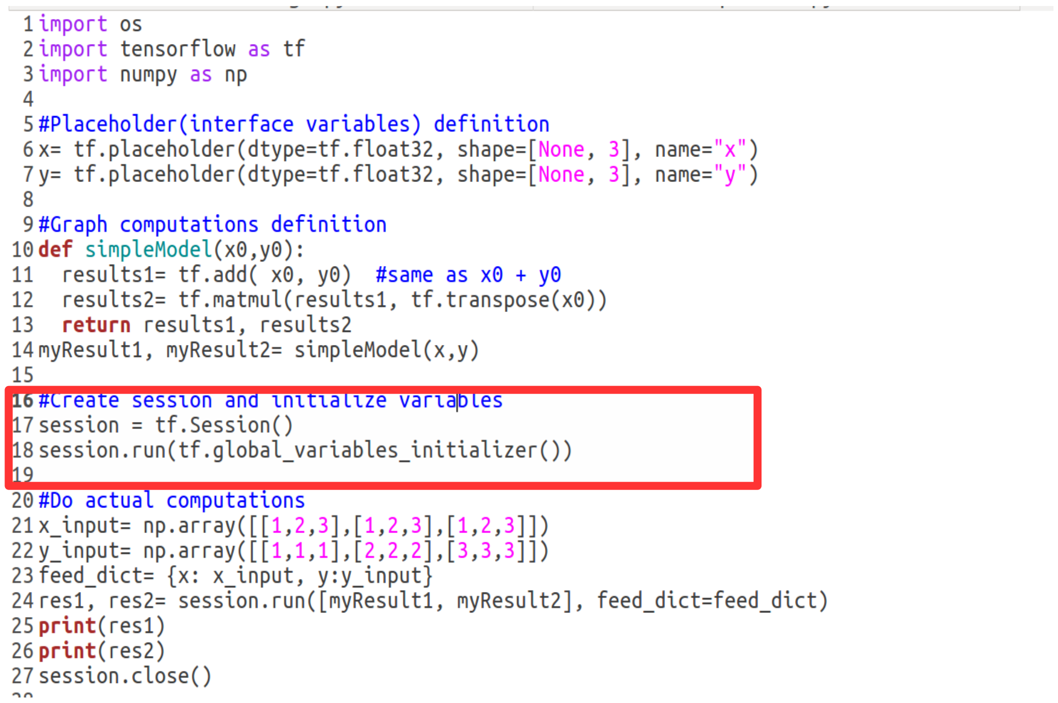
https://www.tensorflow.org/api_docs/

Session creation and variable initialization

By default, tf.Session() will make use of all available resources.

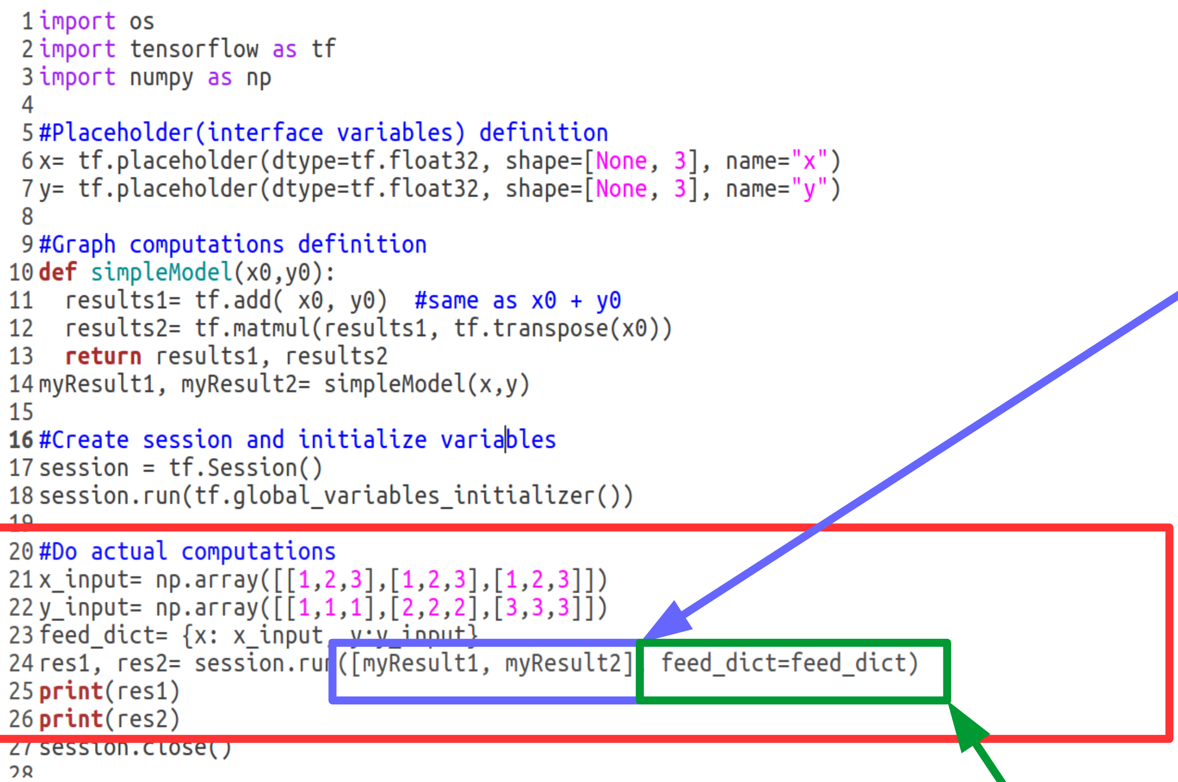
- If several GPUs, all be allocated (although not necessarily used)
  - Select GPU using environmental variable CUDA_VISIBLE_DEVICE
- If no GPU all CPUs will be used (use tf.ConfigProto as argument to tf.Session() to set desired number)

Variables must be either initialized or loaded from checkpoints before being used



```python
import os
import tensorflow as tf
import numpy as np

#Placeholder(interface variables) definition
x= tf.placeholder(dtype=tf.float32, shape=[None, 3], name="x")
y= tf.placeholder(dtype=tf.float32, shape=[None, 3], name="y")

#Graph computations definition
def simpleModel(x0,y0):
  results1= tf.add( x0, y0)  #same as x0 + y0
  results2= tf.matmul(results1, tf.transpose(x0))
  return results1, results2
myResult1, myResult2= simpleModel(x,y)

#Create session and initialize variables
session = tf.Session()
session.run(tf.global_variables_initializer())

#Do actual computations
x_input= np.array([[1,2,3],[1,2,3],[1,2,3]])
y_input= np.array([[1,1,1],[2,2,2],[3,3,3]])
feed_dict= {x: x_input, y:y_input}
res1, res2= session.run([myResult1, myResult2], feed_dict=feed_dict)
print(res1)
print(res2)
session.close()
```

Actual computation



```
import os
import tensorflow as tf
import numpy as np

#Placeholder(interface variables) definition
x= tf.placeholder(dtype=tf.float32, shape=[None, 3], name="x")
y= tf.placeholder(dtype=tf.float32, shape=[None, 3], name="y")

#Graph computations definition
def simpleModel(x0,y0):
  results1= tf.add( x0, y0)  #same as x0 + y0
  results2= tf.matmul(results1, tf.transpose(x0))
  return results1, results2
myResult1, myResult2= simpleModel(x,y)

#Create session and initialize variables
session = tf.Session()
session.run(tf.global_variables_initializer())

#Do actual computations
x_input= np.array([[1,2,3],[1,2,3],[1,2,3]])
y_input= np.array([[1,1,1],[2,2,2],[3,3,3]])
feed_dict= {x: x_input, y:y_input}
res1, res2= session.run([myResult1, myResult2], feed_dict=feed_dict)
print(res1)
print(res2)
session.close()
```

Graph variables we want to obtain.
- Tf will do just needed computations to reach these variables.
- MyResults1 and myResults2 state will be save as numpy narrays res1 and res2

WARNING: session.run is not recommended for real Deep Learning situations. See tf.data for recommended data pipelines.

