EE 240: Pattern Recognition and Machine Learning
Homework 3

Due date: May 20, 2023

Description: Neural networks and deep learning.

Reading assignment and references: AML Book Ch. 7, UFLDL notes, and Michael Nielson book

Homework and lab assignment submission policy:
All homework and lab assignments must be submitted online via https://eLearn.ucr.edu.
Submit your homeworks as a single Python notebook that is free of any typos or errors. Talk to your TA to make sure that the Python version match.
Homework solutions should be written and submitted individually, but discussions among students are encouraged.
All assignments should be submitted by the due date. You will incur 25% penalty for every late day.

H3.1 Least-squares solution via gradient computation for a convolutional system:

(a) **Linear system:** Let us consider a linear model $\mathbf{y} = W\mathbf{x} + b$, where W is an $m \times d$ matrix and $b \in \mathbb{R}$. Write an expression for the optimal values of W, b that minimize the following cost function:

$$J(W, b) = \frac{1}{2}\|\mathbf{y} - W\mathbf{x} - b\|^2.$$

Remember the optimality condition that the gradient of the cost function w.r.t. the optimization variables should be zero at their optimal values. Therefore, first derive expressions for $\frac{\partial J}{\partial W_{ij}}, \frac{\partial J}{\partial b}$, where W_{ij} denotes i, j entry in matrix W. Then set the gradients to zero to derive the expressions for the optimal solution. **(0 pts)**

(b) **Convolutional system:** Let us consider the following linear model $\mathbf{y} = W\mathbf{x} + b$, where W is a Toeplitz matrix corresponding to a filter $\mathbf{w}$ of length k. We can write the $d + k - 1 \times d$ Toeplitz matrix as **(10 pts)**

$$W = \begin{bmatrix} \mathbf{w}_1 & 0 & 0 & \dots & 0 & 0 \\ \mathbf{w}_2 & \mathbf{w}_1 & 0 & \dots & 0 & 0 \\ \mathbf{w}_3 & \mathbf{w}_2 & \mathbf{w}_1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{w}_k & \mathbf{w}_{k-1} & \mathbf{w}_{k-2} & \dots & \dots & \dots \\ 0 & \mathbf{w}_k & \mathbf{w}_{k-1} & \dots & \dots & \dots \\ 0 & 0 & \mathbf{w}_k & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \mathbf{w}_k & \mathbf{w}_{k-1} \\ 0 & 0 & 0 & \dots & 0 & \mathbf{w}_k \end{bmatrix}.$$

We can also define the linear model above as $\mathbf{y} = \mathbf{w} * \mathbf{x} + b$, where $\mathbf{x} * \mathbf{w}$ denotes the convolution of two vectors that can be defined as

$$(\mathbf{x} * \mathbf{w})_j = \sum_{i=1}^{d} \mathbf{x}_i \mathbf{w}_{j-i+1},$$

where $\mathbf{w}_i$ denotes ith entry in vector $\mathbf{w}$ and $\mathbf{w}_i = 0$ for $i < 1$ and $i > k$. With our definition above, we assume that j is only valid for $1 \leq j \leq d + k - 1$.

Derive an expression for the optimal values of $\mathbf{w}, b$ that minimize the following cost function:

$$J(W, b) = \frac{1}{2}\|\mathbf{y} - \mathbf{w} * \mathbf{x} - b\|^2. \tag{1}$$

Also, derive expressions for $\dfrac{\partial J}{\partial \mathbf{w}_i}$ and $\dfrac{\partial J}{\partial b}$.

(c) **Deconvolution:** In this question, you will estimate a filter $\mathbf{w}$ by minimizing the cost function in (1) via gradient descent using an input-output pair $(\mathbf{x}, \mathbf{y})$. **(10 pts)**

i. $\mathbf{y} = [1, 1, 1, 1, 1, 1, 1, 1, 1, -9]$ and $\mathbf{x} = [1, 2, \ldots, 9]$ is a vector of length 9. The filter $\mathbf{w}$ is of length 2. Compute $\mathbf{w} \in \mathbb{R}^2$ and $b \in \mathbb{R}$ using gradient descent.

ii. $\mathbf{y} = [1.5, 2, 3, 4, 5, 6, 7, 8, 9, 5, 5.5]$ and $\mathbf{x} = [1, 2, \ldots, 9]$ is a vector of length 9. The filter $\mathbf{w}$ is of length 3. Compute $\mathbf{w} \in \mathbb{R}^3$ and $b \in \mathbb{R}$ using gradient descent.

H3.2 In this question, we will learn how to implement backpropagation (or backprop) for "vanilla" neural networks (or Multi-Layer Perceptrons). **(60 pts)**

You will begin by writing the forward and backward passes for different types of layers and then go on to train a two-layer network on the CIFAR-10 dataset in Python.

