ECE521
Assignment 1:
k Nearest Neighbours (k-NN)

TA: Use Piazza for Q&A
Due date: Feb 2
Submission: A hard copy of the report should be handed in the lecture
The code should be submitted electronically to: ece521ta2018@gmail.com

Objectives:

- In this assignment, you will implement k-NN using TensorFlow, apply your implementation to some simulated and real datasets, and analyze the results. You will use k-NN for both regression and classification tasks.

General Notes:

- You are encouraged to look up TensorFlow APIs for useful utility functions to help you complete the programming portion of the assignment at: `https://www.tensorflow.org/api_docs/python/`.
- Full points are given for complete solutions, including justifying the choices or assumptions you made to solve each question. Both your complete source code and written up report should be included in the final submission.
- Homework assignments are to be solved in groups of **three**. You are encouraged to discuss the assignment with other students, but you must solve it within your own group. Make sure to be closely involved in all aspects of the assignment. Please indicate the contribution percentage from each group member at the beginning of your report.

Introduction

k-NN is one of the simplest nonparametric machine learning models. In this assignment, you will first implement the basic k-NN algorithm in TensorFlow. You will then use it for both regression and classification tasks.

In this assignment, when $k > 1$, we will aggregate the k nearest neighbours' predictions by averaging their target values for regression tasks. For classification tasks, a majority voting scheme is assumed.

1 Euclidean distance function [4 points]

The squared Euclidean distance between two D-dimensional vectors $\mathbf{x}$ and $\mathbf{z}$ is given by $\|\mathbf{x}-\mathbf{z}\|_2^2 = (\mathbf{x}-\mathbf{z})^\top(\mathbf{x}-\mathbf{z}) = \sum_{d=1}^{D}(x_d - z_d)^2$. Assume we have two groups of D-dimensional data points represented respectively by a $N_1 \times D$ matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)^T} \\ \vdots \\ \mathbf{x}^{(N_1)^T} \end{bmatrix}$ and a $N_2 \times D$ matrix $\mathbf{Z} = \begin{bmatrix} \mathbf{z}^{(1)^T} \\ \vdots \\ \mathbf{z}^{(N_2)^T} \end{bmatrix}$. Let the function $D_{euc}(\mathbf{X}, \mathbf{Z})$ returns a $N_1 \times N_2$ matrix $\begin{bmatrix} \|\mathbf{x}^{(1)} - \mathbf{z}^{(1)}\|_2^2 & \dots & \|\mathbf{x}^{(1)} - \mathbf{z}^{(N_2)}\|_2^2 \\ \vdots & \ddots & \vdots \\ \|\mathbf{x}^{(N_1)} - \mathbf{z}^{(1)}\|_2^2 & \dots & \|\mathbf{x}^{(N_1)} - \mathbf{z}^{(N_2)}\|_2^2 \end{bmatrix}$ containing the pairwise Euclidean distances.

Pairwise distance [4 pt.]:
Write a vectorized Tensorflow Python function that implements the pairwise squared Euclidean distance function for two input matrices. It should not contain loops and you should make use of Tensorflow broadcasting. Include the snippets of the Python code.