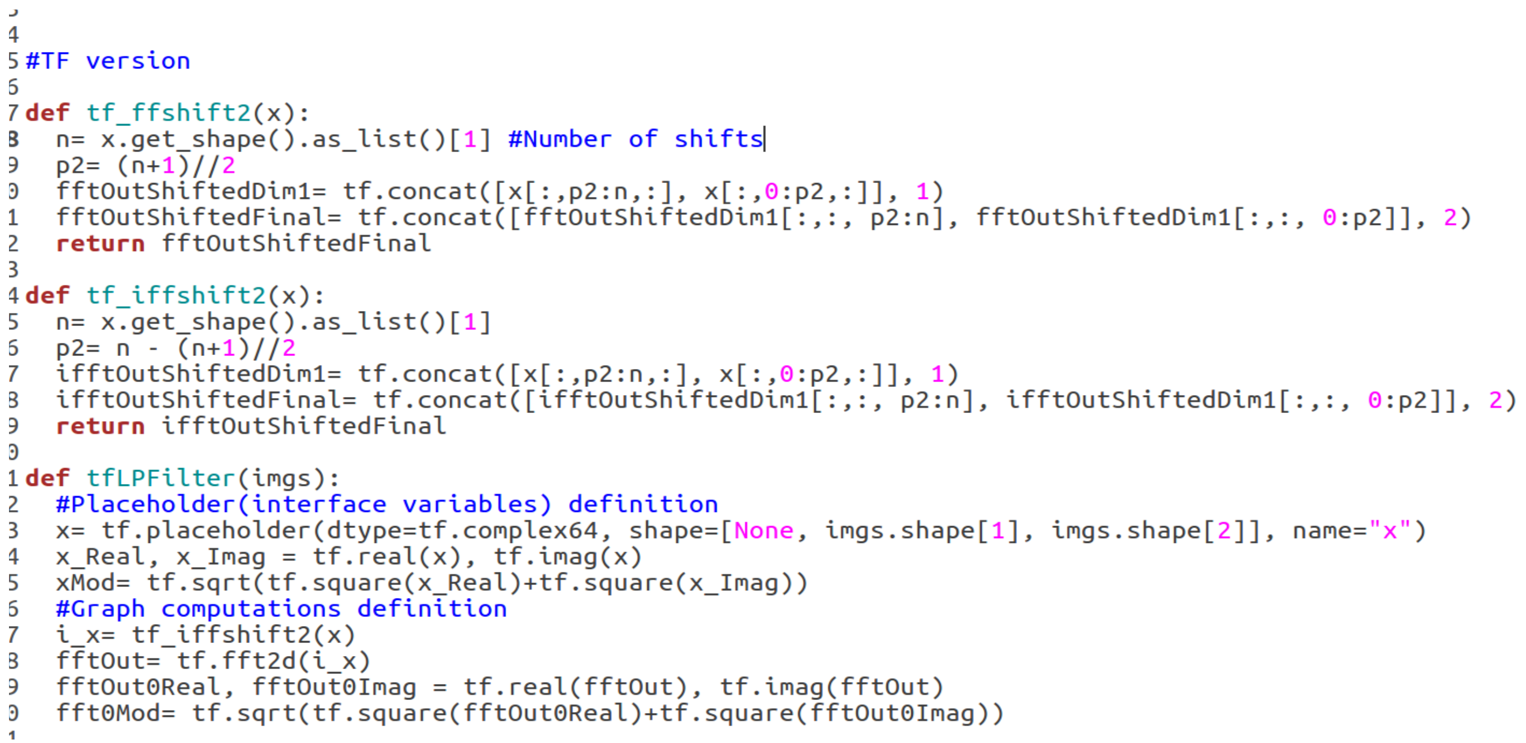
Dictionary
- Keys: graph placeholders
- Values: np.array's to inject into placeholders

One more interesting example

FFT usage example (numpy vs tf).

./1_tf_intro/examples/e2_fftExample.py

Matplotlib needed for results visualization:
conda install matplotlib



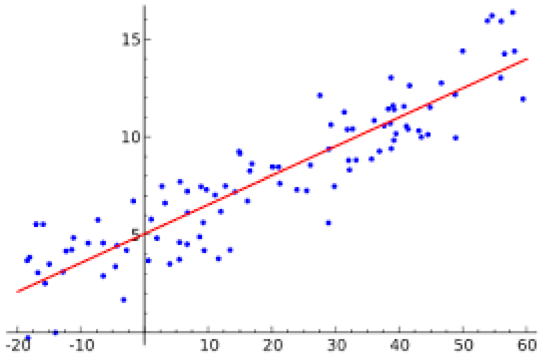
```python

#TF version

def tf_ffshift2(x):
  n= x.get_shape().as_list()[1] #Number of shifts
  p2= (n+1)//2
  fftOutShiftedDim1= tf.concat([x[:,p2:n,:], x[:,0:p2,:]], 1)
  fftOutShiftedFinal= tf.concat([fftOutShiftedDim1[:,:, p2:n], fftOutShiftedDim1[:,:, 0:p2]], 2)
  return fftOutShiftedFinal

def tf_iffshift2(x):
  n= x.get_shape().as_list()[1]
  p2= n - (n+1)//2
  ifftOutShiftedDim1= tf.concat([x[:,p2:n,:], x[:,0:p2,:]], 1)
  ifftOutShiftedFinal= tf.concat([ifftOutShiftedDim1[:,:, p2:n], ifftOutShiftedDim1[:,:, 0:p2]], 2)
  return ifftOutShiftedFinal

def tfLPFilter(imgs):
  #Placeholder(interface variables) definition
  x= tf.placeholder(dtype=tf.complex64, shape=[None, imgs.shape[1], imgs.shape[2]], name="x")
  x_Real, x_Imag = tf.real(x), tf.imag(x)
  xMod= tf.sqrt(tf.square(x_Real)+tf.square(x_Imag))
  #Graph computations definition
  i_x= tf_iffshift2(x)
  fftOut= tf.fft2d(i_x)
  fftOut0Real, fftOut0Imag = tf.real(fftOut), tf.imag(fftOut)
  fft0Mod= tf.sqrt(tf.square(fftOut0Real)+tf.square(fftOut0Imag))
```

Machine Learning: linear regression

$$y_{pred}=b+x_1 w_1+x_2 w_2+\ldots+x_n w_n=(1,x_1,x_2,\ldots,x_n)(b,w_1,w_2,\ldots,w_n)^T$$

