TensorFlow

Tensorboard

- https://itnext.io/how-to-use-tensorboard-5d82f8654496
-

```
[17]  !pip install tensorboardcolab
```

```
Requirement already satisfied: tensorboardcolab in /usr/loca
```

```
[21]  from tensorboardcolab import *
      tbc = TensorBoardColab()
      writer = tbc.get_writer()
      writer.add_graph(tf.get_default_graph())
      writer.flush()
```

```
Wait for 8 seconds...
TensorBoard link:
https://f23e4892.ngrok.io
```

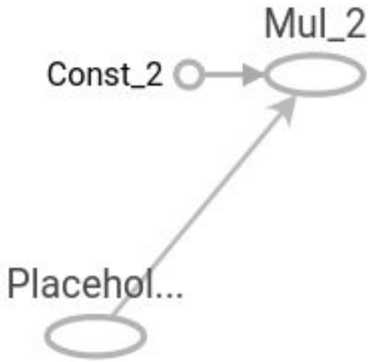
```
[12]  %load_ext tensorboard
```

```
[22]  %tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 1212), started 0:28:34 ago. (Use '!kill 1212' to kill it.)
```

```
[20] import numpy as np
sess = tf.Session()
x_vals = np.array([1., 3., 5., 7., 9.])
x_data = tf.placeholder(tf.float32)
m_const = tf.constant(3.)
my_product = tf.multiply(x_data, m_const)
for x_val in x_vals:
  print(sess.run(my_product, feed_dict={x_data: x_val}))
```

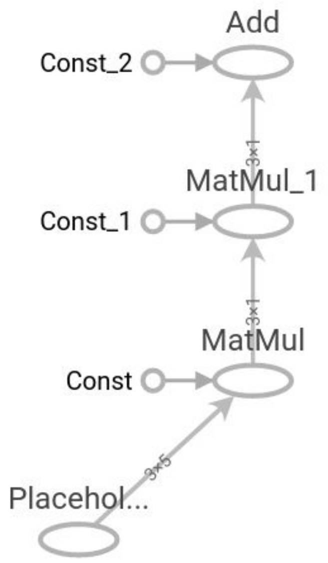
```
3.0
9.0
15.0
21.0
27.0
```



[10]

```python
import tensorflow as tf
tf.reset_default_graph()
graph1 = tf.Graph()

with graph1.as_default():
  my_array = np.array([[1., 3., 5., 7., 9.],
  [-2., 0., 2., 4., 6.],
  [-6., -3., 0., 3., 6.]])
  x_vals = np.array([my_array, my_array + 1])
  x_data = tf.placeholder(tf.float32, shape=(3, 5))

  m1 = tf.constant([[1.],[0.],[-1.],[2.],[4.]])
  m2 = tf.constant([[2.]])
  a1 = tf.constant([[10.]])

  prod1 = tf.matmul(x_data, m1)
  prod2 = tf.matmul(prod1, m2)
  add1 = tf.add(prod2, a1)
sess = tf.Session(graph=graph1)
for x_val in x_vals:
  print(sess.run(add1, feed_dict={x_data: x_val}))
```

```
[[102.]
 [ 66.]
 [ 58.]]
[[114.]
 [ 78.]
 [ 70.]]
```




```python
from tensorboardcolab import *
tbc = TensorBoardColab()
writer = tbc.get_writer()
#writer.add_graph(tf.get_default_graph())
writer.add_graph(graph1)
writer.flush()
```

```
Using TensorFlow backend.
Wait for 8 seconds...
TensorBoard link:
https://2885951c.ngrok.io
WARNING:tensorflow:From /usr/local/lib/python3.
```

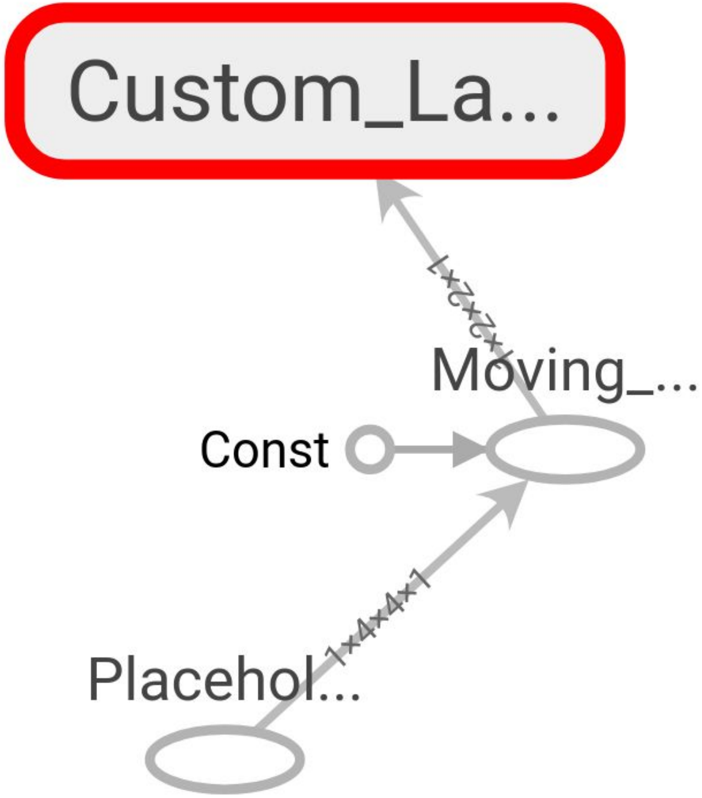
Creating a Custom Layer

[2]

