Homework 6: Due Tue 09-25-2018
Total Points (40 pts)

1. (10 pts) **Numerical Computation of Jacobian Matrix** Consider the function

$$\mathbf{y} = f(\mathbf{x}) \text{ where } \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ and } f(\mathbf{x}) = \begin{pmatrix} \sqrt{x_1 x_2} \\ x_2^2 + x_1 e^{x_2} \end{pmatrix}.$$

   (a) Compute the Jacobian matrix $\frac{d\mathbf{y}}{d\mathbf{x}}$ by hand and evaluate it at the point $\mathbf{x} = \begin{pmatrix} 1 & 1 \end{pmatrix}^\top$.

   (b) Use the central difference formula to numerically compute the Jacobian matrix $\frac{d\mathbf{y}}{d\mathbf{x}}$ at $\mathbf{x} = \begin{pmatrix} 1 & 1 \end{pmatrix}^\top$. (Use the numpy code given below.)

```
import numpy as np

def dfdx(f,x):
    eps = 10**(-5)
    y = f(x)
    n = len(x)
    m = len(y)
    dfdx = np.zeros((m,n))
    for k in range(n):
        dx = np.zeros(n)
        dx[k] = eps
        fp = f(x + dx)
        fm = f(x - dx)
        dfdx[:,k] = (fp - fm)/(2*eps)
    return dfdx
```

2. (10 pts) **Jacobian Matrix of Softmax Function** Define a function in Python that computes the exact (not numerical) Jacobian matrix of the softmax function. Your function should work for any number of inputs $\mathbf{x} = \begin{pmatrix} x_1, \ldots, x_N \end{pmatrix}^\top$ and avoid overflow. Use the function `dfdx(f,x)` defined above to numerically check the output of your function. Avoid using for loops by "vectorizing" your code. Hint: The softmax Jacobian matrix $\frac{d\mathbf{p}}{d\mathbf{x}}$ can be programming in three lines using: `np.exp`, `.max()`, `.sum()`, `np.diag`, `np.outer`.

3. (10 pts) **Linear Regression Using Keras** The `Boston_home_prices.csv`[1] data set contains data on 13 attributes and median home prices of homes in suburbs of Boston. Use `Keras` to train a linear model for predicting home prices. Report training MSE for your model and compare it to a baseline.

```
Attributes:
1. CRIM      per capita crime rate by town
2. ZN        proportion of residential land zoned for lots over
             25,000 sq.ft.
3. INDUS     proportion of non-retail business acres per town
```

[1] UCI Machine Learning Repository

```
4. CHAS      Charles River dummy variable (= 1 if tract bounds
             river; 0 otherwise)
5. NOX       nitric oxides concentration (parts per 10 million)
6. RM        average number of rooms per dwelling
7. AGE       proportion of owner-occupied units built prior to 1940
8. DIS       weighted distances to five Boston employment centres
9. RAD       index of accessibility to radial highways
10. TAX      full-value property-tax rate per $10,000
11. PTRATIO  pupil-teacher ratio by town
12. B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks
             by town
13. LSTAT    % lower status of the population
14. MEDV     Median value of owner-occupied homes in $1000's
```

4. (10 pts) L^2 **Regularized Logistic Regression Using Keras** Use `Keras` to train an L^2 regularized, logistic regression classifier for predicting the number corresponding to the image of a character. Use the dataset `MNIST_train_100.npz`. Explore how training and validation accuracy varies with the L^2 regularization parameter in the following way:

   Split the training set into 80% training 20% validation datasets and plot training and validation accuracy on the same axes as a function of the number of iterations (epochs) of the optimizer. Also include a line identifying a baseline level of performance. Describe what happens to training and testing accuracy when the L^2 regularization parameter is $\alpha = 0, 1, 10, 100, 1000$. Useful commands are listed below.

```
from keras import regularizers

model.add(Dense(...,kernel_regularizer=regularizers.l2(10)))

hist = model.fit(...,validation_split=0.2)

accuracy = pd.DataFrame()
accuracy['epoch']    = hist.epoch
accuracy['epoch']    = accuracy['epoch'] + 1
accuracy['training'] = hist.history['acc']
accuracy['testing']  = hist.history['val_acc']
accuracy['baseline'] = baseline
accuracy.head()

ax = accuracy.plot.line(x='epoch',y='training')
ax = accuracy.plot.line(x='epoch',y='testing',ax=ax)
accuracy.plot.line(x='epoch',y='baseline',ax=ax)
```