Or alternatively

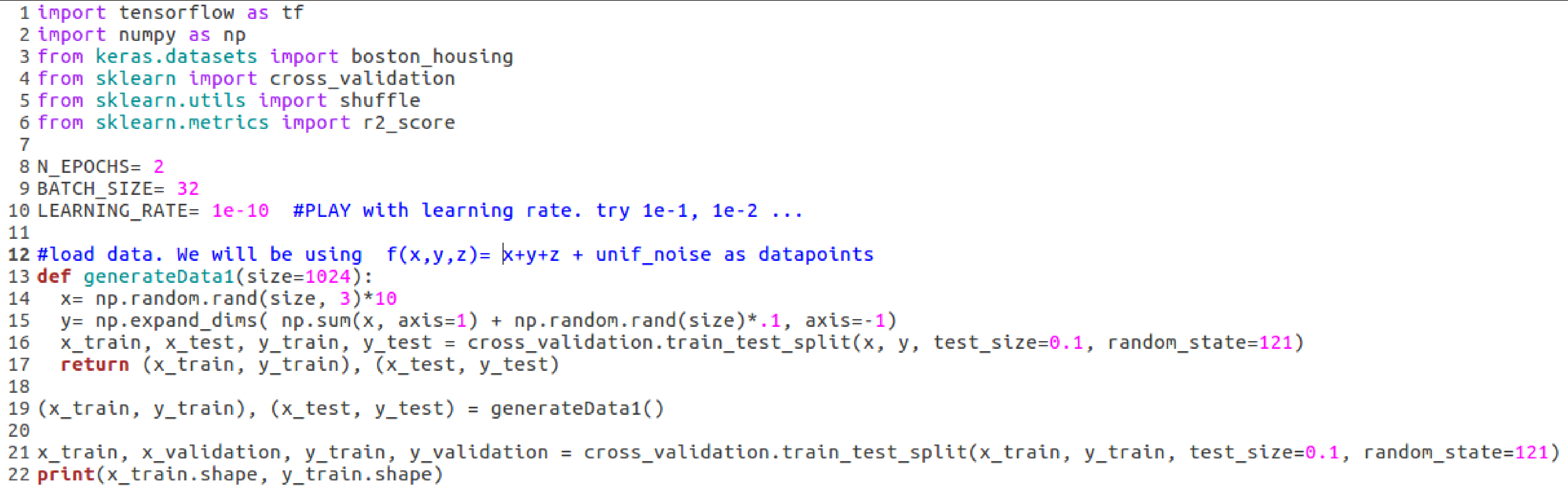
$$y=XW+b$$



./1_tf_intro/examples/e3_linRegr.py

Machine Learning: linear regression

Data preparation: f(x,y,z)= x+y+z +uniform_noise

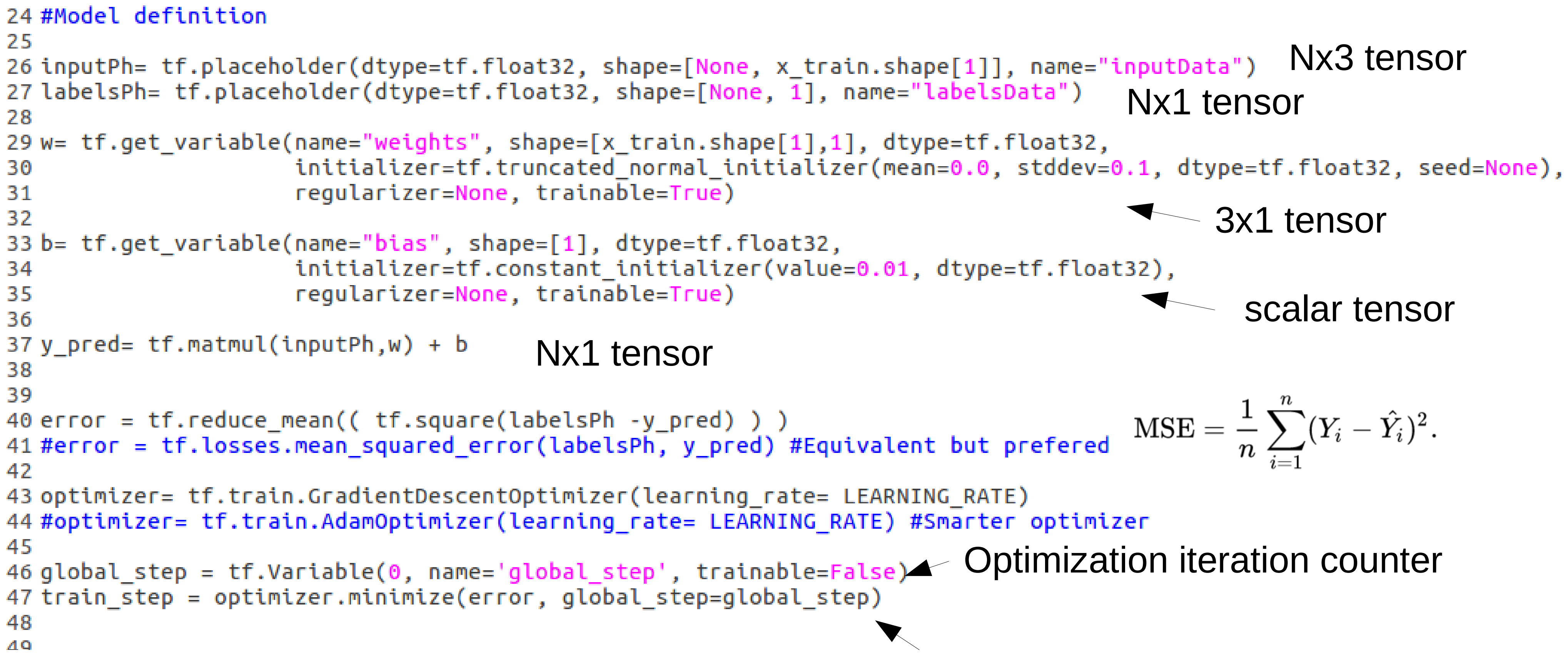


```
import tensorflow as tf
import numpy as np
from keras.datasets import boston_housing
from sklearn import cross_validation
from sklearn.utils import shuffle
from sklearn.metrics import r2_score

N_EPOCHS= 2
BATCH_SIZE= 32
LEARNING_RATE= 1e-10  #PLAY with learning rate. try 1e-1, 1e-2 ...

#load data. We will be using  f(x,y,z)= x+y+z + unif_noise as datapoints
def generateData1(size=1024):
  x= np.random.rand(size, 3)*10
  y= np.expand_dims( np.sum(x, axis=1) + np.random.rand(size)*.1, axis=-1)
  x_train, x_test, y_train, y_test = cross_validation.train_test_split(x, y, test_size=0.1, random_state=121)
  return (x_train, y_train), (x_test, y_test)

(x_train, y_train), (x_test, y_test) = generateData1()

x_train, x_validation, y_train, y_validation = cross_validation.train_test_split(x_train, y_train, test_size=0.1, random_state=121)
print(x_train.shape, y_train.shape)
```

