**CPSC 470 – Artificial Intelligence**

**Problem Set #6 – Deep Neural Network**

**\* points**

**Due Monday April \*, 11:59:59pm**

Some reminders:

* **Grading contact:** \* (\*) is the point of contact for initial questions about grading for this problem set.
* **Late assignments** are not accepted without a Dean’s excuse.
* **Collaboration policy:** You are encouraged to discuss assignments with the course staff and with other students. However, you are required to implement and write any assignment on your own. This includes both pencil-and-paper and coding exercises. You are not permitted to copy, in whole or in part, any written assignment or program as part of this course. You are not to take code from any online repository or web source. You will not allow your own work to be copied. Homework assignments are your individual responsibility, and plagiarism will not be tolerated.
* **Students taking CPSC570:** There is no extra section for this assignment. Your assignment is the same as CPSC470.

In this exercise, you will implement part of a deep neural network and apply it to the task of hand-written digit recognition. This assignment is adapted from Andrew Ng’s machine learning class on coursera.

# Linear Algebra and Numpy

The mathematics tools for deep neural network is linear algebra. For the purpose of this assignment, you don’t need to know a lot about matrices to finish this assignment, as most of the code has been implemented for you. However, if you find it challenging, here are some basic linear algebra that may help:

<https://minireference.com/static/tutorials/linear_algebra_in_4_pages.pdf>

<https://math.boisestate.edu/~wright/courses/m365/LAIntroSlides.pdf>

To represent matrices in python, we used the numpy library. Again, as most of the code has been implemented for you, if you find it challenging or would like to have an overview of how to use numpy, the links below may be helpful:

<http://cs231n.github.io/python-numpy-tutorial/#numpy>

<https://www.numpy.org/devdocs/user/quickstart.html>

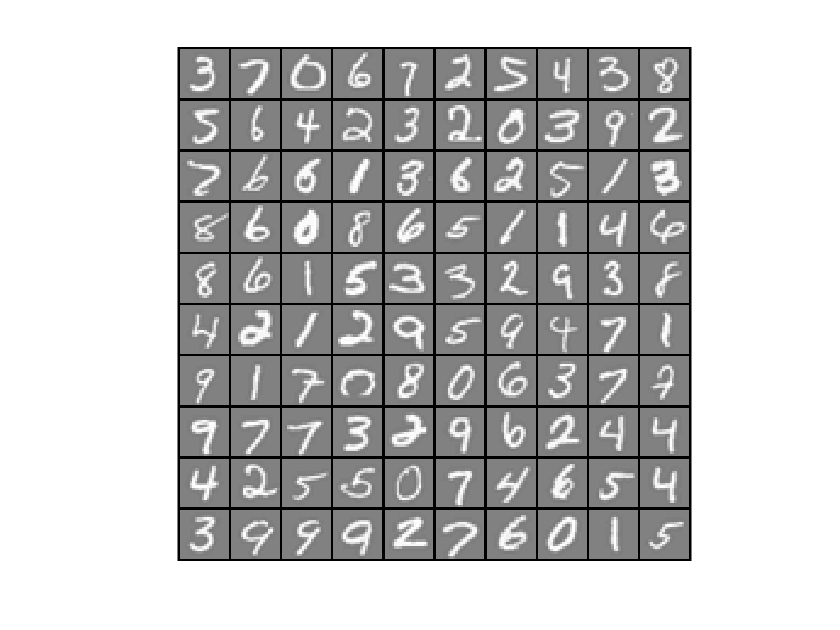
And here is the numpy documentation:

<http://www.numpy.org/>

All of the libraries needed of this assignment has been installed on zoo. And unfortunately, we won’t be able to help with library installation problems due to the size of the class.

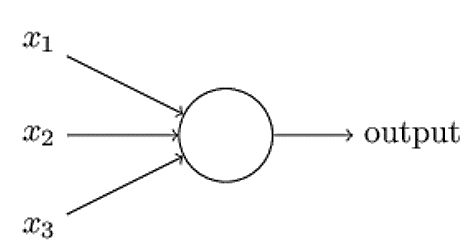
# Dataset

The dataset you will use is taken and modified from the MNIST digit dataset (<http://yann.lecun.com/exdb/mnist/>). The dataset consists of 5000 handwritten digit images and the corresponding labels. Each image is 20 pixel by 20 pixel. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. The figure below showed some examples from the dataset.

