Homework 3 – Deep Neural Networks (CS525 191N, Whitehill, Spring 2017)

You may complete this homework assignment either individually or in teams up to 2 people.

1. **Newton's method** [10 points]: Show that, for a 2-layer linear neural network (i.e., $\hat{y} = f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$) and the same cost function J as from Homework 2, Newton's method (see Equation 4.12 in *Deep Learning*) will converge to the optimal solution $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$ in 1 iteration no matter what the starting point $\mathbf{w}_0$ of the search is.

2. **Whitening transformation** [15 points]: Using the same smile data as in Homework 2, perform a whitening transformation on the input data (`trainingFaces`) prior to running gradient descent. Verify that the eigenvalues of the covariance matrix of the transformed faces are almost all close to 1. After transforming the input data, the gradient descent procedure should tolerate a much higher learning rate and converge much faster. Note that this kind of whitening is a form of **data preprocessing** which is sometimes also called **normalization** in neural networks literature. In the optimization literature, a similar technique is known as **preconditioning**.

   Implement the transformation inside a method called `whiten(faces)` which takes a Numpy array of faces (rows: examples; columns: pixels) as input and returns a (transformed) array of faces of the same dimensions.

   Update: *please include a screenshot showing the entire gradient descent trajectory (i.e., the training costs from the beginning through the end of training) when you conduct gradient descent on the whitened training data. It should only take a few iterations to converge.*

3. **Logistic regression** [15 points]: Using the same smile data as in Homework 2, train a 2-layer neural network using gradient descent to output a probability of smile ($\hat{y} \in (0, 1)$ instead of $\hat{y} \in \mathbb{R}$) by making the output unit of the network be a sigmoid unit (using the logistic sigmoid function: $\sigma(z) = \frac{1}{1+\exp(-z)}$) and using the (regularized) cross-entropy loss function (for 2 classes):

$$J(\mathbf{w}) = -\frac{1}{m} \sum_{j=1}^{m} [y_j \log \hat{y}_j + (1 - y_j) \log(1 - \hat{y}_j)] + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}$$

   You can use `check_grad` to verify your gradient expression is correct. To check your final solution (with $\alpha = 0$), you can compare against `sklearn.linear_model.LogisticRegression`, where the `C` parameter (which is the inverse of α) should be set to some very large number and the `fit_intercept` term should be `False`.

   Implement the optimization inside a method called `method4(trainingFaces, trainingLabels)` that is analogous to `method1` (etc.) from Homework 2.

   Update: *please include a screenshot showing the training costs during the last 20 iterations of the gradient descent trajectory when you conduct gradient descent on the* unwhitened *training data.*

Put Problem 1 in a PDF file called `homework3_WPIUSERNAME1.pdf`
(or `homework3_WPIUSERNAME1_WPIUSERNAME2.pdf` for teams). Put Problems 2 and 3 in a Python file called `homework3_WPIUSERNAME1.py`
(or `homework3_WPIUSERNAME1_WPIUSERNAME2.py` for teams).

Update: Please put the screenshots inside the same PDF as you include for Problem 1.