./1_tf_intro/examples/e3_linRegr.py

Machine Learning: linear regression

$$y_{pred} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 = (1, x_1, x_2, x_3)(b, w_1, w_2, w_3)^T$$



```
#Model definition

inputPh= tf.placeholder(dtype=tf.float32, shape=[None, x_train.shape[1]], name="inputData")
labelsPh= tf.placeholder(dtype=tf.float32, shape=[None, 1], name="labelsData")

w= tf.get_variable(name="weights", shape=[x_train.shape[1],1], dtype=tf.float32,
                   initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=tf.float32, seed=None),
                   regularizer=None, trainable=True)

b= tf.get_variable(name="bias", shape=[1], dtype=tf.float32,
                   initializer=tf.constant_initializer(value=0.01, dtype=tf.float32),
                   regularizer=None, trainable=True)

y_pred= tf.matmul(inputPh,w) + b


error = tf.reduce_mean(( tf.square(labelsPh -y_pred) ) )
#error = tf.losses.mean_squared_error(labelsPh, y_pred) #Equivalent but prefered

optimizer= tf.train.GradientDescentOptimizer(learning_rate= LEARNING_RATE)
#optimizer= tf.train.AdamOptimizer(learning_rate= LEARNING_RATE) #Smarter optimizer

global_step = tf.Variable(0, name='global_step', trainable=False)
train_step = optimizer.minimize(error, global_step=global_step)
```

Nx3 tensor

Nx1 tensor

3x1 tensor

scalar tensor

Nx1 tensor

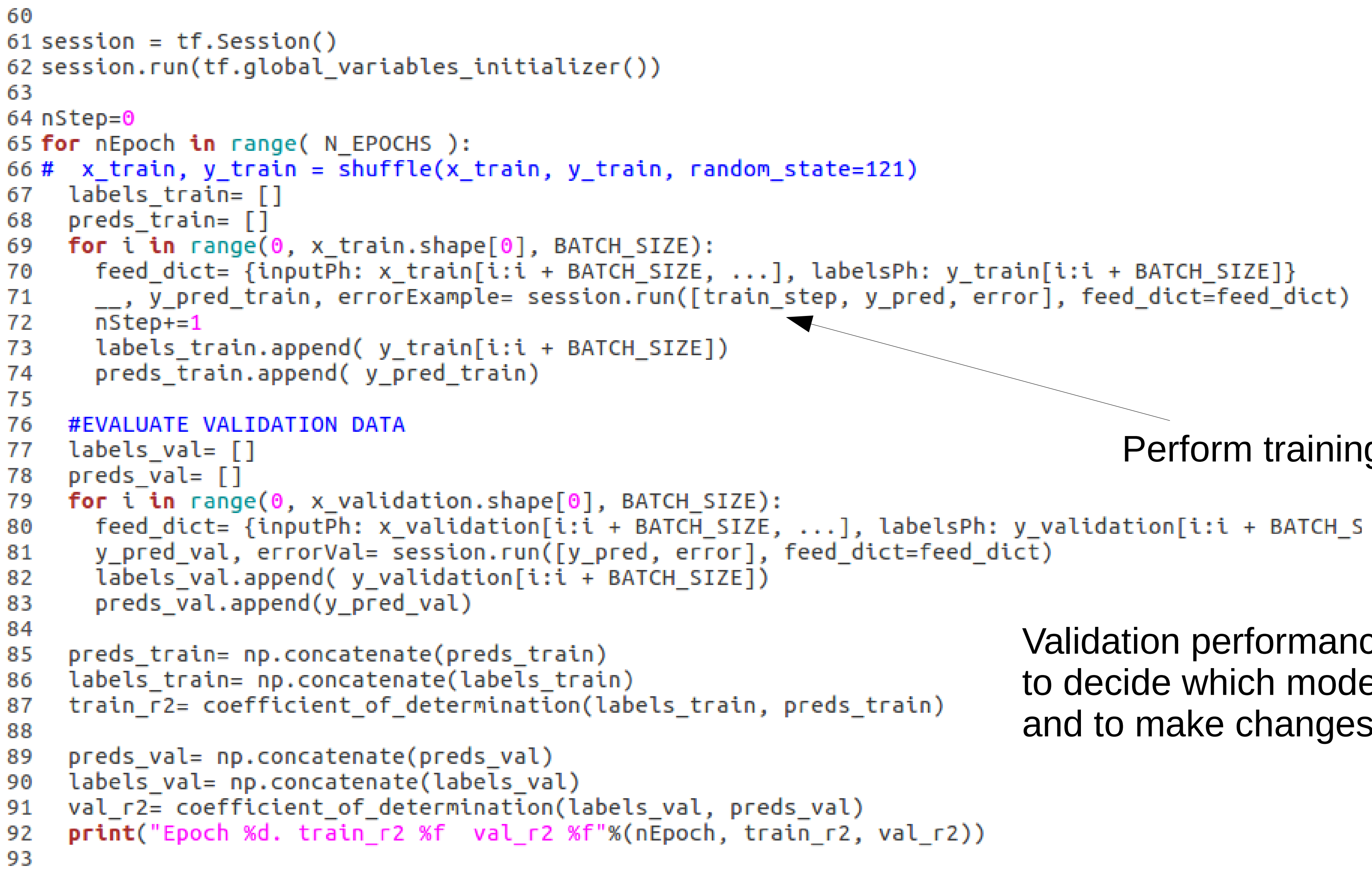
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Optimization iteration counter

Optimization step operation. Ask tensorflow to compute it in order to update parameters

./1_tf_intro/examples/e3_linRegr.py

Machine Learning: linear regression



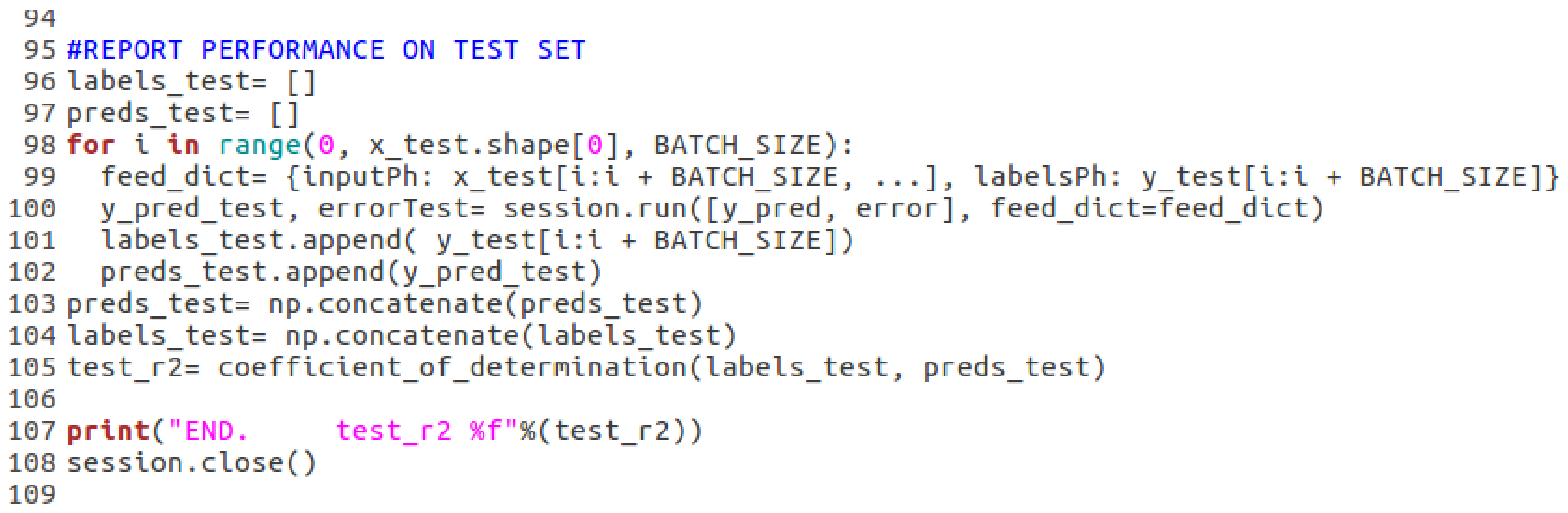
```python

session = tf.Session()
session.run(tf.global_variables_initializer())

nStep=0
for nEpoch in range( N_EPOCHS ):
#  x_train, y_train = shuffle(x_train, y_train, random_state=121)
  labels_train= []
  preds_train= []
  for i in range(0, x_train.shape[0], BATCH_SIZE):
    feed_dict= {inputPh: x_train[i:i + BATCH_SIZE, ...], labelsPh: y_train[i:i + BATCH_SIZE]}
    __, y_pred_train, errorExample= session.run([train_step, y_pred, error], feed_dict=feed_dict)
    nStep+=1
    labels_train.append( y_train[i:i + BATCH_SIZE])
    preds_train.append( y_pred_train)

  #EVALUATE VALIDATION DATA
  labels_val= []
  preds_val= []
  for i in range(0, x_validation.shape[0], BATCH_SIZE):
    feed_dict= {inputPh: x_validation[i:i + BATCH_SIZE, ...], labelsPh: y_validation[i:i + BATCH_S
    y_pred_val, errorVal= session.run([y_pred, error], feed_dict=feed_dict)
    labels_val.append( y_validation[i:i + BATCH_SIZE])
    preds_val.append(y_pred_val)

  preds_train= np.concatenate(preds_train)
  labels_train= np.concatenate(labels_train)
  train_r2= coefficient_of_determination(labels_train, preds_train)

  preds_val= np.concatenate(preds_val)
  labels_val= np.concatenate(labels_val)
  val_r2= coefficient_of_determination(labels_val, preds_val)
  print("Epoch %d. train_r2 %f  val_r2 %f"%(nEpoch, train_r2, val_r2))