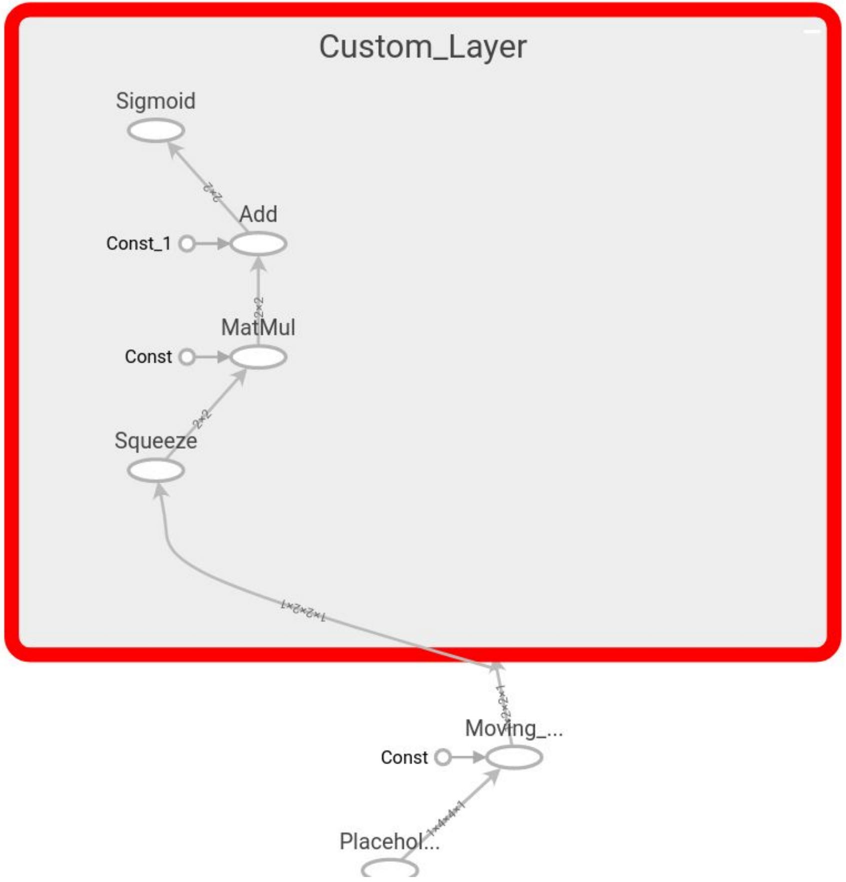
```python
%tensorflow_version 1.x
import tensorflow as tf
print(tf.__version__)
import numpy as np
tf.reset_default_graph()
sess = tf.Session()
# dimension: no of images, width, height, no. of channels
x_shape = [1, 4, 4, 1]
x_val = np.random.uniform(size=x_shape)
x_data = tf.placeholder(tf.float32, shape=x_shape)

# filter 2x2
my_filter = tf.constant(0.25, shape=[2, 2, 1, 1])
my_strides = [1, 2, 2, 1]
mov_avg_layer = tf.nn.conv2d(x_data, my_filter, my_strides, padding='SAME',\
                             name='Moving_Avg_Window')
# output image dimensions is 2x2

def custom_layer(input_matrix):
    input_matrix_squeezed = tf.squeeze(input_matrix)
    A = tf.constant([[1., 2.], [-1.,3.]])
    b = tf.constant(1.,shape=[2,2])
    temp1 = tf.matmul(A, input_matrix_squeezed)
    temp = tf.add(temp1, b)
    return (tf.sigmoid(temp))

with tf.name_scope('Custom_Layer') as scope:
    custom_layer1 = custom_layer(mov_avg_layer)
print(sess.run(custom_layer1, feed_dict={x_data: x_val}))
```

```
1.15.2
[[0.9231764  0.9021275 ]
 [0.8062733  0.89410377]]
```

- Take a single input image of size 4x4 pixels and one channel: [1,4,4,1]
- Convolve it using a 2x2 filter with a stride of length 2 and zero padding
- Output of convolution is a 1x2x2x1 image
- Custom Layer implements the following function: y = sigmoid(A * Input_image + b)



Custom_La...
1×2×2×1
Moving_...
Const
1×4×4×1
Placehol...



Custom_Layer
Sigmoid
2×2
Add
Const_1
2×2
MatMul
Const
2×2
Squeeze
1×2×2×1
1×2×2×1
Moving_...
Const
1×4×4×1
Placehol...

Back Propagation Training

- Create the data set

- Initialize placeholders and variables.

- Create a loss function. Sigmoid cross-entropy is used as the loss function.

- Define an optimization algorithm.

- And finally, iterate across random data samples to iteratively update our variables - Training
- Incremental / stochastic training - Weights are updated with each pair of input-output data.
- Batch training - Weights are updated only once for each batch of input-output data.

```python
%tensorflow_version 1.x
import numpy as np
import tensorflow as tf
sess = tf.Session()
x_vals = np.random.normal(1, 0.1, 100) # mean = 1, s.d = 0.1
y_vals = np.repeat(10., 100)
x_data = tf.placeholder(shape=[1], dtype=tf.float32)
y_target = tf.placeholder(shape=[1], dtype=tf.float32)

# Weight is randomly initialized, will be updated during training
A = tf.Variable(tf.random.normal(shape=[1,]))

my_output = tf.multiply(x_data, A)

# L2 loss
loss = tf.square(my_output - y_target)

# initialize all the variables
init = tf.initialize_all_variables()
sess.run(init)

# define an optimizer to minimize loss
my_opt = tf.train.GradientDescentOptimizer(learning_rate=0.02)
train_step = my_opt.minimize(loss)

# Train
for i in range(100):
    rand_index = np.random.choice(100)
    rand_x = [x_vals[rand_index]]
    rand_y = [y_vals[rand_index]]
    sess.run(train_step, feed_dict={x_data: rand_x, y_target:rand_y})
    if (i+1)% 25 == 0:
        print('Step #'+str(i+1)+'A =' + str(sess.run(A)))
        print('Loss=' + str(sess.run(loss, feed_dict={x_data: rand_x, \
                                                        y_target: rand_y })))
```

