In [2]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score
import time
from sklearn.preprocessing import scale
```

In [4]:
```
dd_train = pd.read_csv('Training Data.csv')
dd_test = pd.read_csv('Testing Data.csv')

dd_real = pd.read_csv('Real Data.csv')
```

In [51]:
```
in_train = dd_train.values[:,1:129].astype(np.float64)
in_train = scale(in_train, axis = 1)
out_train = dd_train.label.astype('category')
out_train = pd.get_dummies( out_train ).values.astype(np.float64)
in_valid = dd_test.values[:,1:129].astype(np.float64)
in_valid = scale(in_valid, axis = 1)
out_valid = dd_test.label.astype('category')
out_valid = pd.get_dummies( out_valid ).values.astype(np.float64)

in_real = dd_real.values[:,1:129].astype(np.float64)
in_real = scale(in_real, axis = 1)
```

In [52]:
```
all_labels = np.unique(dd_test.label.astype('category'))
dd_real.label.values
```

Out[52]:
```
array(['Looter Scooter'], dtype=object)
```

In [53]:
```
out_real = np.reshape((all_labels == "Smuggler's Copter") +0,
[1,264]).astype(np.float64)
in_real.shape
```

Out[53]:
```
(1, 128)
```

In [54]:
```
N_train = in_train.shape[0]
N_feat = in_train.shape[1]
N_cat = out_train.shape[1]
N_valid = in_valid.shape[0]
N_real = in_real.shape[0]
```

In [55]:
```
N_train
```

Out[55]:
```
52800
```

In [56]:
```
# hyper-parameters
EPOCHS  = 300        # number of training epochs
N_nodes  = 256        # nodes in hidden layer
ALPHA   = 0          # regularization parameter
BS      = 52800         # batch size
STD     = 0.1        # weight initialization standard deviation
```

In [86]:
```
x_train = tf.placeholder( tf.float32, [BS, N_feat] )
y_train = tf.placeholder( tf.float32, [BS, N_cat] )

x_train_f = tf.constant( in_train , tf.float32, [N_train,N_feat])
y_train_f = tf.constant( out_train , tf.float32, [N_train,N_cat])

x_valid = tf.constant( in_valid , tf.float32, [N_valid , N_feat] )
y_valid = tf.constant( out_valid, tf.float32, [N_valid, N_cat])

x_real = tf.constant( in_real , tf.float32, [N_real , N_feat] )
y_real = tf.constant( out_real, tf.float32, [N_real, N_cat])
```

In [71]:
```
w1 = tf.Variable( tf.truncated_normal( [N_feat,N_nodes], stddev = STD,
seed=0))
b1 = tf.Variable( tf.truncated_normal( [1,N_nodes], stddev = STD, seed=0))

y1_train = tf.nn.relu( tf.matmul(x_train,w1) + b1 )
y1_train_f = tf.nn.relu( tf.matmul(x_train_f,w1) + b1 )
y1_valid = tf.nn.relu( tf.matmul( x_valid, w1) + b1 )
y1_real = tf.nn.relu( tf.matmul( x_real, w1) + b1 )

w2 = tf.Variable( tf.truncated_normal( [N_nodes,N_cat], stddev = STD, seed=0))
b2 = tf.Variable( tf.truncated_normal( [1,N_cat], stddev = STD, seed=0))

logits_train = tf.matmul(y1_train,w2) + b2
logits_train_f = tf.matmul(y1_train_f,w2) + b2
logits_valid = tf.matmul(y1_valid,w2) + b2
logits_real = tf.matmul(y1_real,w2) + b2
```

In [72]:
```
CE = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(
        logits_train, y_train) )
L2 = ALPHA*(tf.nn.l2_loss(w1) + tf.nn.l2_loss(b1) + \
               tf.nn.l2_loss(w2) + tf.nn.l2_loss(b2))
CE_train_f = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(
        logits_train_f, y_train_f) )
CE_valid = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(
        logits_valid, y_valid) )