2 Making Predictions for Regression [6 points]

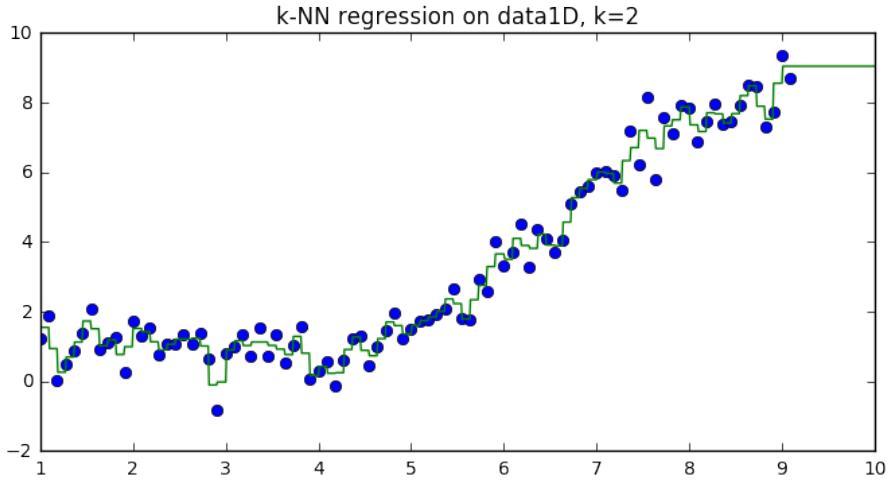
Suppose a particular training dataset consists of N training examples with input features $\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)^T} \\ \vdots \\ \mathbf{x}^{(N)^T} \end{bmatrix}$ and targets $\mathbf{Y} = \begin{bmatrix} \mathbf{y}^{(1)^T} \\ \vdots \\ \mathbf{y}^{(N)^T} \end{bmatrix}$ represented by two matrices. In k-NN modelling for regression, a standard way to obtain a prediction for a new test point $\mathbf{x}^*$ is to average the targets/labels of the k training points nearest to the test point $\mathbf{x}^*$. Let $\mathcal{N}_{\mathbf{x}^*}^k \subseteq \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ denote the neighbourhood set of the k training examples closest to $\mathbf{x}^*$. A k-NN prediction function $\hat{\mathbf{y}}(\mathbf{x}^*)$ can therefore be written concisely as follows:

$$\hat{\mathbf{y}}(\mathbf{x}^*) = \mathbf{Y}^T\mathbf{r}^*, \text{ where } \mathbf{r}^* = [r_1, \dots, r_N], r_n = \begin{cases} \frac{1}{k}, & \mathbf{x}^{(n)} \in \mathcal{N}_{\mathbf{x}^*}^k \\ 0, & \text{otherwise.} \end{cases}$$

$\mathbf{r}^*$ can be understood as a "responsibility" vector that encodes the contribution of each training example to the final prediction. In a standard k-NN model, the k nearest training points will have equal contribution. To measure the performance of the k-NN predictions, we will consider the mean squared error (MSE) loss:

$$\mathcal{L} = \frac{1}{2N}\sum_{n=1}^{N}\|\hat{\mathbf{y}}(\mathbf{x}^{(n)}) - \mathbf{y}^{(n)}\|_2^2$$

The first dataset for this assignment is an 1-D toy dataset, *data1D*, with 100 data points. We randomly partition 80 data points as training examples and split the rest into 10 validation points



and 10 test points. You can generate the training and test dataset using the following python commands:

```
import numpy as np
np.random.seed(521)
Data = np.linspace(1.0 , 10.0 , num =100) [:, np. newaxis]
Target = np.sin( Data ) + 0.1 * np.power( Data , 2) \
         + 0.5 * np.random.randn(100 , 1)
randIdx = np.arange(100)
np.random.shuffle(randIdx)
trainData, trainTarget  = Data[randIdx[:80]], Target[randIdx[:80]]
validData, validTarget = Data[randIdx[80:90]], Target[randIdx[80:90]]
testData, testTarget = Data[randIdx[90:100]], Target[randIdx[90:100]]
```

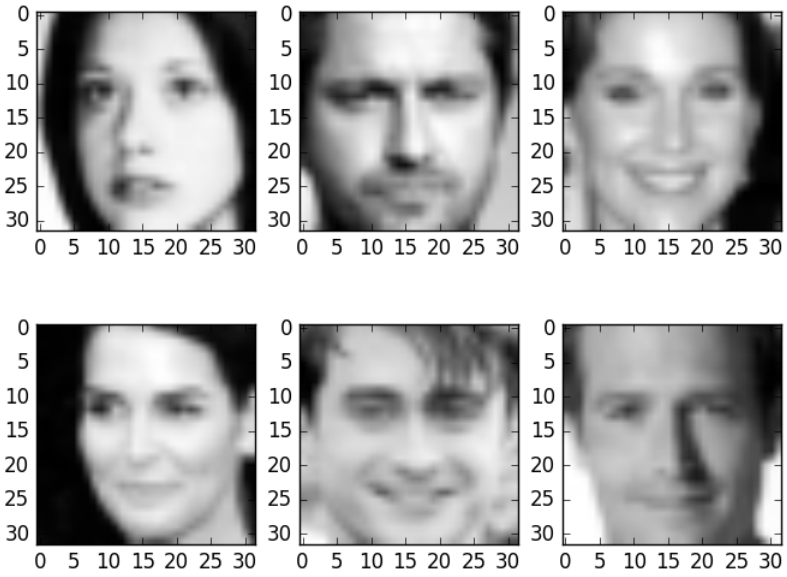
1. **Choosing the nearest neighbours [2 pt.]:**
   Write a vectorized Tensorflow Python function that takes a pairwise distance matrix and returns the responsibilities of the training examples to a new test data point. It should not contain loops. You may find the *tf.nn.top_k* function useful. Include the relevant snippets of your Python code.

