Chapter 2
Tensorflow

2.1 Tensorflow Preliminaries

```
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow.compat.v1 as tf

x = tf.constant("Hello World")
sess = tf.Session()
print(sess.run(x))
```

Python 3.7
Tensorflow 1.15

```
x = tf.constant(2.0)
z = tf.placeholder(tf.float32)
sess = tf.Session()
comp = tf.add(x,z)

print(sess.run(comp,feed_dict={z:3.0})) # Prints out 5.0
print(sess.run(comp,feed_dict={z:16.0})) # Prints out 18.0
print(sess.run(x)) # Prints out 2.0
print(sess.run(comp)) # Prints out a very long error message
```

```
x = 2.0
def sillyAdd(z):
return z+x
print(sillyAdd(3)) # Prints out 5.0
print(sillyAdd(16)) # Prints out 18.0
```

`print(sess.run(comp,feed_dict={z:3.0}))` → `tf.add(x,3.0)`

```
bt = tf.random_normal([10], stddev=.1)
b = tf.Variable(bt)
W = tf.Variable(tf.random_normal([784,10], stddev=.1))
sess = tf.Session()
sess.run(tf.global_variables_initializer())
print(sess.run(b))
```

```
[-0.05206999 0.08943175 -0.09178174 -0.13757218 0.15039739
0.05112269 -0.02723283 -0.02022207 0.12535755 -0.12932496]
```

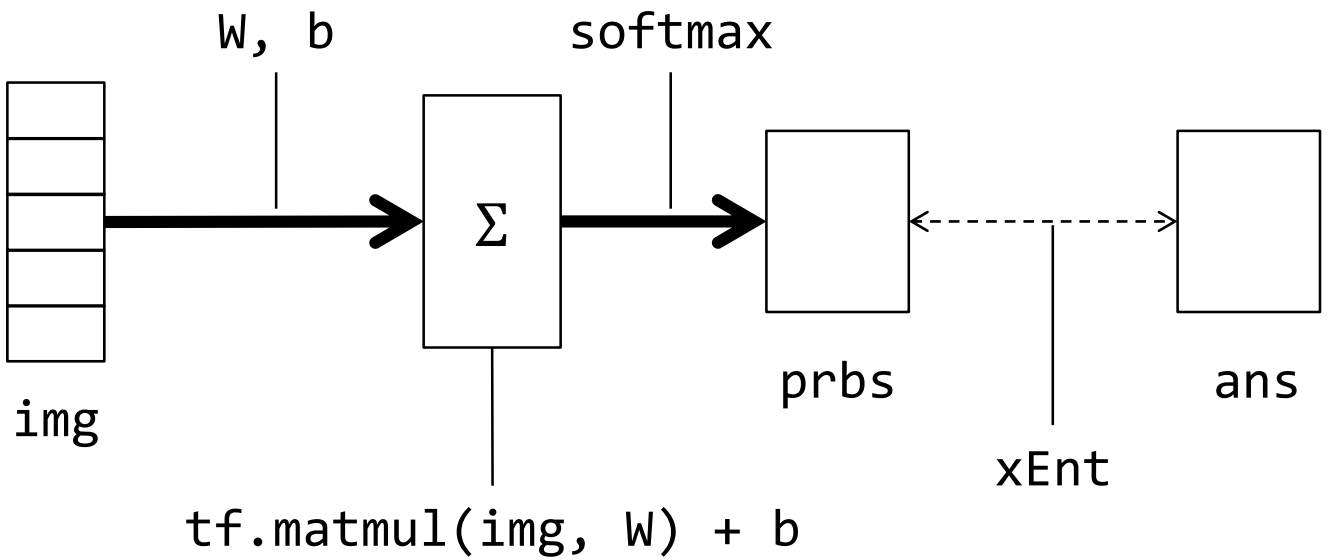
`sess.run(tf.global_variables_initializer())` → initializes b and W

