Homework 3 – Deep Learning (CS/DS 541, Whitehill, Spring 2019)

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 cost function

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

Newton's method (see Equation 4.12 in *Deep Learning*) will converge to the optimal solution $\mathbf{w}^* = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}$ in 1 iteration no matter what the starting point $\mathbf{w}_0$ of the search is.

2. **Derivation of softmax regression gradient updates** [20 points]: As explained in class, let

$$\mathbf{W} = \left[\begin{array}{ccc} \mathbf{w}^{(1)} & \dots & \mathbf{w}^{(c)} \end{array} \right]$$

be an $m \times c$ matrix containing the weight vectors from the c different classes. The output of the softmax regression neural network is a vector with c dimensions such that:

$$\begin{aligned} \hat{y}_k &= \frac{\exp z_k}{\sum_{k'=1}^{c} \exp z_{k'}} \qquad (1) \\ z_k &= \mathbf{x}^\top \mathbf{w}_k \end{aligned}$$

for each $k = 1, \dots, c$. Correspondingly, our cost function will sum over all c classes:

$$f_{\mathrm{CE}}(\mathbf{W}) = -\frac{1}{2n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \log \hat{y}_k^{(i)}$$

Important note: When deriving the gradient expression for each weight vector $\mathbf{w}_l$, it is crucial to keep in mind that the weight vector for each class $l \in \{1, \dots, c\}$ affects the outputs of the network for *every* class, *not* just for class l. This is due to the normalization in Equation 1 – if changing the weight vector *increases* the value of $\hat{y}_l$, then it necessarily must *decrease* the values of the other $\hat{y}_{l' \neq l}$.

In this homework problem, please complete the following derivation that is outlined below:

Derivation: For each weight vector $\mathbf{w}_l$, we can derive the gradient expression as:

$$\begin{aligned} \nabla_{\mathbf{w}_l} f_{\mathrm{CE}}(\mathbf{W}) &= -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \nabla_{\mathbf{w}_l} \log \hat{y}_k^{(i)} \\ &= -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \left(\frac{\nabla_{\mathbf{w}_l} \hat{y}_k^{(i)}}{\hat{y}_k^{(i)}} \right) \end{aligned}$$

We handle the two cases $l = k$ and $l \neq k$ separately. For $l = k$:

$$\begin{aligned} \nabla_{\mathbf{w}_l} \hat{y}_k^{(i)} &= \text{complete me...} \\ &= \mathbf{x}^{(i)} \hat{y}_l^{(i)} (1 - \hat{y}_l^{(i)}) \end{aligned}$$

For $l \neq k$:

$$\begin{aligned} \nabla_{\mathbf{w}_l} \hat{y}_k^{(i)} &= \text{complete me...} \\ &= -\mathbf{x}^{(i)} \hat{y}_k^{(i)} \hat{y}_l^{(i)} \end{aligned}$$

To compute the total gradient of f_{CE} w.r.t. each $\mathbf{w}_k$, we have to sum over all examples *and* over $l = 1, \ldots, c$. (**Hint**: $\sum_k a_k = a_l + \sum_{k \neq l} a_k$.)

$$\begin{aligned}
\nabla_{\mathbf{w}_l} f_{\text{CE}}(\mathbf{W}) &= -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \nabla_{\mathbf{w}_l} \log \hat{y}_k^{(i)} \\
&= \text{complete me...} \\
&= -\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(y_l^{(i)} - \hat{y}_l^{(i)} \right)
\end{aligned}$$

3. **Implementation of softmax regression** [25 points]: In this problem you will train a 2-layer neural network to classify images of hand-written digits from the MNIST dataset. The input to the network will be a 28×28-pixel image (converted into a 784-dimensional vector); the output will be a vector of 10 probabilities (one for each digit). Specifically, the network you create should implement a function $f : \mathbb{R}^{784} \to \mathbb{R}^{10}$, where the kth component of $f(x)$ (i.e., the probability that input $\mathbf{x}$ belongs to class k) is given by

$$\frac{\exp(\mathbf{x}^\top \mathbf{w}_k)}{\sum_{k'=1}^{10} \exp(\mathbf{x}^\top \mathbf{w}_{k'})}$$

The cross-entropy loss function should be

$$f_{\text{CE}}(\mathbf{w}_1, \ldots, \mathbf{w}_{10}) = -\frac{1}{2n} \sum_{i=1}^{n} \sum_{k=1}^{10} y_k^{(i)} \log \hat{y}_k^{(i)} + \frac{\alpha}{2} \sum_{k=1}^{c} \mathbf{w}_k^\top \mathbf{w}_k$$

where n is the number of examples and α is a regularization constant.. Note that each $\hat{y}_k$ implicitly depends on all the weights $\mathbf{w}_1, \ldots, \mathbf{w}_{10}$.

To get started, first download the MNIST dataset (including both the training, validation, and testing subsets) from the following web links:

- `https://s3.amazonaws.com/jrwprojects/mnist_train_images.npy`
- `https://s3.amazonaws.com/jrwprojects/mnist_train_labels.npy`
- `https://s3.amazonaws.com/jrwprojects/mnist_validation_images.npy`
- `https://s3.amazonaws.com/jrwprojects/mnist_validation_labels.npy`
- `https://s3.amazonaws.com/jrwprojects/mnist_test_images.npy`
- `https://s3.amazonaws.com/jrwprojects/mnist_test_labels.npy`

These files can be loaded into `numpy` using `np.load`.

Then implement stochastic gradient descent (SGD) to minimize the cross-entropy loss function.

Hyperparameter tuning: In this problem, there are several different hyperparameters that will impact the network's performance:

- Mini-batch size $\tilde{n}$.
- Learning rate ϵ and number of gradient descent iterations T; *or* the learning rate decay rate γ and number of iterations per decay K. (You can choose which strategy you prefer.)
- Regularization strength α.

In order not to cheat (in the machine learning sense) – and thus overestimate the performance of the network – it is crucial to optimize the hyperparameters **only** on the validation set. (The training set would also be acceptable but typically leads to worse performance.)

Performance evaluation: Once you have tuned the hyperparameters and optimized the weights so as to maximize performance on the validation set, then: (1) **stop** training the network and (2) evaluate the network on the **test** set. Record the performance both in terms of (unregularized) cross-entropy loss (smaller is better) and percent correctly classified examples (larger is better).

Put your code in a Python file called `homework3_WPIUSERNAME1.py` (or `homework3_WPIUSERNAME1_WPIUSERNAME3.py` for teams). For the proof and derivation, please create a PDF called `homework3_WPIUSERNAME1.pdf` (or `homework3_WPIUSERNAME1_WPIUSERNAME3.pdf` for teams). Create a Zip file containing both your Python and PDF files, and then submit on Canvas.