Implementing Back Propagation to optimize weights for a Regression Problem

- This is a regression example: Y = A * X
- A is the tunable parameter updated by the optimization routine during training.
- First define a loss function to minimize
- Then use the built-in gradient descent optimizer
- Then train using random input-output pairs

```
Step #25A =[6.47813]
Loss=[11.517864]
Step #50A =[8.76959]
Loss=[2.456357]
Step #75A =[9.557501]
Loss=[3.2807326e-05]
Step #100A =[9.903524]
Loss=[0.45638904]
```

Implementing Backpropagation for a Classification Problem

- The loss function expects batches of data
- Model is: Y = A + X
- Task is to learn A from input-output data
- Input: scalar normal random number x: N(-1,1) + N(3,1)
- Output: scalar (0/1)
- Sigmoid cross-entropy is used as a loss function.
- TF's `GradientDescent' is used to update weights

```
%tensorflow_version 1.x
import numpy as np
import tensorflow as tf
print(tf.__version__)
from tensorflow.python.framework import ops
ops.reset_default_graph()
sess = tf.Session()

# create dataset using two different normal distributions
x_vals = np.concatenate((np.random.normal(-1,1,50), np.random.normal(3,1,50)))
y_vals = np.concatenate((np.repeat(0., 50), np.repeat(1., 50)))
x_data = tf.placeholder(shape=[1], dtype=tf.float32)
y_target = tf.placeholder(shape=[1], dtype=tf.float32)

# Tunable parameter to be learnt
A = tf.Variable(tf.random_normal(mean=10, shape=[1]))
# Model output
my_output = tf.add(x_data, A)

# Create batches of input-output data for training
my_output_expanded = tf.expand_dims(my_output, 0)
y_target_expanded = tf.expand_dims(y_target, 0)

# Initialize
init = tf.initialize_all_variables()
sess.run(init)

# define the loss function
xentropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=my_output_expanded, \
                                                    labels=y_target_expanded)
# Add optimizer
my_opt = tf.train.GradientDescentOptimizer(0.05)
train_step = my_opt.minimize(xentropy)

# Train
for i in range(2000):
  rand_index = np.random.choice(100)
  rand_x = [x_vals[rand_index]]
  rand_y = [y_vals[rand_index]]

  sess.run(train_step, feed_dict={x_data: rand_x, y_target: rand_y})

  if (i+1)%200 == 0:
    print('Step #' + str(i+1) + ' A= ' + str(sess.run(A)))
    print('Loss = ' + str(sess.run(xentropy, feed_dict={\
        x_data: rand_x, y_target: rand_y })))
```

```
1.15.2
Step #200 A= [5.704602]
Loss = [[7.4643416]]
Step #400 A= [1.1865458]
Loss = [[0.03637219]]
Step #600 A= [-0.22917922]
Loss = [[0.1883726]]
Step #800 A= [-0.8221022]
Loss = [[0.10990714]]
Step #1000 A= [-0.8956753]
Loss = [[0.19115523]]
Step #1200 A= [-0.97926164]
Loss = [[0.04974011]]
Step #1400 A= [-0.9753762]
Loss = [[0.06269645]]
Step #1600 A= [-0.9933789]
Loss = [[0.05458826]]
Step #1800 A= [-0.9098405]
Loss = [[0.12116913]]
Step #2000 A= [-0.9930337]
Loss = [[0.09670994]]
```

```python
%tensorflow_version 1.x
import tensorflow as tf
print(tf.__version__)
import matplotlib.pyplot as plt
import numpy as np
tf.reset_default_graph()
sess = tf.Session()

# define a batch size
batch_size = 20

# define data, placeholders and variables
x_vals = np.random.normal(1, 0.1, 100)
y_vals = np.repeat(10., 100)
x_data = tf.placeholder(shape=[None, 1], dtype=tf.float32)
y_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)
A = tf.Variable(tf.random_normal(shape=[1,1]))

# model output
my_output = tf.matmul(x_data, A)

# Initialize
init = tf.initialize_all_variables()
sess.run(init)

# Define Loss function
loss = tf.reduce_mean(tf.square(my_output-y_target))

# Define Optimizer
my_opt = tf.train.GradientDescentOptimizer(0.02)
train_step = my_opt.minimize(loss)

# Batch training
loss_batch = []
for i in range(200):
  rand_index = np.random.choice(100, size=batch_size)
  rand_x = np.transpose([x_vals[rand_index]])
  rand_y = np.transpose([y_vals[rand_index]])
  sess.run(train_step, feed_dict={x_data: rand_x, y_target: rand_y})

  if (i+1)%20 == 0:
    print('Step #' + str(i+1) + ' A= ' + str(sess.run(A)))
    temp_loss = sess.run(loss, feed_dict={x_data: rand_x, y_target: rand_y})
    print('Loss = ' + str(temp_loss))
    loss_batch.append(temp_loss)
```