2.2 A TF Program

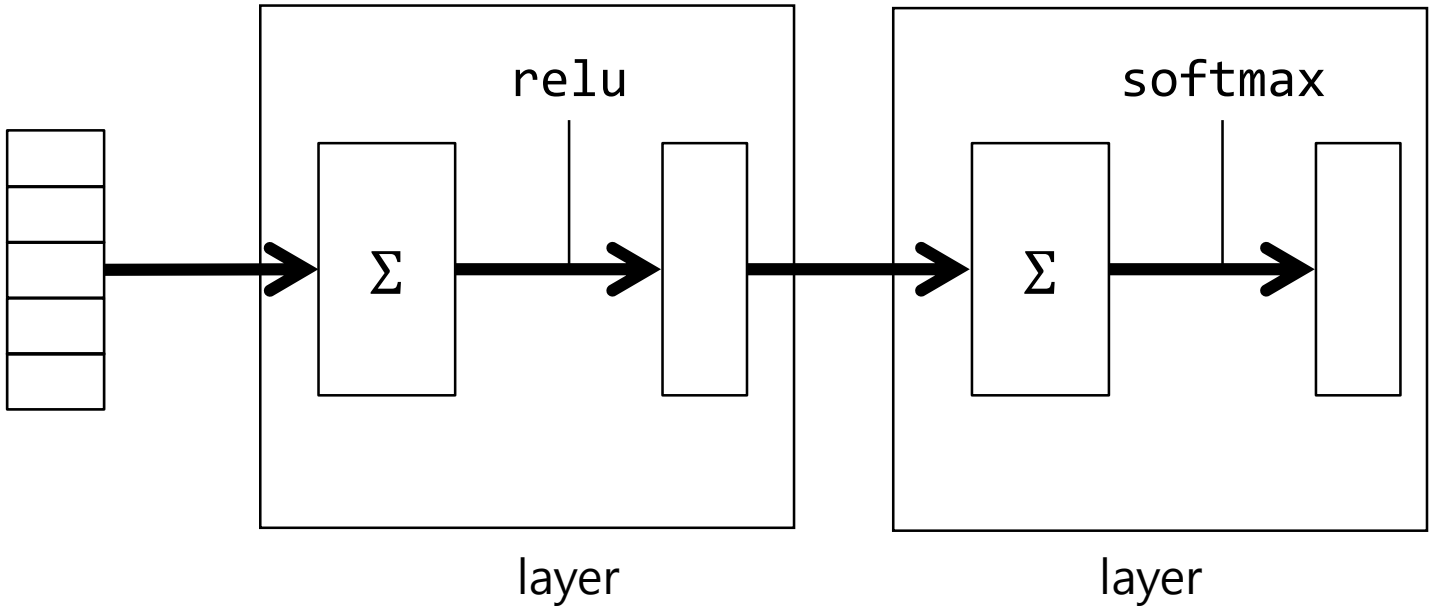
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow.compat.v1 as tf

old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
tf.logging.set_verbosity(old_v)
```

```
batchSz = 100

W = tf.Variable(tf.random_normal([784, 10], stddev=.1))
b = tf.Variable(tf.random_normal([10], stddev=.1))

img = tf.placeholder(tf.float32, [batchSz, 784])
ans = tf.placeholder(tf.float32, [batchSz, 10])

prbs = tf.nn.softmax(tf.matmul(img, W) + b)
xEnt = tf.reduce_mean(-tf.reduce_sum(ans * tf.log(prbs), reduction_indices=[1]))

train = tf.train.GradientDescentOptimizer(0.5).minimize(xEnt)
numCorrect = tf.equal(tf.argmax(prbs, 1), tf.argmax(ans, 1))
accuracy = tf.reduce_mean(tf.cast(numCorrect, tf.float32))

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

```
# ------------------------------------------------------------------
for i in range(1000):
    imgs, anss = mnist.train.next_batch(batchSz)
    sess.run(train, feed_dict={img: imgs, ans: anss})

sumAcc = 0
for i in range(1000):
    imgs, anss = mnist.test.next_batch(batchSz)
    sumAcc += sess.run(accuracy, feed_dict={img: imgs, ans: anss})
print("Test Accuracy: %r" % (sumAcc / 1000))
```



