Homework 2 – Deep Neural Networks (CS525 191D, Whitehill, Spring 2018)

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

1. **XOR problem** [10 points]: Show (by deriving the gradient, setting to 0, and solving mathematically, not in Python) that the values for $\mathbf{w} = (w_1, w_2)$ and b that minimize the function $J(\mathbf{w}, b)$ in Equation 6.1 (in the *Deep Learning* textbook) are: $w_1 = 0$, $w_2 = 0$, and $b = 0.5$. Put your solution in a PDF file called `homework2_WPIUSERNAME1.pdf`
(or `homework2_WPIUSERNAME1_WPIUSERNAME2.pdf` for teams).

2. **Smile detector**: Train a simple "smile detector" that analyzes a $(24 \times 24 = 576)$-pixel grayscale face image and outputs a real number $\hat{y}$ representing whether or not the image is smiling ($\hat{y}$ close to 1 means "smile"; $\hat{y}$ close to 0 means "non-smile"). Your detector should be implemented as a neural network $f_{\mathbf{w}} : \mathbb{R}^{576} \to \mathbb{R}$ consisting of just an input layer and an output layer, with no layers in between. (This network is therefore not very "deep", but you have to start somewhere.) **Note 1**: you must complete this problem using only linear algebraic operations in `numpy` – you may **not** use any off-the-shelf linear regression or neural network training software, as that would defeat the purpose. **Note 2**: If you have OpenCV 2.4.13 or higher, you can run a real-time demo (uncomment the corresponding lines in `homework2_template.py`) of the smile detector you train.

   (a) **Method 1 – set gradient to 0 and solve** [12 points]: Compute the parameters $\mathbf{w} = (w_1, \ldots, w_{576})$ representing the "weights"/parameters of the neural network by deriving the expression for the gradient of the cost function w.r.t. $\mathbf{w}$, setting it to 0, and then solving. The cost function is

   $$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

   where $\hat{y} = f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ and m is the number of examples in the training set $\mathcal{D}_{\text{tr}} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$, each $\mathbf{x}^{(i)} \in \mathbb{R}^{576}$ and each $y^{(i)} \in \{0, 1\}$. Note that this "one-shot" method of optimizing the neural network parameters only works in **very particular cases** (such as linear regression, as we have here). After optimizing $\mathbf{w}$ only on the **training set**, compute and report the cost J on the training set $\mathcal{D}_{\text{tr}}$ and (separately) on the testing set $\mathcal{D}_{\text{te}}$.

   (b) **Method 2 – gradient descent** [12 points]: Pick a random starting value for $\mathbf{w} \in \mathbb{R}^{576}$ and a small learning rate ($\epsilon \ll 1$). Then, using the expression for the gradient of the cost function, iteratively update $\mathbf{w}$ to reduce the cost $J(\mathbf{w})$. Stop when the difference between J over successive training rounds is below some "tolerance" (e.g., $\delta = 0.001$). After optimizing $\mathbf{w}$ only on the **training set**, compute and report the cost J on the training set $\mathcal{D}_{\text{tr}}$ and (separately) on the testing set $\mathcal{D}_{\text{te}}$. Both of these values should be very close to what you computed using Method 1. Note that this method of optimizing neural network parameters is **much more general** than Method 1 above, at the expense of requiring some additional optimization hyperparameters (e.g., learning rate, tolerance).

   (c) **Method 3 – gradient descent with regularization** [6 points]: Same as (b) above, but change the cost function to include a penalty for $|\mathbf{w}|^2$ growing too large:

   $$\tilde{J}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 + \frac{\alpha}{2}\mathbf{w}^\top \mathbf{w}$$

   where $\alpha \in \mathbb{R}^+$. Set $\alpha = 1000$ and then optimize $\tilde{J}$ w.r.t. $\mathbf{w}$. After optimizing $\mathbf{w}$ only on the **training set** (using $\tilde{J}$), compute and report the *unregularized* cost J on the training set $\mathcal{D}_{\text{tr}}$ and (separately) the testing set $\mathcal{D}_{\text{te}}$. The training cost should be higher (i.e., worse), but the testing cost should be lower (i.e., better). How does the value of $|\mathbf{w}|^2$ using Method 3 compare to its value using Method 2?

Put your solution in a Python file called `homework2_WPIUSERNAME1.py`
(or `homework2_WPIUSERNAME1_WPIUSERNAME2.py` for teams).