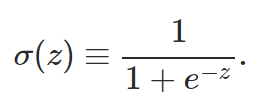


# Neural Networks

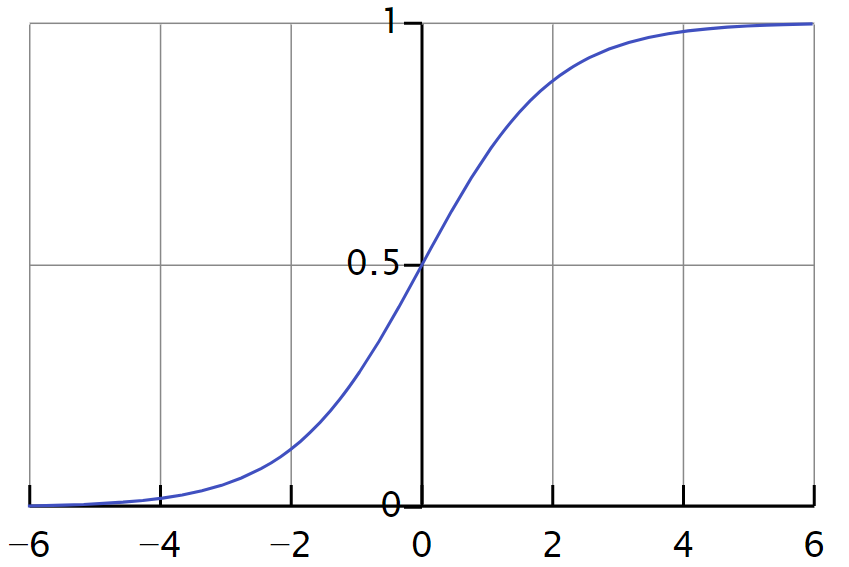
Here is a basic unit/node of a neural network:



It takes weighted inputs. In this example, z = w1 \* x1 + w2 \* x2 + w3 \* x3 + b (assume w1, w2, and w3 are the weights correspondingly, b is the bias which didn’t show in the figure above). The output is a = g(z) where g is a non-linear activation function. In this assignment, we use the sigmoid function which is defined as:



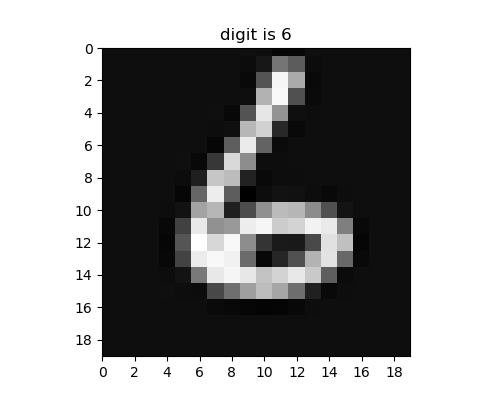
A sigmoid function looks like this:



The neural network is composed of those units. There are many such units in each layer, and there are many layers. In this assignment, we will firstly use a 3-layers neural net. The 3 layers are an input layer, a hidden layer and an output layer. Recall that our inputs are pixel values of digit images. Since the images are of size 20 by 20, this gives us 400 input layer units (not counting the extra bias unit which always outputs +1).

The training data will be loaded into the variables train\_x and train\_y by the function load\_data(training\_percentage). We will play around the training\_percentage, but for now let’s leave it as 1. You can ignore the test\_x and test\_y generated by the load\_data function for now.

The train\_x contains 5000 vectorized samples, and the train\_y stores the corresponding labels like 6, 1, 2, etc. To visualize a sample from the training examples, please use the display\_digit\_image(…) function, and please refer to the source code for documentation. One example of image looks like this:



For the hidden layer, we will first use one with 25 units, and we will play around with it later. For the output layer, we will set it to have 10 units.