2.3 Multilayered NNs



ReLu

$$\rho(x) = \max(x, 0)$$

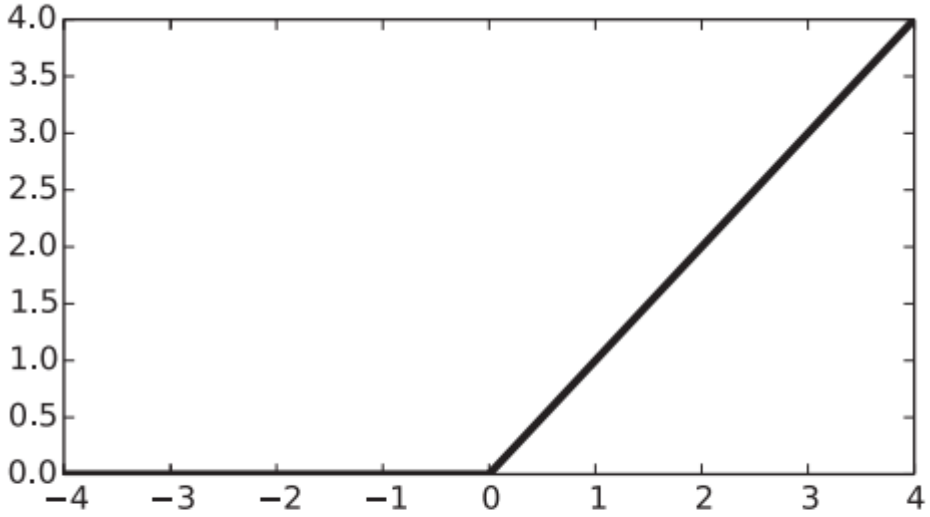


Figure 2.6: Behavior of `tf.nn.relu`

Sigmoid

$$S(x) = \frac{e^{-x}}{1 + e^{-x}}$$

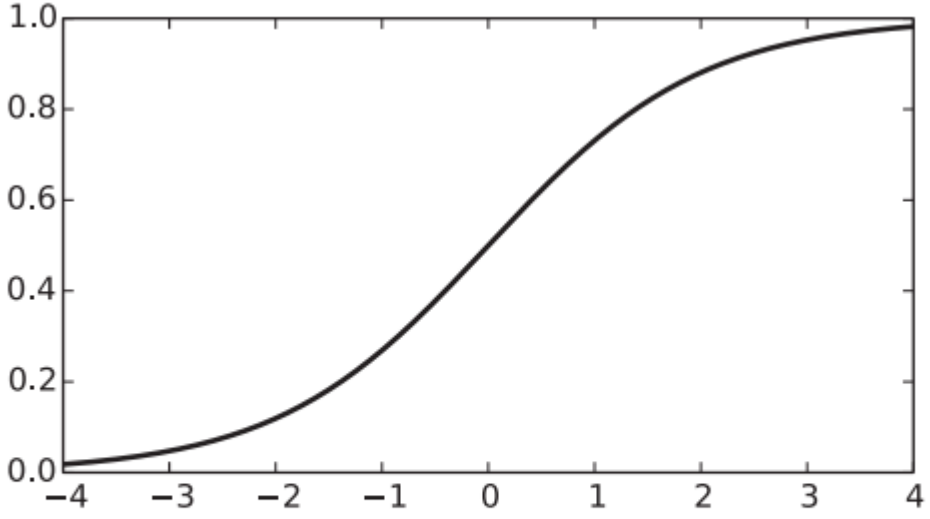
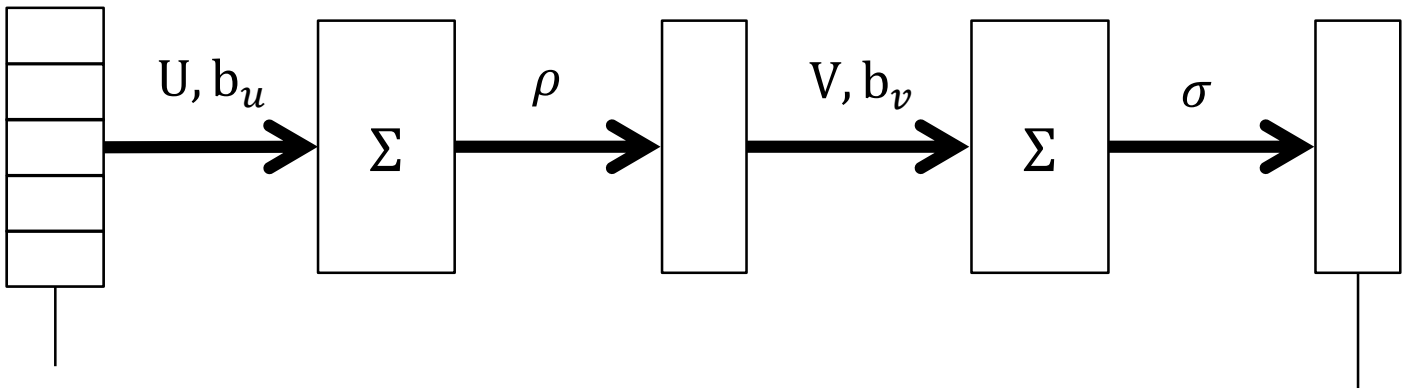


