Assignment 2: Single-Neuron Classifier Coded from Scratch

Deadline: September 19, 2019 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, from scratch with almost no library support, the basic linear function of neuron. More significantly, you'll use a set of data to both train and 'validate' the training. This training will make use of the gradient descent method of optimization, as discussed 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 process and engine that is used in all modern 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, which you must add to the skeleton code given in the file `a2.py`. This skeleton code shows you how to input the parameters to your python function, and it must be able to generate all of the data required in Section 6.

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 (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 a '1' when the input is the pattern in Figure 1, and a '0' when it is any other pattern of 1's and 0's in the 3x3 grid.

1	0	1
0	1	0
1	0	1

Figure 1: Problem Definition - 'Recognize' this Pattern

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. 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.

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.

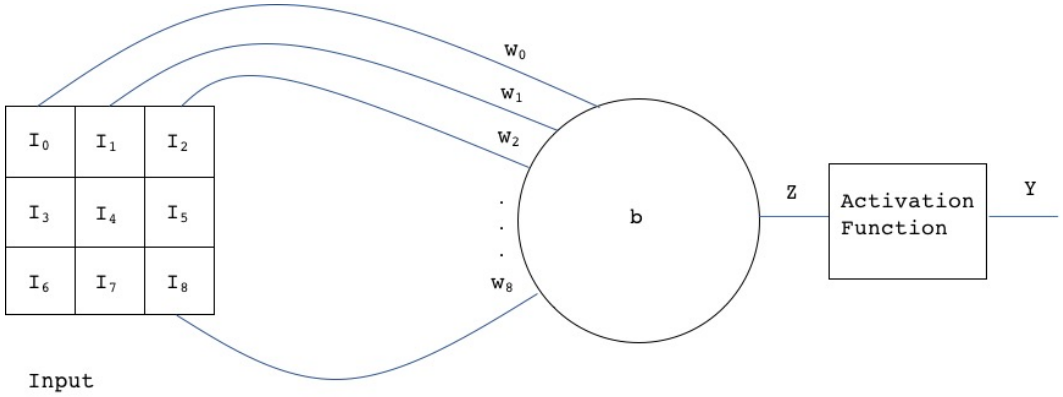

Figure 2: Single Neuron Classifier

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 a ReLU activation function, and determined by inspection, values for the weights (w_i) and bias (b) parameters that would make the classifier work as described. For this to work, you must also say how to interpret the output, Z, of the classifier - i.e. say in words what output values of Z indicate a 'match' with the required pattern and which indicate 'no match.' Answer the following questions relating to the problem of solving for the weights by inspection:

i. Is the answer that we discussed in class unique? If your answer is yes, say why. If not, 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 a 4x4 problem, and an NxN problem? 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. 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.py`, using PyCharm, 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 (i.e. weight and bias) gradients through analytic equations. Your program should have input parameters that allow you to easily be able to select the following different choices or parameters (see below for a pointer on how to do this parameterization easily):

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 (as described in class).

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).

As described in class 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 example 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 a selection as an input to your program.

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. The skeleton code shows you how to use *command line arguments* (which are used to tell a program things like the learning rate you want to use, or the activation function, or number of epochs), using the `argparse` library. See `https://docs.python.org/3/howto/argparse.html` for a good description of the library.

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 shown 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 following 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. Indicate in your report, what those parameters are, and what test and validation accuracy you 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 the three activation functions, `linear`, `sigmoid` and `ReLU`.