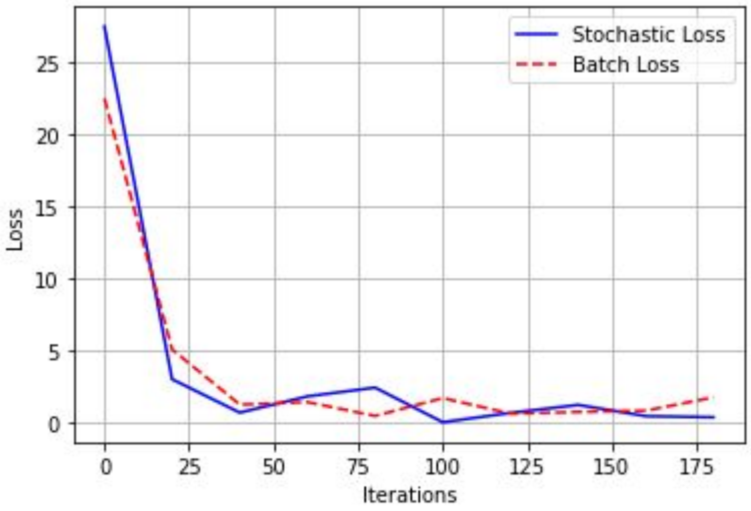
Batch training example:

- Loss function needs a batch of samples
- Batch_size should be selected as trade-off between computation time and convergence performance.

```
1.15.2
Step #20 A= [[4.5326767]]
Loss = 30.542887
Step #40 A= [[7.5594163]]
Loss = 7.492503
Step #60 A= [[8.870335]]
Loss = 1.9721674
Step #80 A= [[9.49131]]
Loss = 1.530283
Step #100 A= [[9.726826]]
Loss = 0.8185464
Step #120 A= [[9.841177]]
Loss = 0.93517625
Step #140 A= [[9.933927]]
Loss = 0.6304198
Step #160 A= [[9.969463]]
Loss = 0.93619955
Step #180 A= [[9.955895]]
Loss = 1.1733254
Step #200 A= [[9.981412]]
Loss = 0.9398285
```

Comparing stochastic and batch training

```
# Reinitialize network parameters
sess.run(init)

# stochastic training
loss_stochastic = []
for i in range(200):
    rand_index = np.random.choice(100)
    rand_x = [x_vals[rand_index]]
    rand_y = [y_vals[rand_index]]
    rand_x = np.reshape(rand_x, [1,1])
    rand_y = np.reshape(rand_y, [1,1])
    sess.run(train_step, feed_dict={x_data: rand_x, y_target: rand_y})
    if(i+1)%20 == 0:
        print('Step #' + str(i+1) + ' A= ' + str(sess.run(A)))
        temp_loss = sess.run(loss, feed_dict = {x_data: rand_x, y_target: rand_y})
        print('Loss = ' + str(temp_loss))
        loss_stochastic.append(temp_loss)

# Plot
plt.plot(range(0,200,20), loss_stochastic, 'b-', label='Stochastic Loss')
plt.plot(range(0,200,20), loss_batch, 'r--', label='Batch Loss')
plt.legend(loc='upper right')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.grid()
plt.show()
```



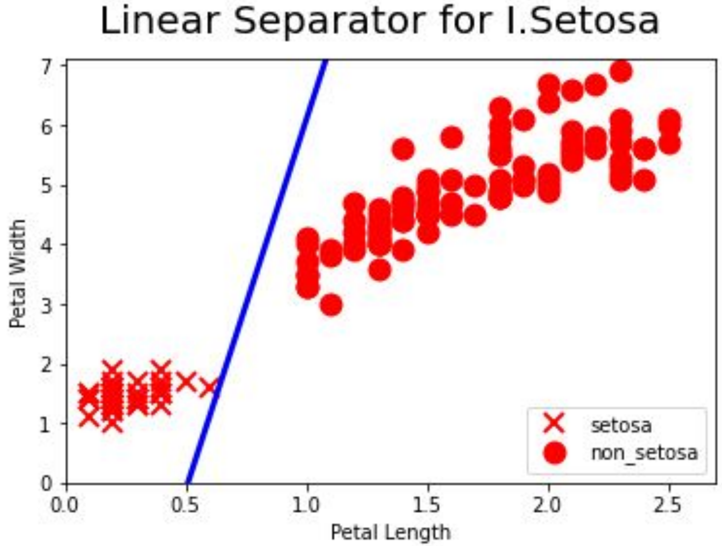
```python
%tensorflow_version 1.x
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
import tensorflow as tf
sess = tf.Session()

# Load IRIS dataset
iris = datasets.load_iris()
print('shape of iris dataset: ' + str(np.shape(iris.data)))
binary_target = np.array([1.0 if x == 0 else 0.0 for x in iris.target])
iris_2d = np.array([[x[2], x[3]] for x in iris.data])
print('shape of iris 2d dataset: ' + str(np.shape(iris_2d)))

# define batch_size, placeholders, variables
batch_size = 20
x1_data = tf.placeholder(shape=[None, 1], dtype=tf.float32)
x2_data = tf.placeholder(shape=[None, 1], dtype=tf.float32)
y_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)
A = tf.Variable(tf.random_normal(shape=[1,1]))
b = tf.Variable(tf.random_normal(shape=[1,1]))

# define a linear classifier
my_mult = tf.matmul(x2_data, A)
my_add = tf.add(my_mult, b)
my_output = tf.subtract(x1_data, my_add)

# define loss function
xentropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=my_output, \
                                                   labels=y_target)

# define optimizer
my_opt = tf.train.GradientDescentOptimizer(0.05)
train_step = my_opt.minimize(xentropy)

# Initialize
init = tf.initialize_all_variables()
sess.run(init)
```

