CSE 151B Deep Learning
UCSD, Spring 2021

Homework 1

Due 11:59 p.m, April 17, 2022

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). 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

Problem A [2 points]: Feature Representation
Suppose you have a set of Git commit messages and you want to know which messages indicate a bugfix:

1. This was a bug, and a fix to correct it.
2. Correct an error message that was wrong.
3. OOPS all the code was broken but we made it work again.
4. This is a feature, not a bug!

Convert each sentence to a bag-of-words vector using the dictionary [*bug*, *fix*, *correct*, *error*, *wrong*]. Describe the feature vector and write out the matrix representing all of the commit messages.

Problem B [3 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

Problem A [4 points]: Function Approximation

i. [2 points]: Draw or describe (use `includegraphics` command in latex) a fully-connected network with ReLU units that implements the OR function on two Boolean inputs, x_1 and x_2. Your networks should contain the minimum number of hidden units possible. The OR function OR(x_1, x_2) is defined as:

$$\begin{aligned} \text{OR}(1,0) &\geq 1 \\ \text{OR}(0,1) &\geq 1 \\ \text{OR}(1,1) &\geq 1 \\ \text{OR}(0,0) &= 0 \end{aligned}$$

Your network need only produce the correct output when $x_1 \in \{0, 1\}$ and $x_2 \in \{0, 1\}$ (as described in the examples above).

ii. [2 points]: What is the minimum number of fully-connected layers (with ReLU units) needed to implement an XOR of two Boolean inputs x_1, x_2? Recall that the XOR function is defined as:

$$\begin{aligned} \text{XOR}(1,0) &\geq 1 \\ \text{XOR}(0,1) &\geq 1 \\ \text{XOR}(0,0) = \text{XOR}(1,1) &= 0 \end{aligned}$$

For the purposes of this problem, we say that a network f computes the XOR function if $f(x_1, x_2) = \text{XOR}(x_1, x_2)$ when $x_1 \in \{0, 1\}$ and $x_2 \in \{0, 1\}$ (as in the examples above).

Explain why a neural network with fewer layers than the number you specified above cannot compute XOR.

Problem B [6 points]: Perceptron Implementation
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

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

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

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

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.

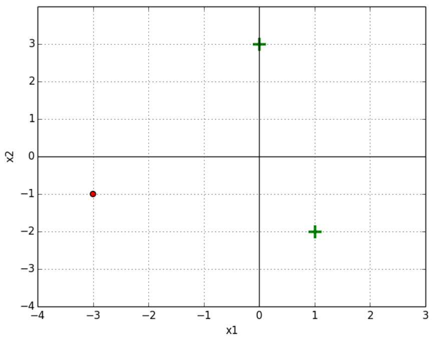

Figure 1: The green $+$ are positive and the red $\circ$ is negative

i. [3 points]: Implementation of Perceptron

Implement the `update_perceptron` and `run_perceptron` methods in the notebook, and perform the perceptron algorithm with initial weights $w_1 = 0, w_2 = 1, b = 0$. Give your solution in the form a table showing the weights and bias at each timestep and the misclassified point $([x_1, x_2], y)$ that is chosen for the next iteration's update. You can iterate through the three points in any order. Your code should output the values in the table below; cross-check your answer with the table to confirm that your perceptron code is operating correctly.

t	b	w_1	w_2	x_1	x_2	y
0	0	0	1	1	-2	+1
1	1	1	-1	0	3	+1
2	2	1	2	1	-2	+1
3	3	2	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. [3 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.