Assignment 2: Single-Neuron Classifier Coded from Scratch *and* using PyTorch

Deadline: Thursday October 1, 2020 at 9:00pm

Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted.

The goal of this assignment is to build and understand the basic process of how a single artificial neuron can be trained and used as a classifier. You will build the software in two ways: first, from scratch with almost no library support, and second using the PyTorch library. You'll use one set of data to train the linear neuron and a second set to 'validate' the model produced in training. The training will make use of the gradient descent method of optimization, as described in class. Note that although the actual classification task is very simple to do in normal procedural software; this simplicity will help you gain insight and understanding of the optimization process that is used in all deep learning.

What To Submit

You should hand in the following files to this assignment on Quercus:

- A PDF file `assign2.pdf` containing your answers to the written questions in this assignment; please make it clear which sections contain the questions that you are answering.
- Your code, in the file `part1.ipynb` that must be able to generate all of the data required in Section 6.
- Your code, in the file `part2.ipynb` that must be able to generate all of the data described in Section 7.

1 Problem To Solve

The goal of this assignment is to build a single-neuron classifier that solves the following problem: given a 3x3 array of binary data (containing only 1's and 0's), determine when the 3x3 pattern is an 'X' as illustrated in Figure 1. To be clear, the goal is to make a classifier that outputs the value `1` when the input is the pattern in Figure 1, and the value `0` when it is any other pattern of 1's and 0's in the 3x3 grid.

As noted, this is a simple problem to solve with the usual 'procedural' coding that you learned in first year - a simple **if** statement in Python. It is also possible to determine a correct answer for the linear classifier by inspection, as also discussed in class. Instead, the goal in this assignment is to gain an understanding of the *learning from data* approach, and to use the specific method of learning employed in the successful deep learning approach: an artificial neural network trained with gradient descent. This will underpin your understanding when we apply various versions of this approach to much more difficult problems in later assignments and your course project.

1	0	1
0	1	0
1	0	1

Figure 1: Problem Definition - 'Recognize' this Pattern

2 Neural Network Classifier Method to Solve Problem

The neural net machine learning method is illustrated in Figure 2. It shows the nine inputs of the 3x3 array (the I_i in the figure) all being fed into a single artificial neuron, which computes the linear function $Z = (\sum_{i=0}^{8} w_i I_i) + b$, and then passes it through an *activation function*, such as a sigmoid, ReLU or just linear ($Y = Z$) as described in class. The weights (w_i) and bias (b) must be determined, through a training process, so that the output Y correctly indicates whether the input pattern is the one shown in Figure 1 or not. Note that we will use the following *classification rule* (which I called a decision function in class) on the output Y to determine which outcome the model is predicting: when $Y \geq 0.5$ the output will be classified as and X, and when $Y < 0.5$ the output will be classified as 'not an X.'

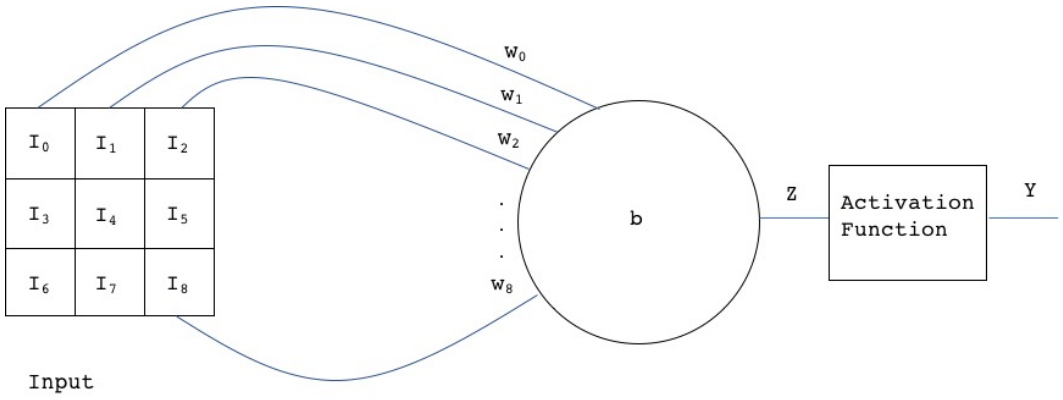

Figure 2: Single Neuron Classifier

The sections below will ask you to write the code implements this neural network computation an *trains* it to set the values of the weights (w_i) and bias (b) parameters.