Building a Binary Classifier for IRIS data

- Classify the IRIS data into two classes: setosa and non-setosa using two features: petal length and petal width
- Linear Classifier is used: x2 = Ax1 + b
- Output: y = x2 - (Ax1 + b)
- If y > 0: Class 0 else class 1

```python
# train
for i in range(1000):
  rand_index = np.random.choice(len(iris_2d), size=batch_size)
  rand_x = iris_2d[rand_index]
  rand_x1 = np.array([[x[0]] for x in rand_x])
  rand_x2 = np.array([[x[1]] for x in rand_x])
  rand_y = np.array([[y] for y in binary_target[rand_index]])
  sess.run(train_step, feed_dict={x1_data: rand_x1, x2_data: rand_x2, \
                                  y_target: rand_y})
  if (i+1) % 200 == 0:
    print('Step # ' + str(i+1) + ' A = ' + str(sess.run(A)) +\
          ' b = ' + str(sess.run(b)))


# plot the plane
[[slope]] = sess.run(A)
[[intercept]] = sess.run(b)
x = np.linspace(0, 3, num=50)# plot the plane
[[slope]] = sess.run(A)
[[intercept]] = sess.run(b)
x = np.linspace(0, 3, num=50)
ablineValues = []
for i in x:
  ablineValues.append(slope * i + intercept)
setosa_x = [a[1] for i,a in enumerate(iris_2d) if binary_target[i] == 1]
setosa_y = [a[0] for i,a in enumerate(iris_2d) if binary_target[i] == 1]

non_setosa_x = [a[1] for i,a in enumerate(iris_2d) if binary_target[i] == 0]
non_setosa_y = [a[0] for i,a in enumerate(iris_2d) if binary_target[i] == 0]

plt.plot(setosa_x, setosa_y, 'rx', ms=10, mew = 2, label='setosa')
plt.plot(non_setosa_x, non_setosa_y, 'ro', ms=10, mew = 2, label='non_setosa')
plt.plot(x, ablineValues,'b-',linewidth=3)
plt.xlim([0.0, 2.7])
plt.ylim([0.0,7.1])
plt.suptitle('Linear Separator for I.Setosa', fontsize=20)
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.legend(loc='lower right')
plt.show()
```

```
shape of iris dataset: (150, 4)
shape of iris 2d dataset: (150, 2)
Step # 200 A = [[8.718943]] b = [[-3.4034028]]
Step # 400 A = [[10.123796]] b = [[-4.7624884]]
Step # 600 A = [[11.165169]] b = [[-5.3947916]]
Step # 800 A = [[11.904662]] b = [[-5.8730826]]
Step # 1000 A = [[12.41249]] b = [[-6.3340874]]
```



Evaluating Models

```
%tensorflow_version 1.x
import tensorflow as tf
print(tf.__version__)
import matplotlib.pyplot as plt
import numpy as np
sess = tf.Session()
x_vals = np.random.normal(1, 0.1, 100)
y_vals = np.repeat(10., 100)
x_data = tf.placeholder(shape=[None, 1], dtype=tf.float32)
y_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)
batch_size = 25

train_indices = np.random.choice(len(x_vals), \
                                 round(len(x_vals)*0.8), replace=False)
test_indices = np.array(list(set(range(len(x_vals))) - set(train_indices)))

x_vals_train = x_vals[train_indices]
x_vals_test = x_vals[test_indices]
y_vals_train = y_vals[train_indices]
y_vals_test = y_vals[test_indices]

A = tf.Variable(tf.random_normal(shape=[1,1]))
my_output = tf.matmul(x_data, A)

# loss
loss = tf.reduce_mean(tf.square(my_output - y_target))
init = tf.initialize_all_variables()
sess.run(init)

# optimizer
my_opt = tf.train.GradientDescentOptimizer(0.02)
train_step = my_opt.minimize(loss)
```

```
# Train
for i in range(1000):
  rand_index = np.random.choice(len(x_vals_train), size=batch_size)
  rand_x = np.transpose([x_vals_train[rand_index]])
  rand_y = np.transpose([y_vals_train[rand_index]])
  sess.run(train_step, feed_dict={x_data: rand_x, y_target: rand_y})
  if (i+1)%25 == 0:
    print('Step #' + str(i+1) + ' A= ' + str(sess.run(A)))
    print('Loss = ' + str(sess.run(loss, feed_dict={x_data: rand_x, \
                                                    y_target: rand_y})))

# Evaluate Model
mse_test = sess.run(loss, feed_dict={\
                      x_data: np.transpose([x_vals_test]),
                      y_target: np.transpose([y_vals_test])
                      })
mse_train = sess.run(loss, feed_dict={\
                      x_data: np.transpose([x_vals_train]),
                      y_target: np.transpose([y_vals_train])
                      })
print('MSE on test:' + str(np.round(mse_test, 2)))
print('MSE on train:' + str(np.round(mse_train,2)))
```

```
Step #750 A= [[9.768568]]
Loss = 1.3893672
Step #775 A= [[9.841519]]
Loss = 0.6651013
Step #800 A= [[9.825241]]
Loss = 1.6956891
Step #825 A= [[9.791046]]
Loss = 1.7853763
Step #850 A= [[9.833269]]
Loss = 1.4429283
Step #875 A= [[9.814627]]
Loss = 1.4609369
Step #900 A= [[9.800744]]
Loss = 1.3102401
Step #925 A= [[9.819744]]
Loss = 0.9488964
Step #950 A= [[9.830872]]
Loss = 0.7032228
Step #975 A= [[9.797334]]
Loss = 0.84393424
Step #1000 A= [[9.773134]]
Loss = 0.81731856
MSE on test:0.96
MSE on train:1.38
```

