In [15]:
```
import pandas as pd
import numpy as np

%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
import plotly.offline as pyo
import plotly.graph_objs as go
import datetime as datetime
```

Crimes in Boston

- Times, locations and descriptions of crimes
- https://www.kaggle.com/AnalyzeBoston/crimes-in-boston (https://www.kaggle.com/AnalyzeBoston/crimes-in-boston)

Load and Prepare the Data.

In [16]:
```
data_path = 'data/crime.csv'


crime_data = pd.read_csv(data_path, encoding='gbk')
```

In [17]:
```
crime_data.head()
```

Out[17]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPOR
0	I182070945	619	Larceny	LARCENY ALL OTHERS	D14	
1	I182070943	1402	Vandalism	VANDALISM	C11	
2	I182070941	3410	Towed	TOWED MOTOR VEHICLE	D4	
3	I182070940	3114	Investigate Property	INVESTIGATE PROPERTY	D4	
4	I182070938	3114	Investigate Property	INVESTIGATE PROPERTY	B3	

In [18]:
```
crime_data.shape
```

Out[18]:
```
(319073, 17)
```

Dummy Variables

In [19]:
```
new_crime_data = crime_data[['DISTRICT',
                 'OFFENSE_CODE',
                 'OCCURRED_ON_DATE',
                 'HOUR',
                 'DAY_OF_WEEK']].copy()
new_crime_data.head()

dummy_fields = ['DISTRICT', 'OFFENSE_CODE', 'HOUR', 'DAY_OF_WEEK']
for each in dummy_fields:
    dummies = pd.get_dummies(new_crime_data[each], prefix=each, drop_first=False)
    new_crime_data = pd.concat([new_crime_data, dummies], axis=1)
```

Splitting the data into training, testing, and validation sets

In [20]:
```
# Save data for approximately 10000 time stamps
test_data = new_crime_data[-10000*24:]

# Now remove the test data from the data set
test_data = test_data[:-10000*24]

# Separate the data into features and targets
target_fields = ['DISTRICT', 'OFFENSE_CODE', 'DAY_OF_WEEK']
features, targets = new_crime_data.drop(target_fields, axis=1), new_crime_data[ta
rget_fields]
test_features, test_targets = test_data.drop(target_fields, axis=1), test_data[ta
rget_fields]
```

In [21]:
```
# Hold out the last 100000 timestamps or the remaining data as a validation set
train_features, train_targets = features[:-200000*24], targets[:-200000*24]
val_features, val_targets = features[-200000*24:], targets[-200000*24:]
```

In [22]:
```
class NeuralNetwork(object):
    def __init__(self, input_nodes, hidden_nodes, output_nodes, learning_rate):
        # Set number of nodes in input, hidden and output layers.
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes

        # Initialize weights
        self.weights_input_to_hidden = np.random.normal(0.0, self.input_nodes**-
0.5,
                                       (self.input_nodes, self.hidden_nodes))

        self.weights_hidden_to_output = np.random.normal(0.0, self.hidden_node
s**-0.5,
                                       (self.hidden_nodes, self.output_nodes))
        self.lr = learning_rate

        #### TODO: Set self.activation_function to your implemented sigmoid funct
ion ####
        #
        # Note: in Python, you can define a function with a lambda expression,
        # as shown below.
        self.activation_function = lambda x : (1 / (1 + np.exp(-x)) )  # Replace
0 with your sigmoid calculation.

        ### If the lambda code above is not something you're familiar with,
        # You can uncomment out the following three lines and put your
        # implementation there instead.
        #
        #def sigmoid(x):
        #    return 0  # Replace 0 with your sigmoid calculation here
        #self.activation_function = sigmoid

        #def sigmoid(x):
        #    return 1 / (1 + np.exp(-x))
        #self.activation_function = sigmoid


    def train(self, features, targets):
        ''' Train the network on batch of features and targets.

            Arguments
            ---------

            features: 2D array, each row is one data record, each column is a fea
ture
            targets: 1D array of target values

        '''
        n_records = np.array(features).shape[0]
        delta_weights_i_h = np.zeros(self.weights_input_to_hidden.shape)
        delta_weights_h_o = np.zeros(self.weights_hidden_to_output.shape)
        for X, y in zip(features, targets):
            #### Implement the forward pass here ####
            ### Forward pass ###
            # TODO: Hidden layer - Replace these values with your calculations.
            hidden_inputs = np.dot(X, self.weights_input_to_hidden) # signals int
o hidden layer
            hidden_outputs = self.activation_function(hidden_inputs) # signals fr
om hidden layer

            # TODO: Output layer - Replace these values with your calculations.
            final_inputs = np.dot(hidden_outputs, self.weights_hidden_to_output)
# signals into final output layer
            final_outputs = final_inputs # signals from final output layer
```