Figure 2.7: The sigmoid function




```
1 U = tf.Variable(tf.random_normal([784,784], stddev=.1))
2 bU = tf.Variable(tf.random_normal([784], stddev=.1))
3 V = tf.Variable(tf.random_normal([784,10], stddev=.1))
4 bV = tf.Variable(tf.random_normal([10], stddev=.1))
5 L1Output = tf.matmul(img,U)+bU
6 L1Output=tf.nn.relu(L1Output)
7 prbs=tf.nn.softmax(tf.matmul(L1Output,V)+bV)
```

Figure 2.9: TF graph construction code for multilevel digit recognition

```
batchSz = 100

U = tf.Variable(tf.random_normal([784, 784], stddev=.1))
bU = tf.Variable(tf.random_normal([784], stddev=.1))
V = tf.Variable(tf.random_normal([784, 10], stddev=.1))
bV = tf.Variable(tf.random_normal([10], stddev=.1))

img = tf.placeholder(tf.float32, [batchSz, 784])
ans = tf.placeholder(tf.float32, [batchSz, 10])

L1Output = tf.matmul(img, U) + bU
L1Output = tf.nn.relu(L1Output)
prbs = tf.nn.softmax(tf.matmul(L1Output, V) + bV)
xEnt = tf.reduce_mean(-tf.reduce_sum(ans * tf.log(prbs), reduction_indices=[1]))

train = tf.train.GradientDescentOptimizer(0.05).minimize(xEnt)
```

2.4 Other Pieces

- 2.4.1 Checkpointing
  - Before initializing variables (calling global variable initialize)

```
saveOb= tf.train.Saver()
```

  - Save

```
saveOb.save(sess, "mylatest.ckpt")
```

  - Restore

```
saveOb.restore(sess, "mylatest.ckpt")
```

- 2.4.2 tensordot

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} -1 & -2 \\ -3 & -4 \\ -5 & -6 \end{pmatrix}$$

  - Maxtrix multiplication

    `tf.matmul(A, B)`

  - tensordot

    `tf.tensordot(A, B, [[ 1 ], [ 0 ]])`

```
eo=  ( ((  1, 2, 3, 4),
        (  1, 1, 1, 1),
        (  1, 1, 1, 1),
        ( -1, 0,-1, 0)),
       ((  1, 2, 3, 4),
        (  1, 1, 1, 1),
        (  1, 1, 1, 1),
        ( -1, 0,-1, 0)) )
encOut=tf.constant(eo, tf.float32)

AT = (  ( .6, .25, .25 ),
        ( .2, .25, .25 ),
        ( .1, .25, .25 ),
        ( .1, .25, .25 ) )
wAT = tf.constant(AT, tf.float32)

encAT = tf.tensordot(encOut,wAT,[[1],[0]])
sess= tf.Session()

print sess.run(encAT)
[[[ 0.80000001  0.5         0.5       ]
  [ 1.50000012  1.          1.        ]
  [ 2.          1.          1.        ]
  [ 2.70000005  1.5         1.5       ]]
     ...]
```

- 2.4.3 Initialization of TF Variables

```
b = tf.Variable(tf.random_normal([10], stddev=.1))
```

  - Xavier initialization

$$\sigma = \sqrt{\frac{2}{n_i + n_o}}$$

- 2.4.4 Simplifying TF Graph Creation

```
layerOut=layers.fully_connected(layerIn,outSz,activeFn)
```

```
# W = tf.Variable(tf.random_normal([784, 10], stddev=.1))
# b = tf.Variable(tf.random_normal([10], stddev=.1))

img = tf.placeholder(tf.float32, [batchSz, 784])
ans = tf.placeholder(tf.float32, [batchSz, 10])

# prbs = tf.nn.softmax(tf.matmul(img, W) + b)
import tensorflow.contrib.layers as layers
prbs = layers.fully_connected(img, 10, tf.nn.softmax)
xEnt = tf.reduce_mean(-tf.reduce_sum(ans * tf.log(prbs), reduction_indices=[1]))
```