Neural Networks

- Operational Gates
- Activation Functions
- One-layer neural network
- Multi-layer neural network

```
%tensorflow_version 1.x
import tensorflow as tf
sess = tf.Session()

a = tf.Variable(tf.constant(4.))
x_val = 5.
x_data = tf.placeholder(dtype=tf.float32)


# output
multiplication = tf.multiply(a, x_data)

# define loss function
loss = tf.square(tf.subtract(multiplication, 50.))

# Initialize
init = tf.initialize_all_variables()
sess.run(init)

# Define optimization
my_opt = tf.train.GradientDescentOptimizer(0.01)
train_step = my_opt.minimize(loss)

# train

print('Optimizing a Multiplication Gate Output to 50.')
for i in range(10):
    sess.run(train_step, feed_dict={x_data: x_val})
    a_val = sess.run(a)
    mult_output = sess.run(multiplication, feed_dict={x_data:x_val})
    print(str(a_val) + ' * ' + str(x_val) + ' = ' + str(mult_output))
```

Operational Gates

- Learn the two functions:
  - $f(x) = a.x$
  - $f(x) = a.x + b$
- input a and b are declared as variables and x as a placeholder.
- Input value is fixed: $x = 5$ and target value is 50.
- In the first case, a converges to 10 which is the unique answer for this operator.
- Values of a and b are not unique in the second gate.

```
Optimizing a Multiplication Gate Output to 50.
7.0 * 5.0 = 35.0
8.5 * 5.0 = 42.5
9.25 * 5.0 = 46.25
9.625 * 5.0 = 48.125
9.8125 * 5.0 = 49.0625
9.90625 * 5.0 = 49.53125
9.953125 * 5.0 = 49.765625
9.9765625 * 5.0 = 49.882812
9.988281 * 5.0 = 49.941406
9.994141 * 5.0 = 49.970703
```

```python
from tensorflow.python.framework import ops
ops.reset_default_graph()
sess = tf.Session()
a = tf.Variable(tf.constant(1.))
b = tf.Variable(tf.constant(1.))
x_val = 5
x_data = tf.placeholder(dtype=tf.float32)
two_gate = tf.add(tf.multiply(a, x_data), b)
loss = tf.square(tf.subtract(two_gate, 50.))
my_opt = tf.train.GradientDescentOptimizer(0.01)
train_step = my_opt.minimize(loss)

init = tf.initialize_all_variables()
sess.run(init)

# train
print('\nOptimize Two Gate output to 50')
for i in range(10):
    sess.run(train_step, feed_dict={x_data: x_val})
    # Get a and b values
    a_val, b_val = (sess.run(a), sess.run(b))

    # get output
    two_gate_output = sess.run(two_gate, feed_dict={x_data: x_val})
    print(str(a_val) + ' * ' + str(x_val) + ' + ' + str(b_val) + \
                                                  ' = ' + str(two_gate_output))

```

```
Optimize Two Gate output to 50
5.4 * 5 + 1.88 = 28.88
7.512 * 5 + 2.3024 = 39.8624
8.52576 * 5 + 2.5051522 = 45.133953
9.012364 * 5 + 2.6024733 = 47.664295
9.2459345 * 5 + 2.6491873 = 48.87886
9.358048 * 5 + 2.67161 = 49.461853
9.411863 * 5 + 2.682373 = 49.74169
9.437695 * 5 + 2.687539 = 49.87601
9.450093 * 5 + 2.690019 = 49.940483
9.456045 * 5 + 2.6912093 = 49.971436
```

Working with Gates and Activation Functions

We will create two one-layer Neural network and see the effect of the following two activation functions:

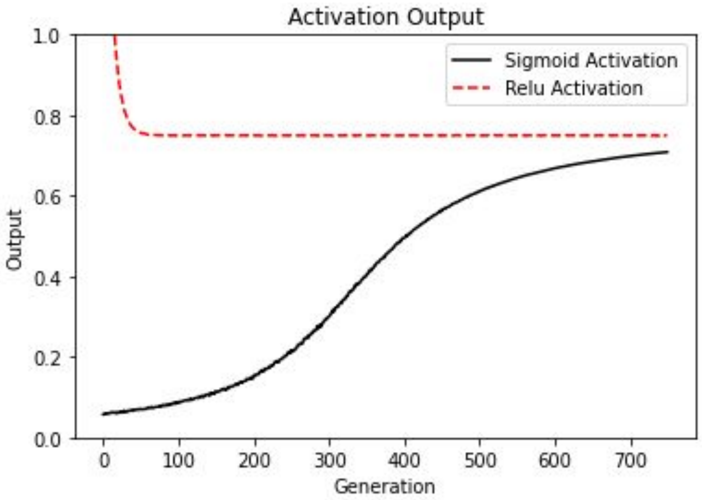
- $sigmoid(x) = \frac{1}{1+e^{x}}$
- $ReLU(x) = \max(0, x)$