In [24]:
```
import unittest

inputs = np.array([[0.5, -0.2, 0.1]])
targets = np.array([[0.4]])
test_w_i_h = np.array([[0.1, -0.2],
                       [0.4, 0.5],
                       [-0.3, 0.2]])
test_w_h_o = np.array([[0.3],
                       [-0.1]])

class TestMethods(unittest.TestCase):

    ##########
    # Unit tests for data loading
    ##########

    def test_data_path(self):
        # Test that file path to dataset has been unaltered
        self.assertTrue(data_path.lower() == 'data/crime.csv')

    def test_data_loaded(self):
        # Test that data frame loaded
        self.assertTrue(isinstance(new_crime_data, pd.DataFrame))

    ##########
    # Unit tests for network functionality
    ##########

    def test_activation(self):
        network = NeuralNetwork(3, 2, 1, 0.5)
        # Test that the activation function is a sigmoid
        self.assertTrue(np.all(network.activation_function(0.5) == 1/(1+np.exp(-
0.5))))

    def test_train(self):
        # Test that weights are updated correctly on training
        network = NeuralNetwork(3, 2, 1, 0.5)
        network.weights_input_to_hidden = test_w_i_h.copy()
        network.weights_hidden_to_output = test_w_h_o.copy()

        network.train(inputs, targets)
        self.assertTrue(np.allclose(network.weights_hidden_to_output,
                                    np.array([[ 0.37275328],
                                              [-0.03172939]])))
        self.assertTrue(np.allclose(network.weights_input_to_hidden,
                                    np.array([[ 0.10562014, -0.20185996],
                                              [0.39775194, 0.50074398],
                                              [-0.29887597, 0.19962801]])))

    def test_run(self):
        # Test correctness of run method
        network = NeuralNetwork(3, 2, 1, 0.5)
        network.weights_input_to_hidden = test_w_i_h.copy()
        network.weights_hidden_to_output = test_w_h_o.copy()

        self.assertTrue(np.allclose(network.run(inputs), 0.09998924))

suite = unittest.TestLoader().loadTestsFromModule(TestMethods())
unittest.TextTestRunner().run(suite)
```

```
.....
----------------------------------------------------------------------
Ran 5 tests in 0.011s

OK
```

Out[24]:
```
<unittest.runner.TextTestResult run=5 errors=0 failures=0>
```

In [27]:
```
### Set the hyperparameters here ###
iterations = 1000
learning_rate = .15
hidden_nodes = 128
output_nodes = 1

N_i = train_features.shape[1]
network = NeuralNetwork(N_i, hidden_nodes, output_nodes, learning_rate)

losses = {'train':[], 'validation':[]}
for ii in range(iterations):
    # Go through a random batch of 256 records from the training data set
    batch = np.random.choice(train_features.index, size=256)
    X, y = train_features.ix[batch].values, train_targets.ix[batch]['DISTRICT']

    network.train(X, y)

    # Printing out the training progress
    train_loss = MSE(network.run(train_features).T, train_targets['DISTRICT'].val
ues)
    val_loss = MSE(network.run(val_features).T, val_targets['DISTRICT'].values)
    sys.stdout.write("\rProgress: {:2.1f}".format(100 * ii/float(iterations)) \
                     + "% ... Training loss: " + str(train_loss)[:5] \
                     + " ... Validation loss: " + str(val_loss)[:5])
    sys.stdout.flush()

    losses['train'].append(train_loss)
    losses['validation'].append(val_loss)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-47fbbed69adb> in <module>
     11 for ii in range(iterations):
     12     # Go through a random batch of 256 records from the training data se
t
---> 13     batch = np.random.choice(train_features.index, size=256)
     14     X, y = train_features.ix[batch].values, train_targets.ix[batch]['DIS
TRICT']
     15

mtrand.pyx in mtrand.RandomState.choice()

ValueError: 'a' cannot be empty unless no samples are taken
```

In [ ]:
```
```