```

Perform training step

Validation performance is used to decide which model is best and to make changes

./1_tf_intro/examples/e3_linRegr.py

Machine Learning: linear regression

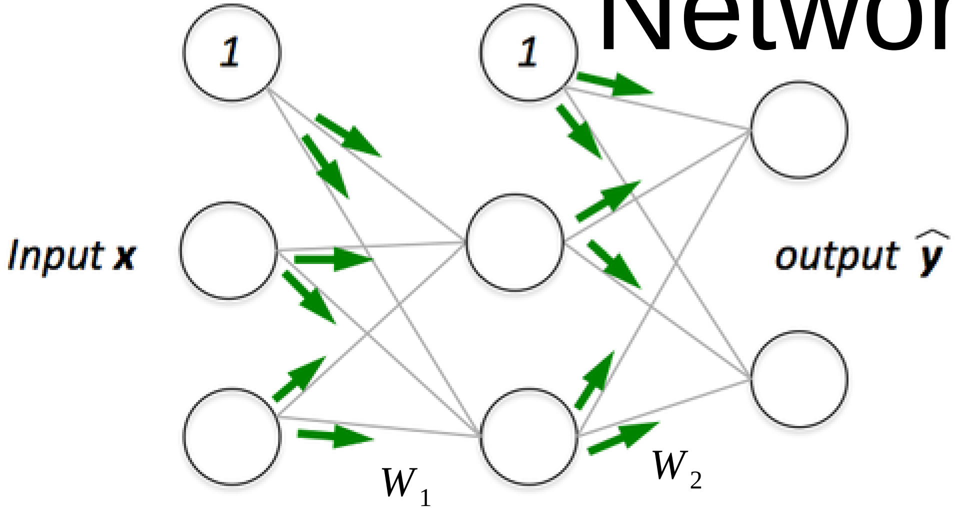


```
#REPORT PERFORMANCE ON TEST SET
labels_test= []
preds_test= []
for i in range(0, x_test.shape[0], BATCH_SIZE):
  feed_dict= {inputPh: x_test[i:i + BATCH_SIZE, ...], labelsPh: y_test[i:i + BATCH_SIZE]}
  y_pred_test, errorTest= session.run([y_pred, error], feed_dict=feed_dict)
  labels_test.append( y_test[i:i + BATCH_SIZE])
  preds_test.append(y_pred_test)
preds_test= np.concatenate(preds_test)
labels_test= np.concatenate(labels_test)
test_r2= coefficient_of_determination(labels_test, preds_test)

print("END.     test_r2 %f"%(test_r2))
session.close()

```

Testing set should be used just once, when reporting performance. No decision must be taken using testing set performance

./1_tf_intro/examples/e3_linRegr.py

Your turn: implement Neural Network for regression

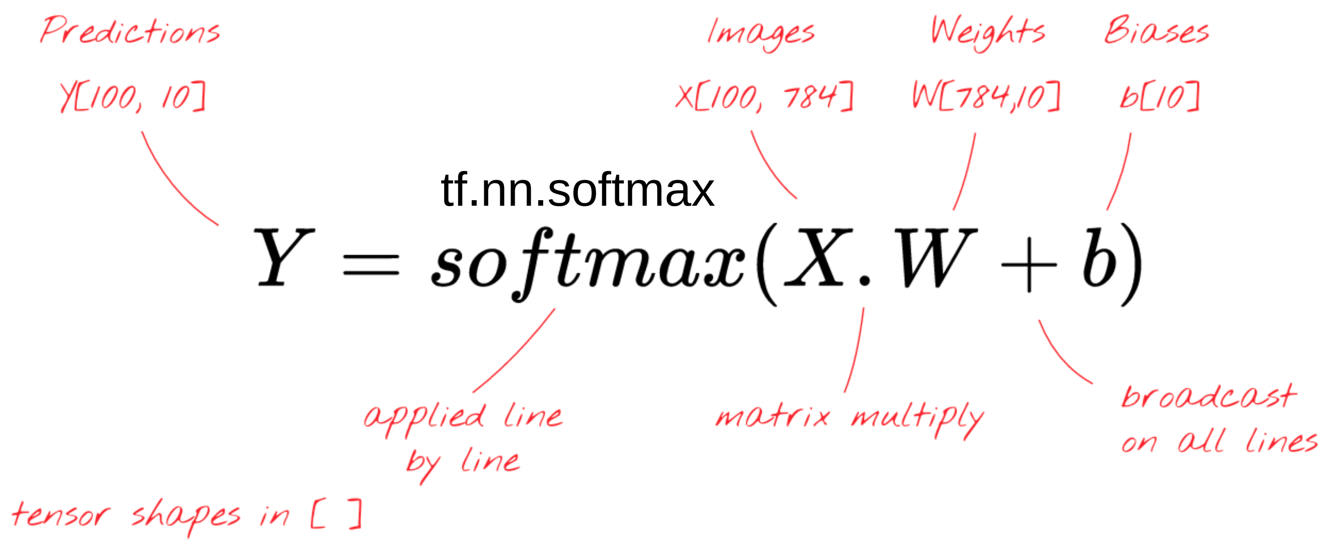


$$y_{pred} = W_2 g(X W_1) \rightarrow g \; can \; be \; sigmoid, \tanh, relu \ldots$$