2. **Prediction [4 pt.]:**
   For the dataset *data1D*, compute the above k-NN prediction function with $k = \{1, 3, 5, 50\}$. For each of these values of k, compute and report the training MSE loss, validation MSE loss and test MSE loss. Choose the best k using the validation error.
   For $k = \{1, 3, 5, 50\}$, plot the prediction function for $x \in [0, 11]$ along with the true targets similar to the figure shown above. (You may create an input matrix $X = np.linspace(0.0, 11.0, num = 1000)[:, np.newaxis]$ and use Matplotlib to plot the predictions.) Comment on this plot and how you would pick the best k from it.

3 Making Predictions for Classification [10 points]

In this section, k-NN will be used for classification task. You will modify the prediction part of the k-NN algorithm and apply it for face and gender classification tasks. You will use a tiny version of the FaceScrub dataset[1]. The FaceScrub dataset consists of 100,000+ images of 500+ celebrities. The tiny version consists only of six celebrities (three actors and three actress) with average of 150 image per subject. The images were downloaded and converted to grey using the available code in http://www.cs.toronto.edu/~guerzhoy/321/proj1/, then, the faces were cropped and resized to 32×32 images and stored as .npy file. You are provided two .npy files; the *data.npy* file is an array of size 936×32×32 which contains 936 images of the six celebrities, and the *target.npy* file that encodes the class of each image and of size 936×2. The first column encodes the name (ID) of the actor and the second column encodes the gender as follows:

- The name (ID) of the actors: 'Lorraine Bracco', 'Gerard Butler', 'Peri Gilpin', 'Angie Harmon', 'Daniel Radcliffe', and 'Michael Vartan' are encoded as '0', '1', '2', '3', '4', and '5', respectively.
- The gender of the actors: 'Male' and 'Female' are encoded as '0' and '1', respectively.



You should use the following code to load the dataset.

```
def data_segmentation(data_path, target_path, task):
    # task = 0 >> select the name ID targets for face recognition task
    # task = 1 >> select the gender ID targets for gender recognition task
    data = np.load(data_path)/255
    data = np.reshape(data, [-1, 32*32])
```

[1] http://vintage.winklerbros.net/facescrub.html

```
target = np.load(target_path)

np.random.seed(45689)
rnd_idx = np.arange(np.shape(data)[0])
np.random.shuffle(rnd_idx)

trBatch = int(0.8*len(rnd_idx))
validBatch = int(0.1*len(rnd_idx))

trainData, validData, testData = data[rnd_idx[1:trBatch],:], \
                                 data[rnd_idx[trBatch+1:trBatch + validBatch],:],\
                                 data[rnd_idx[trBatch + validBatch+1:-1],:]

trainTarget, validTarget, testTarget = target[rnd_idx[1:trBatch], task], \
                            target[rnd_idx[trBatch+1:trBatch + validBatch], task],\
                            target[rnd_idx[trBatch + validBatch + 1:-1], task]

return trainData, validData, testData, trainTarget, validTarget, testTarget
```

1. **Predicting class label [4 pt.]:**
   Modify the prediction function for regression in section 1 and use majority voting over k nearest neighbors to predict the final class. You may find *tf.unique_with_counts* helpful for this task. Include the relevant snippet of code for this task.

2. **Face recognition using k-NN [4 pt.]:**
   Report the recognition accuracy for face recognition using k-NN for different values of k over the validation dataset. Use $k = \{1, 5, 10, 25, 50, 100, 200\}$. Find the value of k that achieves the best validation accuracy and use this value to find the accuracy on the test set. Also, for $k = 10$, display one failure case where the majority votes on the k nearest neighbors is not the same as the true person (display the test image and it's k-nearest images) and comment on your results.

3. **Gender recognition using k-NN [2 pt.]:**
   Repeat part (3) for the gender recognition problem.