**Please answer the following question:**

**Q1. Why there are 10 units in the output layer? Please choose one (\* points):**

**B**

**A. Because 10 can be divided by 400 \* 25;**

**B. Because there are 10 labels;**

**C. The number 10 is generated randomly, and it can be any number here.**

Here is a figure of the neural network:



The vectorized image will be used as the information for the input layer. Each node corresponding to a pixel in the image. The information will then be weighted and pass to the hidden layer. The value at the hidden layer will then go through the activation function and the result will be passed to the output layer. The value will also go through the activation function. At the output layer, the node of the largest value will win and the label represented by the node will be the image’s label. This process is called feedforward.

Since the weights are initialized randomly, the output will be very different from the true label. The cost function is used to evaluate how different it is between the true label and the predicted label. The network will then propagate this information back by calculating the derivatives. The weights of the network will update themselves accordingly. This process is called backpropogation.

**Most of the code has been implemented for you. Here you task is to complete the deep\_NN function by choosing the correct functions. You will implement about 4 lines of code.**

# Feedforward

The initialize\_parameters function returns a dictionary parameters, with the keywords “W1”, “b1”, “W2” and “b1”. The “W\*” represents the weight matrix, and the “b” is the bias vector. “1” means the parameters from the input layer to hidden layer, and “2” represents the parameters from the hidden layer to the output layer. By examine the output of the input\_parameters function, please fill in the dimensions of the parameters in the table below (you may need to write a few lines of code to call the initialize\_parameters, however, you don’t need to submit any code you wrote you this part. You can use the np.shape() function).

**Q2. Please fill in the dimensions (\* points)**

|  |  |  |
| --- | --- | --- |
| **W1** | **25** | **400** |
| **b1** | **25** | **1** |
| **W2** | **10** | **25** |
| **b2** | **10** | **1** |

Now let’s examine the dimensions of each layer. The feedforward function returns a dictionary of caches, with the keyword “a1, z2, a2, z3, a3”. “z\*” represents the weighted result at the corresponding layer. “a\*” represents the value of sigmoid(“z\*”) at the layer. “1” refers to input layer, “2” refers to the hidden layer, and “3” refers to the output layer. Please note that the overall output of the neural net is also represented as AL/al, which should share the same value as a3.

**Please answer the following questions**

A

**Q3. Why there is no “z1”. Please choose one (\* points):**

**A. Because 1 is the input layer, the output of layer 1 is the pixel values themselves;**

**B. Because Godzilla ate “z1”;**

**C. “z1” should exist, but it is just not used in the calculation later. So this value is excluded.**

**Q4. Please fill in the dimensions below (\* points):**

|  |  |  |
| --- | --- | --- |
| **X (input, which is train\_x)** | **400** | **5000** |
| **y (true label, which is train\_y)** | **1** | **5000** |
| **Y (converted label, which is rehape\_Y(train\_y))** | **10** | **5000** |
| **a1** | **400** | **5000** |
| **z2** | **25** | **5000** |
| **a2** | **25** | **5000** |
| **z3** | **10** | **5000** |
| **a3** | **10** | **5000** |

Here you can see that the dimension of Y and y is different. This is because the raw data provided 1, 2, 3, 4 as labels, and the output of the neural net is a vector of length 10, with index 0 corresponds to the probability of digit 0, index 1 corresponds to the probability of digit 1, etc. Based on the information provided here and above,

**Please answer the following questions.**

**Q5. Given a column of Y: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0].T . From 0 to 9, which digit is this label corresponding to:**

**2**

**Q6. Given a column of a3 from a trained neural network with accuracy 99.4%: [0.02, 0.06, 0.1, 0.0004, 0.9, 0.3, 0.1, 0.025, 0.062, 0.12].T . From 0 to 9, which digit is this image mostly represents:**

4

That’s all for feedforward. You don’t need to implement any part of the feedforward algorithm and all has been done for you. If you are curious how the difference between the prediction and the true label is calculated, here is the formula and this is beyond the scope of this assignment:

# Backpropogation

Here is how to update the weights. The main idea is to calculate the “differences” (which is actually derivative. You can perceive it as a way to evaluate as difference, but it is not just a simple subtraction in quantity) of the current value to the target value (which is denoted as dA\*, where \* is the layer), and thus figure out the differences (denoted as dW\* and db\*) between the current weights and the ideal value of weights, and update the weights accordingly. And we will work backwards, which is from the output layer to the input layer.