3 Solve by Inspection (Total Points: 10)

In class we discussed this structure above, using classification rule whose decision point was 0. In class we determined, by inspection, values for the weights (w_i) and bias (b) parameters that would make the classifier work for that classification rule. For the questions below, assume that the classification rule is the with the *0.5 decision point*. Answer the following questions:

i. What are values of w_i and b that will make the classifier 100% correct for all possible inputs using the 0.5 decision point classification rule?

ii. Is the answer to (i) unique? If your answer is yes, say why. If your answer is no, give a second answer that uses different weights and bias.

ii. How many unique inputs (that is, different instances of $I = \{I_0, I_1, ..., I_8\}$) are possible for the 3x3 grid?

iii. Does your solution easily scale to solve finding a single X-type pattern in a 4x4 grid? Explain why or why not.

iv. Suppose that, on a 5x5 grid, you had to match the 'X' as above, but, in addition, an 'X' shifted left by one, and shifted right by 1 position also had to be matched (in the shifted cases, part of the X would be missing). Could you as easily create the single-neuron parameters to solve that problem? Why or why not?

4 Training from Data

You are to write a Python-based program, in a file called `a2.ipynb`, using either Google Colab or Jupyter Notebook, that trains the single-neuron classifier shown in Figure 2 using the gradient descent method as described in class. This includes the calculation of parameter gradients (for the weights and bias) through analytic equations. Your code should be written to make it easy to select the following different choices for hyperparameters:

1. The activation functions should be selectable as: `sigmoid`, `ReLU` and `linear`. (linear means $Y = Z$ in Figure 2). Note that in class we derived the equations for the gradient of the weights and bias for the *linear* activation function, making use of the chain rule. While the derivative of the linear activation function is the constant one, the derivative of the `sigmoid` and `ReLU` functions is not as simple.

2. The learning rate.

3. Number of epochs (number of times the entire training set is used to determine a modification to the weights and bias).

4. Random number seed (setting this value differently changes the random initialization of the weights).

You should use the separate files of data to train and then validate the training. These are provided for you in the associated assignment files named as follows:

File	Contents
`traindata.csv`	200 example 3x3 grids
`trainlabel.csv`	labels for training examples
`validdata.csv`	20 different 3x3 grids
`validlabel.csv`	labels for validation examples

The data in the files is formatted as follows: the `traindata.csv` and `validdata.csv` files contain one sample input per line, given as nine numbers, separated by commas, consecutively representing the inputs $I_0, I_1, ...I_8$. The `trainlabel.csv` and `validlabel.csv` files contain one

number per line, either 0 or 1, indicating if the corresponding line in the Data file is a match for the X pattern (with a value of 1) or not a match (with a value of 0). Be sure to view the files with a text editor of some kind to make sure you agree that the labels are correct.

You can use the `numpy` function `loadtxt` to read this data into your code.

5 Guidance

In this section we give guidance as to the various choices you'll need to make in coding this assignment.

1. You should use a mean squared-error loss function.
2. You should initialize the weights and bias to random numbers between 0 and 1.
3. Be sure to set the seed for the random generator function (using `random.seed()`) so that you can set this for the experiments as required below.
4. Make the activation function easy to change in to your code, by changing a setting at the top of your code.
5. It will be useful (and required) to visualize the weights w_0 through w_8, and we have provided a function that takes in the weights and two parameters and outputs a gray scale 'picture' of the weights, to see if they 'look' correct or close to correct. This function is provided in the assignment, in the file `dispkernel.py`. This function makes the lowest value weight the colour black, and the highest white, and everything elese is in-between black and white. You will need this function to produce some of the outputs required in Section 6.
6. I have to confess that I am somewhat worried about using Notebooks (Jupyter or Colab) in this course, which we're doing for the first time. Most professional programmers won't use notebooks, but instead use straight-up Python (`file.py`) code files. One advantage of that, even for this assignment, is that you can use command-line arguments to vary the hyperparameters (i.e the learning rate, or the activation function, or number of epochs). If you still wanted to do that, the skeleton code `comm_parm_a2.py` shows you how to use *command line arguments* , using the `argparse` library. See https://docs.python.org/3/howto/argparse.html for a good description of the library. If you did this, it might be easier to run scripts with the different hyper-parameters.

6 Experiments and Outputs to Hand In (40 points)

As you write and debug your code, you'll need to test individual parts of your code in the usual way (by inspecting the output from subsections of the code, after setting the input). With machine learning, you will also need to inspect the *learning curves* which shown the progress of the classifier's *loss* function and *accuracy* after each step. We are interested to see if the *training* loss and accuracy are improving after each epoch, and also whether the *validation* loss and accuracy are improving. So, in your code, use `matplotlib` to show how loss and accuracy are changing vs. epoch. Be sure to put the training and validation loss on one plot (as is typical practice in the field), and the training and validation accuracy on a second plot.

