CSE 151B Deep Learning
UCSD, Spring 2025

Homework 1

Due 11:59 p.m, April 22, 2025

Please submit your written solutions as a single PDF file using the provided LaTeX template on the course website. (`https://sites.google.com/view/cse151b-251b`). You can use Overleaf (`https://www.overleaf.com/`) to compile LaTex files online. For problems requiring code, additionally submit all necessary code in one zip file. Code must run and produce the results reported in the PDF for full credit.

1 Supervised Learning [4 Points]

Problem A [2 points]: Feature Representation
Suppose you have a set of documents and you want to know which document is most useful for this homework:

- **Doc1**: "Supervised learning uses labeled data to train models. The model is learning from data."
- **Doc2**: "Unsupervised learning finds hidden patterns in unlabeled data. No labels are given to the model."
- **Doc3**: "Reinforcement learning optimizes actions based on rewards. The agent is learning from interactions with the environment."
- **Doc4**: "Deep learning is a subfield of machine learning. Deep learning models are used for complex data."
- **Doc5**: "Model evaluation involves testing the model on unseen data. Evaluation metrics such as accuracy and F1-score are important."

Convert each sentence to a bag-of-words vector using the dictionary [*learning*, *data*, *model*, *evaluation*, *deep*]. Describe the feature vector and write out the matrix representing all of the documents.

Problem B [2 points]: Logistic Regression
Logistic regression is a binary classification model. Intuitively, logistic regression can be

conceptualized as a single neuron reading in a d-dimensional feature vector $x \in \mathbb{R}^d$ and producing an output $f(x) \in [0, 1]$ which is the predicted probability that output is correct. The "neuron" is parameterized by a weight vector $w \in \mathbb{R}^{d+1}$, where $w_0 = b$ represents the bias term and the input is augmented with $x_0 = 1$.

In logistic regression, the model class is:

$$f(x) = \sigma(w^\top x), \quad \sigma(x) = \frac{1}{1 + \exp(-x)} \tag{1}$$

The loss function is *cross entropy*, defined as

$$L(y, f(x)) = -\sum_{i=1}^{N} \left\{ y^{(i)} \log(f(x^{(i)})) + (1 - y^{(i)}) \log(1 - f(x^{(i)})) \right\} \tag{2}$$

where $y^{(i)} \in \{0, 1\}$ is the binary label for the data sample i.

In order to learn the parameters w, we will use gradient descent. Prove that the gradient of the loss w.r.t. the jth entry of the vector w is: $\frac{\partial L}{\partial w_j} = \sum_{i=1}^{N} (f(x^{(i)}) - y^{(i)}) x_j^{(i)}$.

2 Multi-Layer Perceptron[6 Points]

Problem A [3 points]: Perceptron
The perceptron is a simple linear model used for binary classification. For an input vector $\mathbf{x} \in \mathbb{R}^d$, weights $\mathbf{w} \in \mathbb{R}^d$, and bias $b \in \mathbb{R}$, a perceptron $f : \mathbb{R}^d \to \{-1, 1\}$ takes the form

$$f(\mathbf{x}) = \text{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

The weights and bias of a perceptron can be thought of as defining a hyperplane that divides $\mathbb{R}^d$ such that each side represents an output class. For example, for a two dimensional dataset, a perceptron could be drawn as a line that separates all points of class $+1$ from all points of class -1.

The PLA (or the Perceptron Learning Algorithm) is a simple method of training a perceptron. First, an initial guess is made for the weight vector $\mathbf{w}$. Then, one misclassified point is chosen arbitrarily and the $\mathbf{w}$ vector is updated by

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y(t)\mathbf{x}(t)$$
$$b_{t+1} = b_t + y(t),$$

where $\mathbf{x}(t)$ and $y(t)$ correspond to the misclassified point selected at the t^{th} iteration. This process continues until all points are classified correctly. Download the source file and work with the provided Jupyter notebook, titled `HW1_notebook.ipynb`. This notebook utilizes the file `perceptron_helper.py`, but no modification is needed.

The graph below shows an example 2D dataset. The $+$ points are in the $+1$ class and the $\circ$ point is in the -1 class.

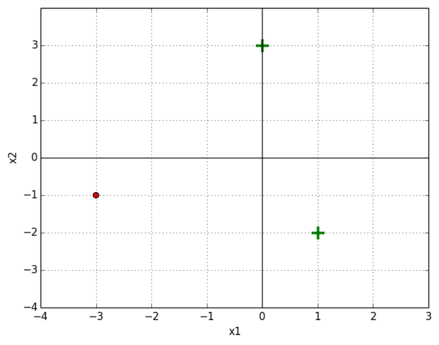

Figure 1: The green $+$ are positive and the red $\circ$ is negative

i. [1 points]: Implementation of Perceptron
Implement the `update_perceptron` and `run_perceptron` methods in the notebook, and perform the perceptron algorithm with initial weights $w_1 = 0, w_2 = 1, b = 0$.

A dataset $S = \{(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(N)}, y^{(N)})\} \subset \mathbb{R}^d \times \mathbb{R}$ is *linearly separable* if there exists a perceptron that correctly classifies all data points in the set. In other words, there exists a hyperplane that separates positive data points and negative data points.

ii. [1 points]: Linear Separability
In a 2D dataset, how many data points are in the smallest dataset that is not linearly separable, such that no three points are collinear? How about for a 3D dataset such that no four points are coplanar? Please limit your solution to a few lines - you should justify but not prove your answer.

Finally, how does this generalize for an N-dimensional set, in which **no** $< N$-dimensional hyperplane contains a non-linearly-separable subset? For the N-dimensional case, you may state your answer without proof or justification.

iii. [1 points]: Non-linearly Separable Dataset
Run the visualization code in the Jupyter notebook corresponding to question iii. Assume a dataset is *not* linearly separable. Will the Perceptron Learning Algorithm ever converge? Why or why not?

Problem B [3 points]: Multilayer Perceptron

In this problem, you will build a 2-layer Multilayer Perceptron(MLP) for MNIST digit classification. You can read more about this task on the Wikipedia page. The MNIST dataset is available at Torchvision.

i. [2 points]: MNIST Classification Implementation

Work with the provided Jupyter notebook, titled `HW1_notebook.ipynb`. Feel free to play around with the model architecture and see how the training time/performance changes, but to begin, try the following: Image (784 dimensions) $\rightarrow$ fully connected layer (500 hidden units) $\rightarrow$ nonlinearity (ReLU)$\rightarrow$ fully connected (10 hidden units)$\rightarrow$ softmax. For code submission, this should be the architecture for your model.

Try building the model both with basic PyTorch operations, and then again with more object-oriented higher-level APIs. You should get similar results!

Some hints:

- Even as we add additional layers, we still only require a single optimizer to learn the parameters. Just make sure to pass all parameters to it!
- This MLP model has many more parameters than the logistic regression example, which makes it more challenging to learn. To get the best performance, you may want to play with the learning rate and increase the number of training epochs.
- Be careful using `torch.nn.CrossEntropyLoss()`. It combines the softmax operation with the cross-entropy. This means you need to pass in the logits (predictions pre-softmax) to this loss. Computing the softmax separately and feeding the result into `torch.nn.CrossEntropyLoss()` will significantly degrade your model's performance!

ii. [1 points]: Parameter Counts

How many trainable parameters does your model have? Pls write out your reasoning process. How does this compare to the logistic regression example?