Homework 2 – Deep Learning (CS/DS541, Whitehill, Fall 2024)

Collaboration policy: You may complete this assignment with a partner, if you choose. In that case, both partners should sign up on Canvas as a "team". You can do this by clicking: Canvas $\rightarrow$ People (on the left side of the page) $\rightarrow$ Group (at the top left of the page) $\rightarrow$ Choose a group and join (e.g., Homework2 10). Only one of you should submit the assignment.

You are not permitted to use ChatGPT on any part of this assignment.

1. **A linear NN will never solve the XOR problem [10 points, on paper]**: Read the description of the XOR problem from Section 6.1 of the *Deep Learning* textbook: `https://www.deeplearningbook.org/contents/mlp.html`. Assume that you wish to classify the 4 points in this dataset using a 2-layer linear NN (i.e., the same model as in Homework 1, Problem 2). Then show (by deriving the gradient, setting to 0, and solving mathematically, not in Python) that the values for $\mathbf{w} = [w_1, w_2]^\top$ and b that minimize the function $f_{\text{MSE}}(\mathbf{w}, b)$ in Equation 6.1 are: $w_1 = 0$, $w_2 = 0$, and $b = 0.5$ – in other words, the best prediction line is simply flat and always guesses $\hat{y} = 0.5$.

2. **Derivation of softmax regression gradient updates [20 points, on paper]**: 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:

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

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

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

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:

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

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 $l \neq k$:

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

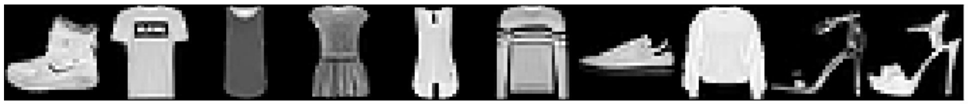
To compute the total gradient of f_{CE} w.r.t. each $\mathbf{w}^{(k)}$, we have to sum over all examples *and* over $l = 1, \ldots, c$. (**Hint**: $\sum_k a_k = a_l + \sum_{k \neq l} a_k$. Also, $\sum_k y_k = 1$.)

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

Finally, show that

$$\nabla_{\mathbf{b}} f_{\text{CE}}(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)} \right)$$

3. **Implementation of softmax regression [25 points, in Python code]**:



Train a 2-layer softmax neural network to classify images of fashion items (10 different classes, such as shoes, t-shirts, dresses, etc.) from the Fashion 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 class). The cross-entropy loss function[1] that you minimize should be

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

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 $\mathbf{W} = \left[\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(10)}\right]$ and biases $\mathbf{b} = \left[b^{(1)}, \ldots, b^{(10)}\right]$.

To get started, first download the Fashion MNIST dataset from the following web links:

- `https://s3.amazonaws.com/jrwprojects/fashion_mnist_train_images.npy`
- `https://s3.amazonaws.com/jrwprojects/fashion_mnist_train_labels.npy`
- `https://s3.amazonaws.com/jrwprojects/fashion_mnist_test_images.npy`
- `https://s3.amazonaws.com/jrwprojects/fashion_mnist_test_labels.npy`

These files can be loaded into `numpy` using `np.load`. Each "labels" file consists of a 1-d array containing n labels (valued 0-9), and each "images" file contains a 2-d array of size $n \times 784$, where n is the number of images.

Next, implement stochastic gradient descent (SGD) to minimize the cross-entropy loss function on this dataset. Regularize the weights but *not* the biases. Optimize the same hyperparameters as in

[1]In this equation, the regularization term is not divided by n like in the lecture notes. Either equation is valid since the $1/n$ can be subsumed into α. Here, for simplicity, the $1/n$ is omitted.

homework 1 problem 2 (age regression). You should also use the same methodology as for the previous homework, including the splitting of the training files into validation and training portions.

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 this information into the PDF you submit.

Hint 1: it accelerates training if you first normalize all the pixel values of both the training and testing data by dividing each pixel by 255. **Hint 2**: when using functions like `np.sum` and `np.mean`, make sure you know what the `axis` and `keepdims` parameters mean and that you use them in a way that is consistent with the math!

4. **Logistic Regression [15 points, on paper]**: Consider a 2-layer neural network that computes the function

$$\hat{y} = \sigma(\mathbf{x}^\top \mathbf{w} + b)$$

where $\mathbf{x}$ is an example, $\mathbf{w}$ is a vector of weights, b is a bias term, and σ is the logistic sigmoid function. Assume we train this network using the log loss, as described in class. Moreover, suppose **all the training examples are positive**. Answer the following questions about convergence. (Informally, a sequence of numbers *converges* if it gets closer and closer to a specific number as the sequence progresses. A sequence that does *not* converge can do different things, e.g., change erratically, or grow towards $+/-\infty$.) While you are not required to give formal proofs, you should explain your reasoning, which could either be a mathematical argument or a simulation result. Put your answers into your PDF file.

   (a) Given a well-chosen learning rate: what value will the training loss converge to during gradient descent?

   (b) Given a well-chosen learning rate: will b always converge; does convergence depend on the exact training examples; or does it never converge?

   (c) Suppose the training set contains exactly 2 examples, $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in \mathbb{R}^2$. Give specific *non-zero* values for these training data such that:

      i. $\mathbf{w}$ will converge during gradient descent (given a well-chosen learning rate).

      ii. $\mathbf{w}$ will *not* converge during gradient descent (no matter what the learning rate).

Create a Zip file containing both your Python and PDF files, and then submit on Canvas. If you are working as part of a group, then only **one** member of your group should submit (but make sure you have already signed up in a pre-allocated team for the homework on Canvas). Please name your submission files using your names in the following format: (<student1's first name_student1's last name>_<student2's first name_student2's last name>). For example: jacob_whitehill_yao_su.zip