This question is adopted from Stanford cs231n course. Please familiarize yourself with the workflow recommended here. It will make your life very easy.

Download the zip file assignment1_colab.zip. Extract the files and upload to a google drive and use colab. Please go through the notebooks even though most of the coding is already done. It will give you an idea how neural networks work.

We will only look at **Two-layer Neural Network** in `two_layer_net.ipynb`.

The notebook `two_layer_net.ipynb` will walk you through implementing a two-layer neural network on CIFAR-10. You will implement the forward/backward pass, test different loss functions, and tune its hyperparameters.

The structure of the two layer fully connected network is: input $\rightarrow$ fully-connected-layer $\rightarrow$ ReLU activation $\rightarrow$ fully-connected-layer $\rightarrow$ softmax. If x is input to the network as a row vector then the forward propagation calculates a prediction vector $\hat{y}$ as follows:

$$h_1 = \text{ReLU}(xW_1 + b_1)$$
$$\hat{y} = \text{softmax}(h_1 W_2 + b_2)$$

where h_1 is the intermediate output in the hidden layer, W_1, b_1 and W_2, b_2 are first and second layer weights and biases respectively. Note that these definitions are "trasnposed" version of the notations we used in our lecture. For a vector of length d, W_1 will be a $d \times K_1$, where d is the size of input vector and K_1 is the output size of the first layer.

Most of the coding has been already done but you will have to write the codes for forward/backward pass, loss calculation and gradients in `cs231n/layers.py`. Finally you need to write the code to tune the hyperparameters for best performance of this neural network. There is a block in `two_layer_net.ipynb` for this tuning.

Deliverables:

Zip the completed iPython notebooks and relevant files (except the CIFAR10 dataset). Submit the generated zip file for H3.2.

You will easily find the solution for this exercise online. Do not use that, do not look at that. If you copy your solution from any online source, you will get ZERO for this assignment.

H3.3 In this question you will solve the XOR problem using 2-layer neural network using **ReLu** nonlinearities. Remember that in the class, we used **step/sign** nonlinearities. **(20 pts)**

Suppose you are given four training samples from $\mathbb{R}^2$ distributed in an XOR pattern (i.e., points on opposite diagonals of a rectangle have same labels, as shown in the table and figure below.)

y	$\mathbf{x}(1)$	$\mathbf{x}(2)$
1	-1	-1
0	-1	1
0	1	-1
1	1	1

Table 1: Table for class labels and sample values in an XOR pattern

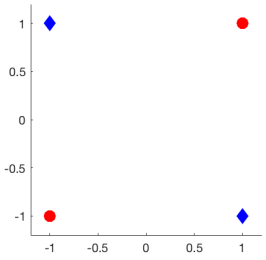


Figure 1: Blue diamonds and red circles denote samples for two classes (only color version)

We want to use a 2-layer neural network shown in Figure 2 to classify this data using ReLu nonlinearity $\varphi(z) = \max(0, z)$.
We use same notations we discussed in class, where superscripts ^{1,2} indicate layer number (not the power).

$$\mathbf{z}^1 = \begin{bmatrix} z_1^1 \\ z_2^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix} = \mathbf{W}^1\mathbf{x} + \mathbf{b}^1,$$

$$\mathbf{y}^1 = \varphi(\mathbf{z}^1)$$

$$\mathbf{z}^2 = [w_{11}^2 \quad w_{12}^2] \begin{bmatrix} y_1^1 \\ y_2^1 \end{bmatrix} + \mathbf{b}^2 = \mathbf{W}^2\mathbf{y}^1 + \mathbf{b}^2,$$

$$\mathbf{y}^2 = \varphi(\mathbf{z}^2).$$

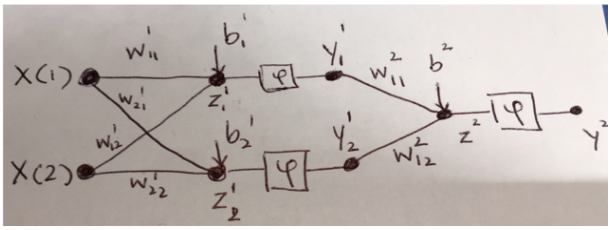


Figure 2: Two-layer neural network.