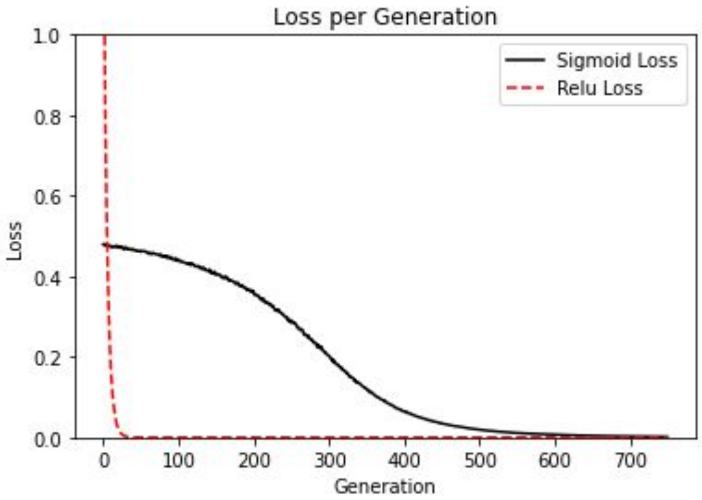
Observations:

- Relu converges faster compared to sigmoid activation
- Relu can lead to extreme output values.

```
%tensorflow_version 1.x
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
sess = tf.Session()
tf.set_random_seed(5)
np.random.seed(42)

# batch size, model variables, data and placeholde
batch_size = 50
a1 = tf.Variable(tf.random_normal(shape=[1,1]))
b1 = tf.Variable(tf.random_normal(shape=[1,1]))
a2 = tf.Variable(tf.random_normal(shape=[1,1]))
b2 = tf.Variable(tf.random_normal(shape=[1,1]))
x = np.random.normal(2, 0.1, 500)
x_data = tf.placeholder(shape=[None,1], dtype=tf.float32)

# declare two models
sigmoid_activation = tf.sigmoid(tf.add(tf.matmul(x_data, a1), b1))
relu_activation = tf.nn.relu(tf.add(tf.matmul(x_data,a2), b2))

# Loss functions for these models
loss1 = tf.reduce_mean(tf.square(tf.subtract(sigmoid_activation, 0.75)))
loss2 = tf.reduce_mean(tf.square(tf.subtract(relu_activation, 0.75)))

# declare optimization algorithms
my_opt = tf.train.GradientDescentOptimizer(0.01)
train_step_sigmoid = my_opt.minimize(loss1)
train_step_relu = my_opt.minimize(loss2)

# Initialize
init = tf.initialize_all_variables()
sess.run(init)
```

```python
# train
loss_vec_sigmoid = []
loss_vec_relu = []
activation_sigmoid = []
activation_relu = []
for i in range (750):
    rand_indices = np.random.choice(len(x), size=batch_size)
    x_vals = np.transpose([x[rand_indices]])
    sess.run(train_step_sigmoid, feed_dict={x_data: x_vals})
    sess.run(train_step_relu, feed_dict={x_data: x_vals})
    loss_vec_sigmoid.append(sess.run(loss1, feed_dict={x_data: x_vals}))
    loss_vec_relu.append(sess.run(loss2, feed_dict={x_data: x_vals}))

    activation_sigmoid.append(np.mean(sess.run(sigmoid_activation, \
                                                feed_dict={x_data: x_vals})))
    activation_relu.append(np.mean(sess.run(relu_activation, \
                                             feed_dict={x_data: x_vals})))

# Plot
plt.plot(activation_sigmoid, 'k-', label='Sigmoid Activation')
plt.plot(activation_relu, 'r--', label='Relu Activation')
plt.ylim([0, 1.0])
plt.title('Activation Output')
plt.xlabel('Generation')
plt.ylabel('Output')
plt.legend(loc='upper right')
plt.show()

plt.plot(loss_vec_sigmoid, 'k-', label='Sigmoid Loss')
plt.plot(loss_vec_relu, 'r--', label='Relu Loss')
plt.ylim([0, 1.0])
plt.title('Loss per Generation')
plt.xlabel('Generation')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.show()
```





Implementing a one-layer NN

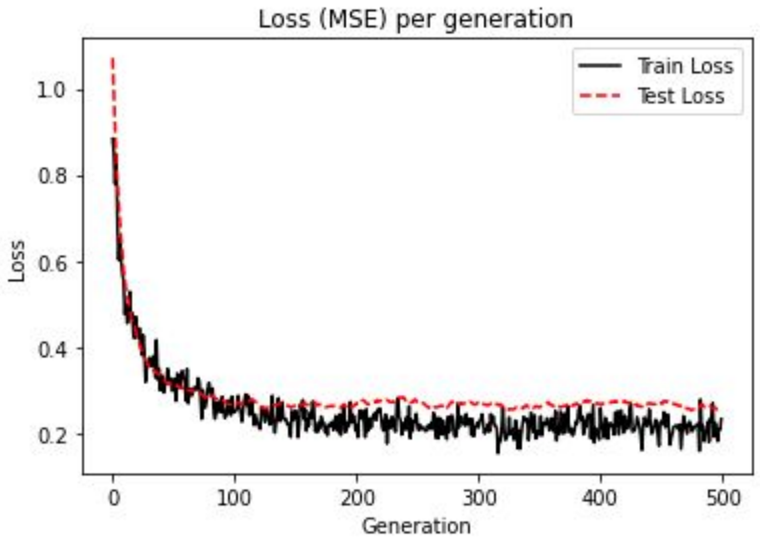
- We will create a one-layer NN which is applied to the IRIS dataset.
- The task is to find a function $x_4 = f(x_1, x_2, x_3)$
- express petal width (PL) as a function of sepal length, sepal width and petal length. In other words, PW = f(SL, SW, PL)
- It is based mostly on matrix multiplication and hence special attention should be paid to dimensions of various parameters.
- We normalize the input to the neural network
- There is no need to normalize the output.
- Network: 3-5-1 (3 Inputs, 5 hidden and 1 output nodes)