At the output layer, the dA3 is calculated as:

At the hidden layer, the dA2, dW2 and db2 are calculated as (\* is matrix multiplication, and .\* is element-wise multiplication)

At the output layer, dW1 and db1 is calculated as:

**Please answer the following question.**

B

**Q7. Why there is no dA1?** **Please choose one (\* points):**

**A. dA1 should exist, but it is just not used in the calculation later. So this value is excluded;**

**B. If we are going to calculate dA1, that will be the difference between the desired value and the actual value we got. The actual value is the image, and we cannot modify the input. So there is no need to get dA1.**

**C. Because the TA made a mistake, there should be dA1.**

The formulas provided above are specific to a 3-layer neural network. Please think about how to generalize it to an n-layer neural network and answer the following questions.

**Q8. For the output layer of an n-layer neural network, the dAn is calculated the same way, which is (\* points):**

**C**

**A.**

**B.**

**C.**

**Q9. For the hidden layer of an n-layer neural network, the dAi is calculated with (\* points):**

**A**

**A.**

**B.**

**C.**

**Q10. For the layer other than the output layer of an n-layer neural network, the dWi is calculated with (\* points):**

**B**

**A.**

**B.**

**C.**

**Q11. For the layer other than the output layer of an n-layer neural network, the dbi is calculated with (\* points):**

**C**

**A.**

**B.**

**C.**

**Now you have a good understanding of how the weights are updated, please go ahead and implement the corresponding part in the function backpropogation. You will implement about 5 lines of code.**

Let’s take a look at how the parameters are updated. Since we have the differences dWi and dbi, the parameters, the parameters are updated simply as:

However, we would like to get some control of how the parameters are updated, so we add a scalar coefficient here called learning rate. As the name indicated, learning rate implies how quickly the model learns or update itself. So the updated formula is:

**Using the formula above, please complete the update\_parameters function. You will implement about 3 lines of code.**

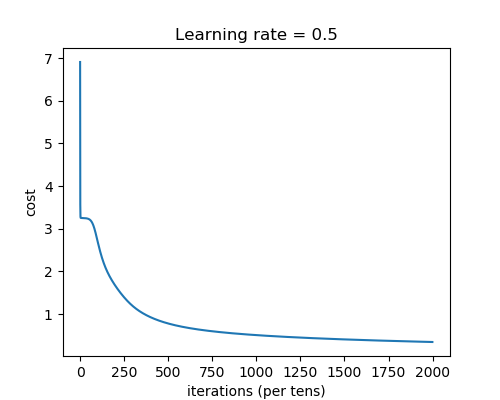
# Train the neural network

Congratulations! Now you have a completed fully-connected deep neural network. Let’s train the neural network by running the existing code in the main section under “section 1”. Please do not modify any of the hyper-parameters here like training\_percentage, learning\_rate, num\_iterations, etc. It will take a few minutes to run and you will see the cost of every iterations printed to screen. At the end, it will print the accuracy of this model on the training set and a figure with the cost of each iteration.

**0.9572**

**Q12. Please fill in the accuracy (\* points):**

**Q13. Please copy and paste your figure below (\* points)**



# Training Set and Test Set

Currently all the dataset is used to train the model. To evaluate a neural network, we usually separate the dataset into two parts, with the majority of data set to be the training set and the rest being testing set. This ensures the test sets shares some commonality with the training set (e.g., samples from similar situation, guidelines, etc.) and the same time test the neural network with examples it has never seen before.

To separate the dataset to training set and test set, we changed training\_percentage to 0.8, which is we randomly choose 80% of the dataset as training set and the rest as test set. Please comment out section 1 and uncomment section 2 and run the code. You will see the training set accuracy is 0.959 and the testing set accuracy is 0.932. Please note that if you hard code the total sample size to be 5000 anywhere in the code, you might not get the correct result. Please go back and fix it.

Please answer the following question.

