Homework 2 – Deep Learning (CS/DS 541, Whitehill, Spring 2019)

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

1. **XOR problem** [10 points, on paper]: 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 $b = 0.5$.

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 $f_{\mathbf{w}} : \mathbb{R}^{576} \rightarrow \mathbb{R}$ 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** [10 points, in Python]: Compute the parameters $\mathbf{w} = (w_1, \ldots, w_{576})$ representing the weights 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

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

   where α is a hyperparameter specifying the L_2 regularization strength, $\hat{y} = f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$, n is the number of examples in the training set $\mathcal{D}_{\text{tr}} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$, each $\mathbf{x}^{(i)} \in \mathbb{R}^{576}$, and each $y^{(i)} \in \{0, 1\}$. Note that the one-shot solution to *regularized* linear regression is slightly different than for unregularized linear regression (which we derived in class); you will need to derive the solution yourself. After optimizing $\mathbf{w}$ only on the **training set**, compute and report the *unregularized* cost J on the training set $\mathcal{D}_{\text{tr}}$ and (separately) on the testing set $\mathcal{D}_{\text{te}}$. Show that there an exists a positive value for α such that the training cost is higher but the testing cost is lower, compared to using $\alpha = 0$.

   (b) **Method 2 – gradient descent** [10 points, in Python]: Let the initial value for $\mathbf{w} \in \mathbb{R}^{576}$ be the $\mathbf{0}$ vector. Choose 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 unregularized 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.

3. **Weighted L_2-regularized linear regression** [10 points, on paper]: Modify the cost function in Equation 1 so that:

   - The weight vector $\mathbf{w}$ contains an extra component at the end that represents that bias term. (See slide 47 in `Class2.pdf` on Canvas).
   - J includes an L_2 penalty term that penalizes each component of $\mathbf{w}$ equally *except* the last element (corresponding to b), that receives 0 penalization. (When training neural networks, one typically regularizes the bias term less than the other weights – and often not at all.)
   - It attributes a *weight* to each training example. These weights have nothing to do with the parameters that we are trying to optimize; rather, they specify how much to "value" each example in the training set. This can be useful, for example, to compensate for one class (e.g., "smile") being much rarer than another.

Then, derive the one-shot solution and gradient expressions (for gradient descent).

4. **Regularization to encourage symmetry** [10 points, on paper]: Faces tend to be left-right symmetric. How can you use L_2 regularization to discourage the weights from becoming too *a*symmetric? For simplicity, consider the case of a tiny 1×2 "image". Hint: instead of using $\frac{\alpha}{2}\mathbf{w}^\top\mathbf{w} = \frac{\alpha}{2}\mathbf{w}^\top\mathbf{I}\mathbf{w}$ as the L_2 penalty term, consider a different matrix in the middle. Your answer should consist of a 2×2 matrix $\mathbf{S}$ as well as an explanation of why it works.

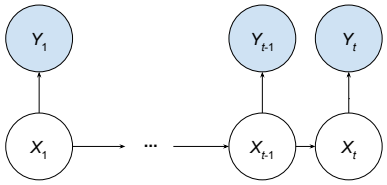
5. **Recursive state estimation in Hidden Markov Models** [10 points, on paper]: Teachers try to monitor their student's knowledge of the subject-matter, but teachers cannot directly peer inside students' brains. Hence, they must make *inferences* about what the student knows based on students' *observable behavior*, i.e., how they perform on tests, their facial expressions during class, etc. Let random variable (RV) X_t represent the student's *state*, and let RV Y_t represent the student's observable behavior, at time t. We can model the student as a Hidden Markov Model (HMM):

   (a) X_t depends *only* on the previous state X_{t-1}, *not* on any states prior to that (*Markov* property), i.e.
   $$P(x_t \mid x_1, \ldots, x_{t-1}) = P(x_t \mid x_{t-1})$$

   (b) The student's behavior Y_t depends only on his/her current state X_t, i.e.:
   $$P(y_t \mid x_t, y_1, \ldots, y_{t-1}) = P(y_t \mid x_t)$$

   (c) X_t cannot be observed directly (it is *hidden*).

   A probabilistic graphical model for the HMM is shown below, where only the observed RVs are shaded (the latent ones are transparent):



   Suppose that the teacher already knows:

   - $P(y_t \mid x_t)$ (*observation likelihood*), i.e., the probability distribution of the student's behaviors given the student's state.
   - $P(x_t \mid x_{t-1})$ (*transition dynamics*), i.e., the probability distribution of the student's current state given the student's previous state.

   The goal of the teacher is to estimate the student's current state X_t given the *entire* history of observations $Y_1, \ldots, Y_t$ he/she has made so far. Show that the teacher can, at each time t, update his/her belief *recursively*:

   $$P(x_t \mid y_1, \ldots, y_t) \propto P(y_t \mid x_t) \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1} \mid y_1, \ldots, y_{t-1})$$

   where $P(x_{t-1} \mid y_1, \ldots, y_{t-1})$ is the teacher's belief of the student's state from time $t-1$, and the summation is over every possible value of the previous state x_{t-1}. **Hint**: You will need to use Bayes' rule, i.e., for any RVs A, B, and C:

   $$P(a \mid b, c) = \frac{P(b \mid a, c) P(a \mid c)}{P(b \mid c)}$$

However, since the denominator in the right-hand side does not depend on a, this can also be rewritten as:

$$P(a \mid b, c) \propto P(b \mid a, c)P(a \mid c)$$

Put your code in a Python file called `homework2_WPIUSERNAME1.py`
(or `homework2_WPIUSERNAME1_WPIUSERNAME2.py` for teams). For the proofs, please create a PDF called `homework2_WPIUSERNAME1.pdf`
(or `homework2_WPIUSERNAME1_WPIUSERNAME2.pdf` for teams). Create a Zip file containing both your Python and PDF files, and then submit on Canvas.