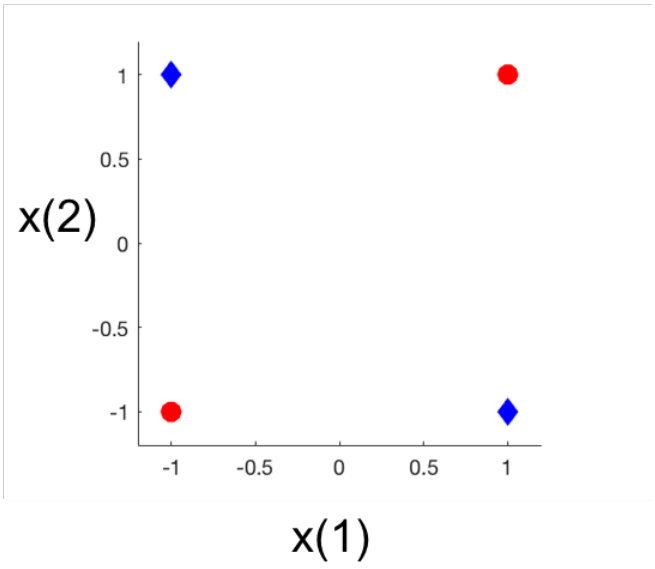
Our goal is to find some values for weights $\mathbf{W}^1, \mathbf{W}^2, \mathbf{b}^1, \mathbf{b}^2$ such that the data can be classified correctly. You can pick any values that can correctly classify data using ReLu nonlinearities: $\varphi(z) = \max(0, z)$.

(a) Write down the values of $\mathbf{W}^1, \mathbf{W}^2, \mathbf{b}^1, \mathbf{b}^2$ that you think will correctly classify data.

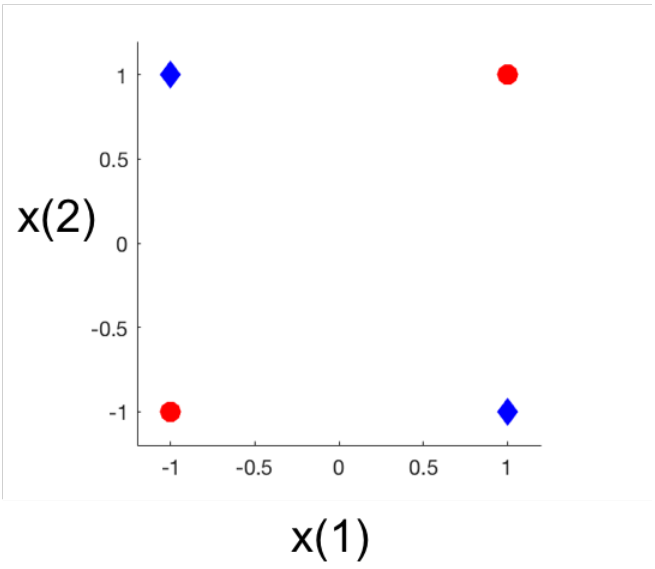
(b) Plot a decision boundary for the first node in layer 1; that is, $w_{11}^1 x(1) + w_{12}^1 x(2) + b_1^1 = 0$ in (b) below.

(c) Plot a decision boundary for the second node in layer 1; that is, $w_{21}^1 x(1) + w_{22}^1 x(2) + b_2^1 = 0$ in (c) below.

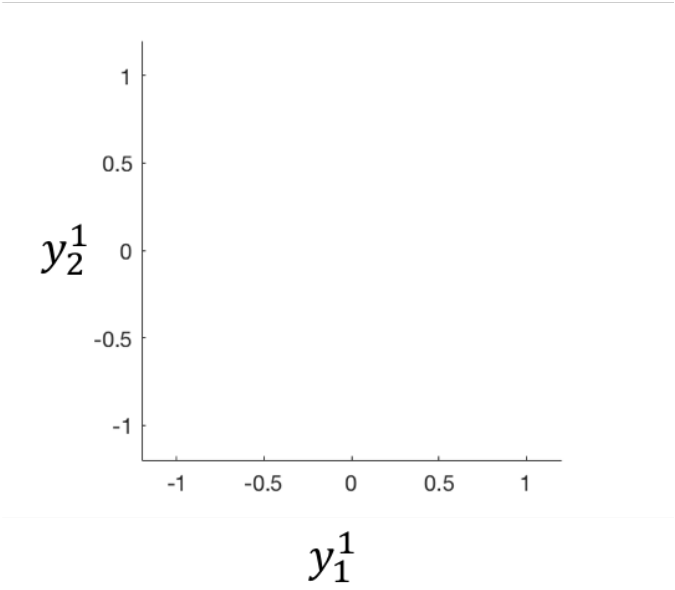
(d) Plot a decision boundary for the output node in layer 2; that is, $w_{11}^2 y_1^1 + w_{12}^2 y_2^1 + b_1^2 = 0$ in (d) below.



(b) Plot a decision boundary for node 1, layer 1.



(c) Plot a decision boundary for node 2, layer 1.



(d) Plot y_1^1, y_2^1 for all four points and a decision boundary for layer 2 output.

Maximum points: 100