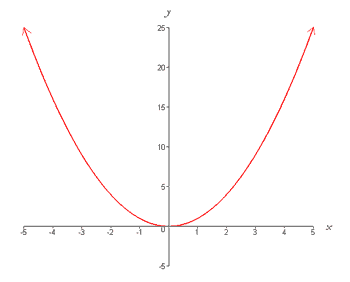
**Q13. Why is the testing set accuracy is lower than the training set accuracy? Is it by chance? (\* points)**

**It is not by chance. The model is trained with the training set, so it will predict more accurately on the training set than a set it has never seen before, which is the testing set.**

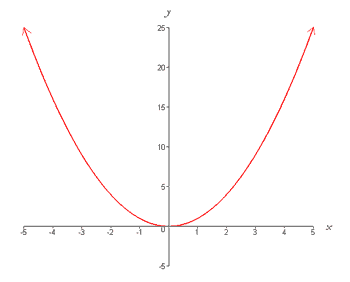
# Learning Rate

Now let’s play around with the learning rate and set it to a large number, 10. Theoretically, this neural net should learn very fast. To see whether this is true, please comment section 1 and 2, and uncomment section 3 and run the code.

You will see that the cost stays at 3.250830, and the accuracy is 0.0934, which is at chance level. This is actually not surprising, and let’s take a look at why. What we do in backpropagation is to find the weights that minimize the cost. For a similar version like the quadratic curve:



It is equivalent to find the x that result in the smallest y value. Suppose the black dot is where we are, then gradient descend (the method we use in backpropagation) simply take as down the hill and to towards the minima in small steps. If we slightly increase our steps, we will get to the minima faster. However, what will happen if the steps are too big? Since W = W – learning\_rate \* dW, where W is the derivative, which will increase as we move along, so we will end up something like this:



And this is what happens when we set the learning rate to 10. However, different from the situation above, the cost we got just stayed at 3.25 and didn’t “explode” like the one above.

**Please answer the question below.**

**Q14. Why that the cost we got stayed at 3.25 and didn’t increase significantly? (\* points)**

**Any one will be acceptable:**

**1) The accuracy is at chance level, which is the worst it can get;**

**2) it stucks at a local minima.**

# Number of Iterations and More layers

Now let’s play around with the number of iterations and set it to a large number, 30000, and let’s train a neural network with 4 layers. To see whether this is true, please comment section 1, 2 and 3, and uncomment section 4 and run the code. It will take longer to complete compared to the previous few sections, and please be patient.

You will see that the training set accurate has been improved and it is 1.0. But the test set accurate is worse than before which is now 0.931. This is called overfitting. Please search online why this happens and explain it.

**Q15. Please explain why overfitting happens. (\* points)**

**Overfitting the model generally takes the form of making an overly complex model to explain idiosyncrasies in the data under study. Thus attempting to make the model conform too closely to slightly inaccurate data can infect the model with substantial errors and reduce its predictive power.**

# Your own experiment with the hyperparameters

You may have more questions to how the hyperparameters influence the neural network, and now is the time for you to experiment with the hyperparameters and try to find out the answers on your own. Please comment sections 1 to 4, and uncomment section 5. Please feel free to experiment with the training\_percentage, learning\_rate, layers\_dims which defines the architecture of the neural network, and num\_iterations. Please document your experiment below.

**Q16. What is the purpose of your experiment, which is, which question are you trying to address? (\* points)**

**Q17. What are the values of the hyperparameters in your experiment? (\* points)**

|  |  |
| --- | --- |
| **training\_percentage** |  |
| **learning\_rate** |  |
| **layers\_dims** |  |
| **Num\_iterations** |  |

**Q18. Please describe the results you found? Here please describe the direct results here, like the accuracy, etc. (\* point)**

**Q19. What is your conclusion, and please briefly explain why this happens.** **Also are there any limitations in your experiment? (\* point)**

# Submission

We encourage to type in your answers directly on this handout. And please only put your answers in the designated areas and we will ignore anything that is not in the designated area.

Also please do not alter the format of this file, otherwise we may not able to find your answer at the right place and you may lose points.

Please submit this file to Problem Set 6 on canvas.

And please submit your code to Problem Set 6 - Programming.