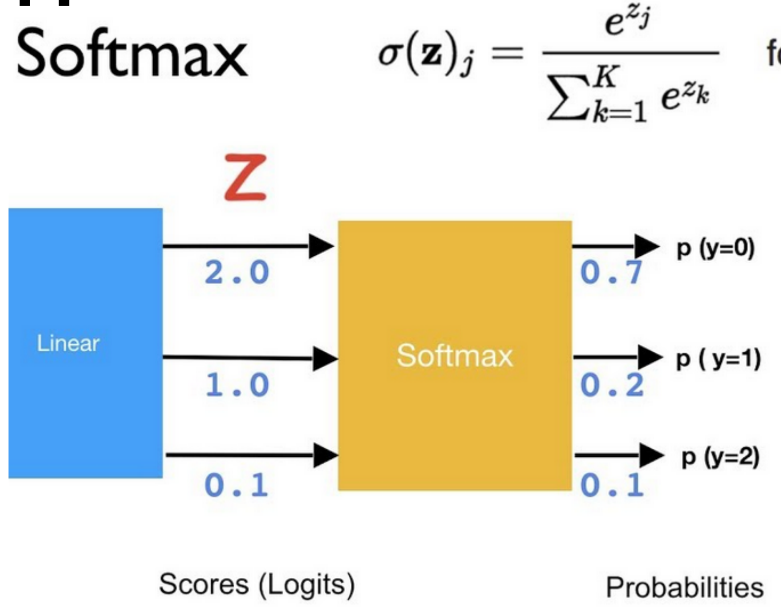
- If doubts API: https://www.tensorflow.org/api_docs/python/
- Try different number of layers, different number of neurons, different learning rate… See if your model is capable to overfit.
- If synthetic data is too easy, try to use realistic data
  - Add these 2 lines:
    - from keras.datasets import boston_housing
    - (x_train, y_train), (x_test, y_test) = boston_housing.load_data()
  - Data scaling can be an issue

My solution: ./1_tf_intro/exercises_sol/sol1_nNetReg.py

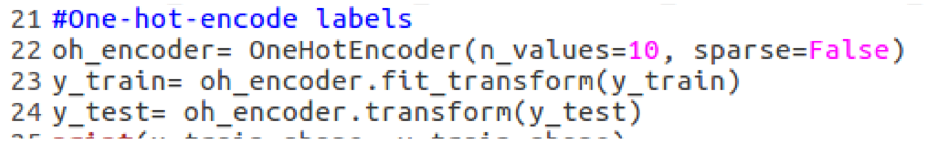
Your turn: implement logistic model for classification



$$Y = softmax(X.W + b)$$



- New data set: cifar10 → 50K images of 10 categories: dog, cat, car, boat, …
  - from keras.datasets import cifar10
  - (x_train, y_train), (x_test, y_test) = cifar10.load_data()
- Labels must be one-hot-encoded: do it automatically with:
  - sklearn.preprocessing.OneHotEncoder



```
#One-hot-encode labels
oh_encoder= OneHotEncoder(n_values=10, sparse=False)
y_train= oh_encoder.fit_transform(y_train)
y_test= oh_encoder.transform(y_test)
```

- Data scaling can be an issue

My solution: ./1_tf_intro/exercises_sol/sol2_logRegr.py

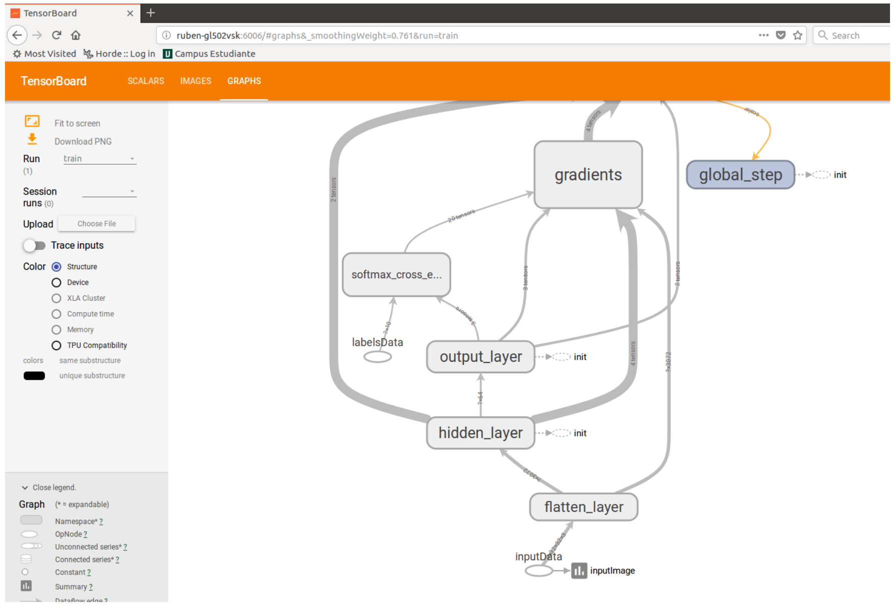
Your turn: implement Neural Network for classification

$$y_{pred} = softmax(W_2 \, g(X W_1)) \rightarrow g \; can \, be \; sigmoid, \tanh, relu \ldots$$

- You can use logistic regression script as starting point
- Try other architectures
  - Explore convolutional layers in API if you have finished
- Help yourself using tensorboard

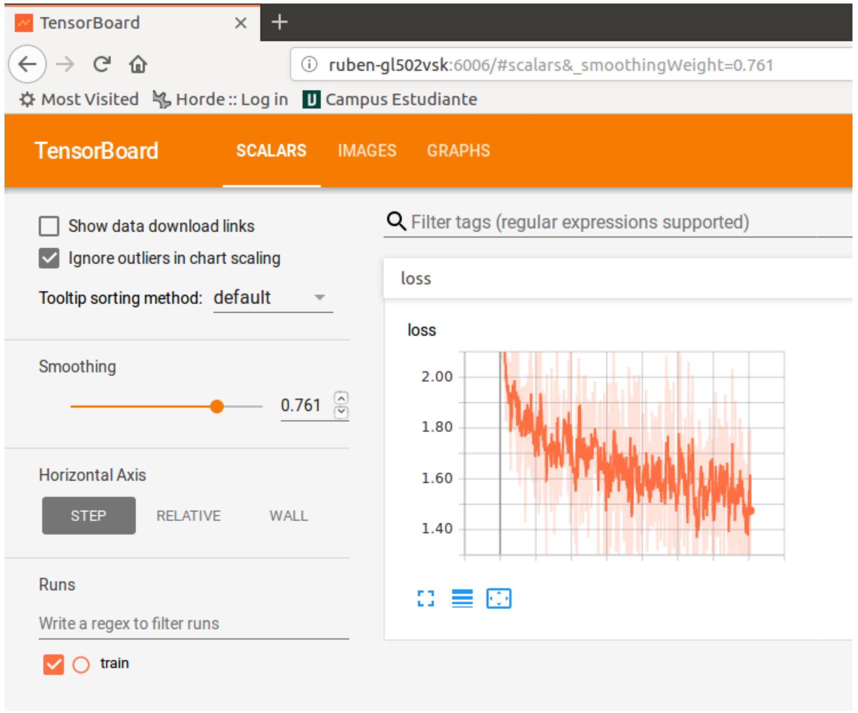
My solution: ./1_tf_intro/exercises_sol/sol3_nNetClas.py

tensorboard



- Choose what you want to monitor in your graph
  - Scalar tensor:
    - tf.summary.scalar(name, scalar_tensor)
  - Images:
    - tf.summary.image(name, image_tensor)
  - Histograms:
    - tf.summary.histogram(name, any_tensor)
- Right after variables initialization
    - merged_summary= tf.summary.merge_all()
    - train_writer = tf.summary.FileWriter(dir/for/logs/train, session.graph)
    - validation_writer = tf.summary.FileWriter(dir/for/logs/test, session.graph)
- When training, ask for merged_summary and save it:
    - __, m_summary= session.run([train_step, merged_summary], feed_dict=feed_dict)
    - train_writer.add_summary(m_summary, nStep)
- Do the same when computing validation
- Launch tensorboard from command line: tensorboard --logdir dir/for/logs/test
- Visualize it using your web browser and the url reported by tensorboard