You will need to experiment with the learning rate parameter to find one that works well, and to determine how many epochs are needed to succeed. (We expect to succeed for this problem by the way, since we know that there are good solutions as determined in Section 3.)

In your report, `assign2.pdf` you should hand in data, tabular form, that shows the effect of the hyperparameters listed below on the training and validation accuracy. For each item below, select (and report) reasonable values of the *other* parameters. (e.g. when showing the effect of epoch, choose fixed values for learning rate and activation function that give a good sense of what the variation in number of epochs does):

1. The number of epochs.
2. The learning rate - be sure to show a range where the learning rate is too high, and where it is too low.
3. The effect of the three activation functions - linear, sigmoid and ReLU. Explain why the best one came out best.
4. The effect of 5 different random seeds; for this choose a learning rate that isn't the best, but one that does not succeed as well as your best. Explain why the answers differ.

Finally, you should determine a single set of parameters that achieve the best result you can get, but in the fewest epochs. In your report, say what those parameters are, and what test and validation accuracy was achieved.

In addition, in you should create and hand in *properly labelled* loss and accuracy plots vs. epoch, as well as the nine weights displayed using the dispKernel function, as described above, for the following cases:

1. Show an example where the learning rate is too *low* (which means learning is too slow). Give an explanation as to what is happening in this case.
2. That shows an example where the learning rate is too *high*. Explain.
3. That shows a 'good' learning rate.
4. Give the two plots for each of the three activation functions, `linear`, `sigmoid` and `ReLU`.

7 Re-Implement the Same Classifier Using PyTorch

Now that you have a sense of how inference and training are done when coded from scratch, you need to learn how to do exactly the same thing using the `PyTorch` deep learning library. PyTorch (and its cousin, `TensorFlow`) automatically do many of the things you've just programmed from scratch, and much much more. The key in this part of the assignment is to be clear on which function/class is doing the same things that you coded from scratch in the first parts of this assignment.

7.1 Set up your Environment

If you are using Google Colab, then there is nothing to install. That environment already contains the PyTorch and TorchVision libraries by default.

If you have installed Anaconda and Python as described in Assignment 1, you will also need to install the PyTorch libraries, as described in the Section below.

7.1.1 Installing PyTorch (if not using Google Colab)

1. On a command line, activate the `conda` environment for this course first: `conda activate ece324` (macOS/Linux) or `activate ece324` (Windows).

2. To download PyTorch go to https://pytorch.org/, which at the bottom of the page will give you a `conda` command to download the correct verson for your specific operating system and hardware.

3. You can test that your system has installed PyTorch successfully by bringing up a Python interpreter (i.e. run `python` in command line while inside your `ece324` conda environment) and running '`import torch`'.

7.2 Re-implementation of Classifier in PyTorch (10 points)

Above you were asked to code a single-neuron classifier to recognize the 'X' pattern of ones in a 3x3 'picture.' You should modify that code (largely re-using the data input code, and the plotting output code) and re-implement it using PyTorch, as described in class. You should use the nn.Linear module from PyTorch to implement the one neuron, and SGD optimizer to train it. Make sure you understand the torch.tensor object, and how to work with it, including bringing your data into that form from a `Numpy` array. Also try to be clear on what causes the parameters to be instantiated and initialized, as that becomes somewhat invisible in PyTorch. Finally, try to see what gradients are being computed, and be sure to understand what PyTorch method causes that to happen.

Submit your code as a Notebook file `part2.ipynb`. Also, answer/submit results from the following questions, as part of the `assign2.pdf` you submit:

1. (2 points) For the single-neuron classifier that you instantiate, what is the full name of the tensor object that contains the weights, and what is the name of the object that contains the bias?

2. (2 points) What is the name of the tensor object that contains the calculated gradients of the weights and the bias? (This was not covered in class).

3. (2 points) Which part of your code computes gradients (i.e. give the line of code that causes the gradients to be computed). Explain, in a general way, what this line must cause to happen to compute the gradients, and how PyTorch 'knows' how to compute the gradients.

4. (4 points) Give the training/validation plot versus epoch, and the accuracy vs. epoch for the three cases required in Section 6: a too-slow learning rate, a too-fast learning rate, and a 'just-right' learning rate.