y_train_p = tf.nn.softmax( logits_train )
y_train_fp = tf.nn.softmax( logits_train_f )
y_valid_p = tf.nn.softmax( logits_valid )
y_real_p = tf.nn.softmax( logits_real)

optimizer = tf.train.AdamOptimizer().minimize(CE+L2)
init = tf.initialize_all_variables()
```

In [73]:
```
sess = tf.Session()
sess.run(init)
```

In [74]:
```
t0 = time.time()
np.random.seed(0)

print( 'Epochs =%3d, Training Set Size =%4d, Nodes = %5d, Alpha = %3.4f, Batch
 Size = %4d, STD = %5.3f' %
      (EPOCHS, N_train, N_nodes, ALPHA, BS, STD))
print()
print('%15s%24s%24s' % (' ','cross-entropy','error-rate'))
print('%15s%12s%12s%12s%12s%12s%12s' %
      ('epoch','training','validation','training','validation','L2','time (mi
n)'))
for i in range(EPOCHS+1): # For every Epoch
    ran = np.random.permutation(N_train) # Order the data
    reran = np.reshape( ran, [ int(N_train/BS) ,BS] ) #Reshape ordering as a m
atrix
    for j in range( int(N_train/BS) ): # For every batch
        mini = reran[j,:].astype(int) # Find the batch indices
        x_batch = in_train[mini,:] # Call the batch features
        y_batch = out_train[mini] # Call the batch labels
        # Do a step with a batch
        sess.run([optimizer],feed_dict = {x_train:x_batch, y_train:y_batch})
    if (i % int(EPOCHS/10)) == 0:
        ( ce_train, ce_valid, out_train_pf, out_valid_p, l2 ) = \
        sess.run( [CE_train_f, CE_valid, y_train_fp, y_valid_p,L2])
        err_train = 1-accuracy_score( out_train.argmax(axis=1),
                                      out_train_pf.argmax(axis=1))
        err_valid  = 1-accuracy_score( out_valid.argmax(axis=1),
                                       out_valid_p.argmax(axis=1))
        t = (time.time() - t0)/60
        print('%7d %7d %12.5f %12.5f %12.6f %12.6f %12.3f %12.1f' %
              (EPOCHS,i,ce_train,ce_valid,err_train,err_valid,l2,t))
```

```
Epochs =300, Training Set Size =52800, Nodes =   256, Alpha = 0.0000, Batch S
ize = 52800, STD = 0.100

                        cross-entropy             error-rate
          epoch    training  validation    training  validation          L2
 time (min)
    300       0     5.85669     5.85707    0.997443    0.998030       0.
000          0.0
    300      30     1.45093     1.45758    0.081269    0.084508       0.
000          0.2
    300      60     0.09357     0.09711    0.003295    0.003750       0.
000          0.3
    300      90     0.03798     0.04135    0.001326    0.001932       0.
000          0.5
    300     120     0.02485     0.02827    0.000795    0.001515       0.
000          0.7
    300     150     0.01810     0.02155    0.000436    0.001174       0.
000          0.8
    300     180     0.01389     0.01733    0.000227    0.000985       0.
000          1.0
    300     210     0.01104     0.01445    0.000133    0.000871       0.
000          1.1
    300     240     0.00901     0.01238    0.000114    0.000795       0.
000          1.3
    300     270     0.00751     0.01083    0.000057    0.000720       0.
000          1.4
    300     300     0.00636     0.00963    0.000038    0.000682       0.
000          1.6
```

In [75]:
```
(out_real_p) = sess.run( [y_real_p])
```

In [89]:
```
all_labels[out_real_p[0].argmax()]
```

Out[89]: 'Demolish'

In [92]:
```
out_real_p = out_real_p[0]
out_real_p.argmax()
```

Out[92]: 60

In [93]:
```
all_labels[60]
```

Out[93]: 'Demolish'

In [94]:
```
out_real_p[60]
```

Out[94]: 0.34933463

In [ ]:
```

```