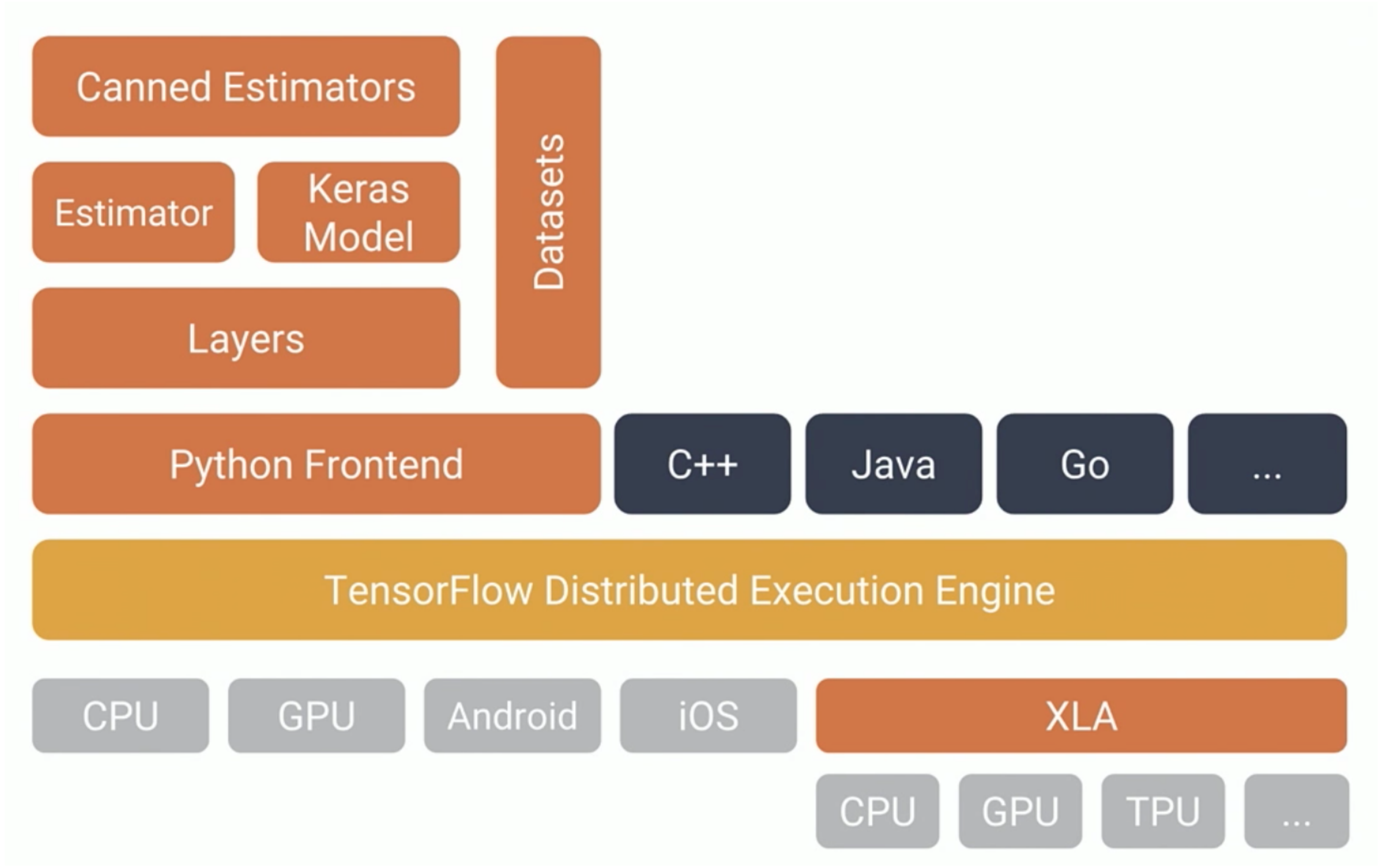
tensorboard


TensorBoard
ruben-gl502vsk:6006/#scalars&_smoothingWeight=0.761
Most Visited
Horde :: Log in
Campus Estudiante
TensorBoard
SCALARS
IMAGES
GRAPHS
Filter tags (regular expressions supported)
Runs
Write a regex to filter runs
loss
loss
2.00
1.80
1.60
1.40

☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.761
Horizontal Axis: [✓ STEP] RELATIVE WALL
☑ train

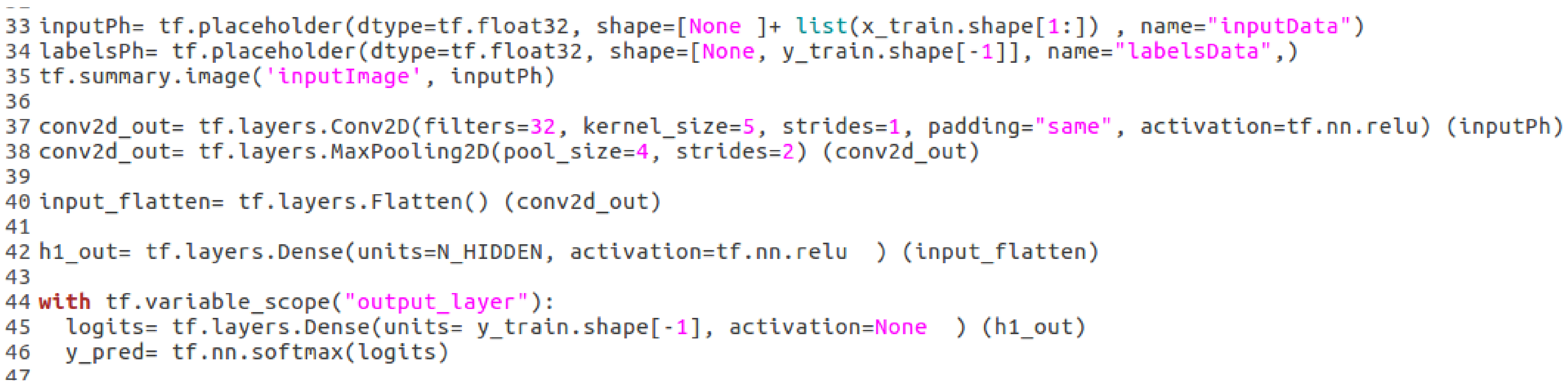
- Choose what you want to monitor in your graph
  - Scalar tensor:
    - tf.summary.scalar(name, scalar_tensor)
  - Images:
    - tf.summary.image(name, image_tensor)
  - Histograms:
    - tf.summary.histogram(name, any_tensor)
- Right after variables initialization
    - merged_summary= tf.summary.merge_all()
    - train_writer = tf.summary.FileWriter(dir/for/logs/train, session.graph)
    - validation_writer = tf.summary.FileWriter(dir/for/logs/test, session.graph)
- When training, ask for merged_summary and save it:
    - __, m_summary= session.run([train_step, merged_summary], feed_dict=feed_dict)
    - train_writer.add_summary(m_summary, nStep)
- Do the same when computing validation
- Launch tensorboard from command line: tensorboard --logdir dir/for/logs/test
- Visualize it using your web browser and the url reported by tensorboard