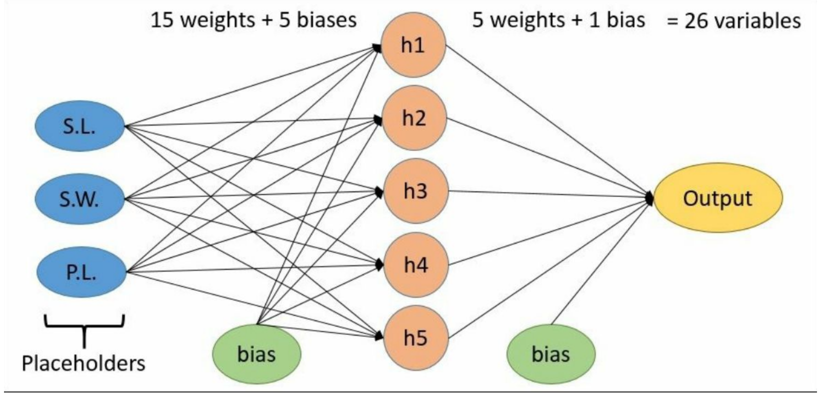
```
import matplotlib.pyplot as plt
import numpy as np
%tensorflow_version 1.x
import tensorflow as tf
from sklearn import datasets
from IPython.core.debugger import set_trace


# load IRIS dataset
iris = datasets.load_iris()
print('shape of IRIS dataset: {}'.format(np.shape(iris.data)))
x_vals = np.array([x[0:3] for x in iris.data])
y_vals = np.array([x[3] for x in iris.data])

sess = tf.Session()
seed = 2
tf.set_random_seed(seed)
np.random.seed(seed)

# Prepare the dataset with 80-20 train-test split
# features are normalized between 0 and 1

train_indices = np.random.choice(len(x_vals), round(len(x_vals)*0.8), \
                                 replace = False)
test_indices = np.array(list(set(range(len(x_vals))) - set(train_indices)))
x_vals_train = x_vals[train_indices]
x_vals_test = x_vals[test_indices]
y_vals_train = y_vals[train_indices]
y_vals_test = y_vals[test_indices]

def normalize_cols(m):
  col_max = m.max(axis=0)
  col_min = m.min(axis=0)
  return (m-col_min)/(col_max - col_min)

# normalize the input data
x_vals_train = np.nan_to_num(normalize_cols(x_vals_train))
x_vals_test = np.nan_to_num(normalize_cols(x_vals_test))
```

```python
# define placeholders, variables
batch_size = 50
x_data = tf.placeholder(shape=[None, 3], dtype=tf.float32)
y_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)
hidden_layer_nodes = 5
A1 = tf.Variable(tf.random_normal(shape=[3, hidden_layer_nodes]))
b1 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes]))
A2 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes, 1]))
b2 = tf.Variable(tf.random_normal(shape=[1]))


# NN model
hidden_output = tf.nn.relu(tf.add(tf.matmul(x_data, A1), b1))
final_output = tf.nn.relu(tf.add(tf.matmul(hidden_output, A2), b2))

# Loss function
loss = tf.reduce_mean(tf.square(y_target - final_output))

# declar optimizer to be used
my_opt = tf.train.GradientDescentOptimizer(0.01)
train_step = my_opt.minimize(loss)

# Initialize
init = tf.initialize_all_variables()
sess.run(init)
```

```python
#Train & Test
loss_vec = []
test_loss = []
for i in range(500):
  # select a random set of indices for the batch
  rand_index = np.random.choice(len(x_vals_train), size=batch_size)
  # select the training values
  rand_x = x_vals_train[rand_index]
  rand_y = np.transpose([y_vals_train[rand_index]])
  # run the training step
  sess.run(train_step, feed_dict={x_data: rand_x, y_target:rand_y})
  temp_loss = sess.run(loss, feed_dict={x_data: rand_x, y_target: rand_y})
  loss_vec.append(np.sqrt(temp_loss))
  test_temp_loss = sess.run(loss, feed_dict={x_data: x_vals_test, \
                          y_target: np.transpose([y_vals_test])})
  test_loss.append(np.sqrt(test_temp_loss))

  if (i+1)%50 == 0:
    print('Generation: ' + str(i+1) + ' Loss = ' + str(temp_loss))

# Plot
plt.plot(loss_vec, 'k-', label='Train Loss')
plt.plot(test_loss, 'r--', label='Test Loss')
plt.title('Loss (MSE) per generation')
plt.xlabel('Generation')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.show()
```

```
shape of IRIS dataset: (150, 4)
Generation: 50 Loss = 0.088808306
Generation: 100 Loss = 0.06148153
Generation: 150 Loss = 0.056416057
Generation: 200 Loss = 0.042471666
Generation: 250 Loss = 0.06005125
Generation: 300 Loss = 0.05051705
Generation: 350 Loss = 0.039771505
Generation: 400 Loss = 0.03431707
Generation: 450 Loss = 0.050341167
Generation: 500 Loss = 0.054154113
```





- 3-5-1 Network
- 26 parameters to tune
- S.L - Sepal Length
- S.W - Sepal Width
- P.L - Petal Length
- Output - Petal width (PW)