High level tf



Layers level tf

- No need to define weights and bias variables
- No need to care about tensor shapes except for the placeholders



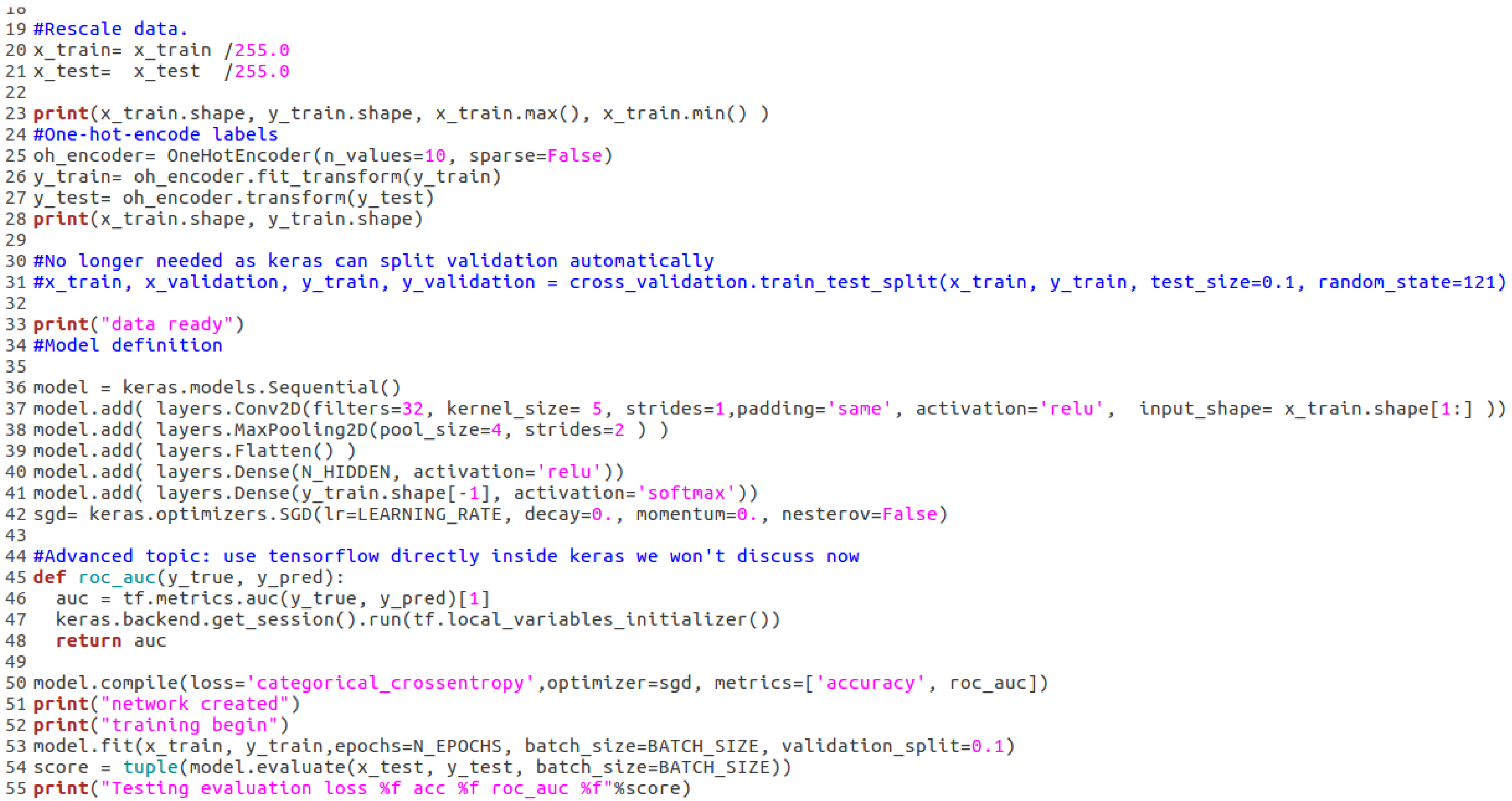
```
inputPh= tf.placeholder(dtype=tf.float32, shape=[None ]+ list(x_train.shape[1:]) , name="inputData")
labelsPh= tf.placeholder(dtype=tf.float32, shape=[None, y_train.shape[-1]], name="labelsData",)
tf.summary.image('inputImage', inputPh)

conv2d_out= tf.layers.Conv2D(filters=32, kernel_size=5, strides=1, padding="same", activation=tf.nn.relu) (inputPh)
conv2d_out= tf.layers.MaxPooling2D(pool_size=4, strides=2) (conv2d_out)

input_flatten= tf.layers.Flatten() (conv2d_out)

h1_out= tf.layers.Dense(units=N_HIDDEN, activation=tf.nn.relu  ) (input_flatten)

with tf.variable_scope("output_layer"):
  logits= tf.layers.Dense(units= y_train.shape[-1], activation=None  ) (h1_out)
  y_pred= tf.nn.softmax(logits)
```

- Training, evaluation, checkpointing, etc. it is still cumbersome

My solution: ./1_tf_intro/examples/e4_nNet_tfLayers.py

keras

- Easy model definition
- Easy model training, evaluation, checkpointing



```python
#Rescale data.
x_train= x_train /255.0
x_test=  x_test  /255.0

print(x_train.shape, y_train.shape, x_train.max(), x_train.min() )
#One-hot-encode labels
oh_encoder= OneHotEncoder(n_values=10, sparse=False)
y_train= oh_encoder.fit_transform(y_train)
y_test= oh_encoder.transform(y_test)
print(x_train.shape, y_train.shape)

#No longer needed as keras can split validation automatically
#x_train, x_validation, y_train, y_validation = cross_validation.train_test_split(x_train, y_train, test_size=0.1, random_state=121)

print("data ready")
#Model definition

model = keras.models.Sequential()
model.add( layers.Conv2D(filters=32, kernel_size= 5, strides=1,padding='same', activation='relu',  input_shape= x_train.shape[1:] ))
model.add( layers.MaxPooling2D(pool_size=4, strides=2 ) )
model.add( layers.Flatten() )
model.add( layers.Dense(N_HIDDEN, activation='relu'))
model.add( layers.Dense(y_train.shape[-1], activation='softmax'))
sgd= keras.optimizers.SGD(lr=LEARNING_RATE, decay=0., momentum=0., nesterov=False)

#Advanced topic: use tensorflow directly inside keras we won't discuss now
def roc_auc(y_true, y_pred):
  auc = tf.metrics.auc(y_true, y_pred)[1]
  keras.backend.get_session().run(tf.local_variables_initializer())
  return auc

model.compile(loss='categorical_crossentropy',optimizer=sgd, metrics=['accuracy', roc_auc])
print("network created")
print("training begin")
model.fit(x_train, y_train,epochs=N_EPOCHS, batch_size=BATCH_SIZE, validation_split=0.1)
score = tuple(model.evaluate(x_test, y_test, batch_size=BATCH_SIZE))
print("Testing evaluation loss %f acc %f roc_auc %f"%score)
```

My solution: ./1_tf_intro/examples/e5_nNet_keras.py

To be continued