Homework 2
600.482/682 Deep Learning
Spring 2020

February 24, 2020

Due Sun Feb. 23 11:59pm.
Please submit a report (LaTeX generated PDF) and
the notebook as python file (file → download .py)
to Gradescope with entry code 9G83Y7
(submit the code as programming assignment)

1. The goal of this problem is to minimize a function given a certain input using gradient descent by breaking down the overall function into smaller components via a computation graph. The function is defined as:

$$f(x_1, x_2, w_1, w_2) = \frac{1}{1 + e^{-(w_1x_1+w_2x_2)}} + 0.5(w_1^2 + w_2^2)\,.$$

   (a) Please calculate $\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}$.
   Answer: $\dfrac{\partial f}{\partial w_1} = \frac{x_1e^{-(w_1x_1+w_2x_2)}}{(1+e^{-(w_1x_1+w_2x_2)})^2} + w_1$, $\dfrac{\partial f}{\partial w_2} = \frac{x_2e^{-(w_1x_1+w_2x_2)}}{(1+e^{-(w_1x_1+w_2x_2)})^2} + w_2$, $\dfrac{\partial f}{\partial x_1} = \frac{w_1e^{-(w_1x_1+w_2x_2)}}{(1+e^{-(w_1x_1+w_2x_2)})^2}$, $\dfrac{\partial f}{\partial x_2} = \frac{w_2e^{-(w_1x_1+w_2x_2)}}{(1+e^{-(w_1x_1+w_2x_2)})^2}$

   (b) Start with the following initialization: $w_1 = 0.3, w_2 = -0.5, x_1 = 0.2, x_2 = 0.4$, draw the computation graph. Please use backpropagation as we did in class.
   You can draw the graph on paper and insert a photo into your report.
   The goal is for you to practice working with computation graphs. As a consequence, you must include the intermediate values during the forward and backward pass.

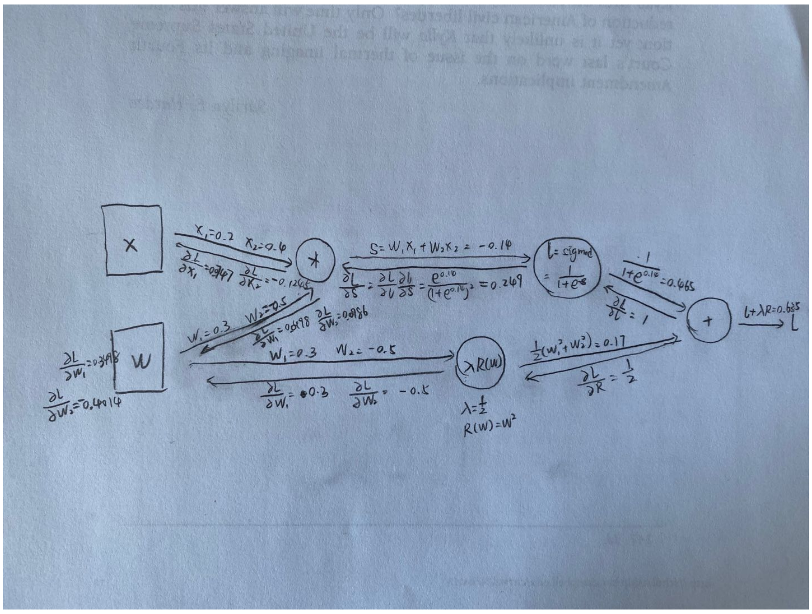

Figure 1: computation graph

(c) Implement the above computation graph in the complimentary Colab Notebook using numpy. Use the values of (b) to initialize the weights and fix the input. Use a constant step size of 0.01. Plot the weight value w_1 and w_2 for 30 iterations in a single figure in the report.

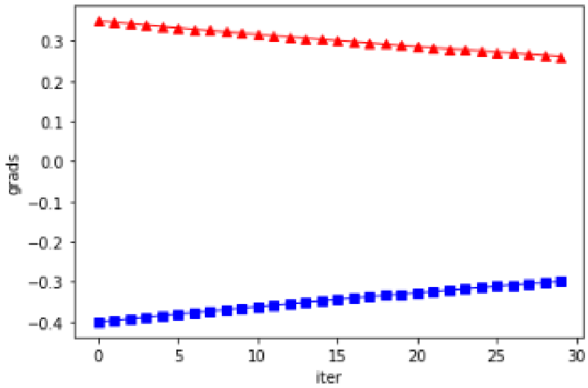

Figure 2: weigh value

In the figure, the red line is w_1 and the blue line is w_2

2. The goal of this problem is to understand the classification ability of a neural network. Specifically, we consider the XOR problem. Go to the link in footnote[1] and answer the following questions. *Hint: hit reset the network right next to the run button after you change the architecture.*

   (a) Can a linear classifier, without any hidden layers, solve the XOR problem?

   Answer: linear layer cannot solve the XOR problems.

   (b) With one hidden layer and $\text{ReLU}(x) = \max(0, x)$, how many neurons in the hidden layer do you need to solve the XOR problem? Describe the training loss and estimated prediction accuracy when using 2, 3 and 4 neurons. Discuss the intuition of why a certain number of neurons is necessary to solve XOR.

   Answer: when using 2 neurons, the training loss is 0.263 and prediction accuracy is 75.1%. when using 3 neurons, the training loss is 0.06 and prediction accuracy is 99.3%. when using 2 neurons, the training loss is 0.001 and prediction accuracy is 99.9%.
   I think the reason is that one neuron can only represent one of NOT, AND and OR. At least three of them can combine to represent XOR, like:
   a XOR b = ((NOT a) AND b) OR (a AND (NOT b))

3. In this problem, we want to build a neural network from scratch using Numpy for a real-world problem. We consider the MNIST dataset (http://yann.lecun.com/exdb/mnist/), a hand-written digit classification dataset. Please follow the formula in the complimentary Colab Notebook. *Hint: Make sure you pass the loss and gradient check in the notebook.*

   (a) Implement the loss and gradient of a linear classifier (python function `linear_classifier_forward_and_backward`).

   (b) Implement the loss and gradient of a multilayer perceptron with one hidden layer and $\text{ReLU}(x) = \max(0, x)$ (python function `mlp_single_hidden_forward_and_backward`).

   (c) Implement the loss and gradient of a multilayer perceptron with two hidden layer, skip connection and $\text{ReLU}(x) = \max(0, x)$ (python function `mlp_two_hidden_forward_and_backward`).

[1] https://playground.tensorflow.org/#activation=relu&batchSize=10&dataset=xor&regDataset=reg-plane&learningRate=0.01&regularizationRate=0&noise=0&networkShape=&seed=0.10699&showTestData=false&discretize=true&percTrainData=80&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

(d) Plot the development accuracy of each epoch of three models in a single figure using the following hyperparameters: the batch size is 50, the learning rate is 0.005 and the number of epochs is 20.

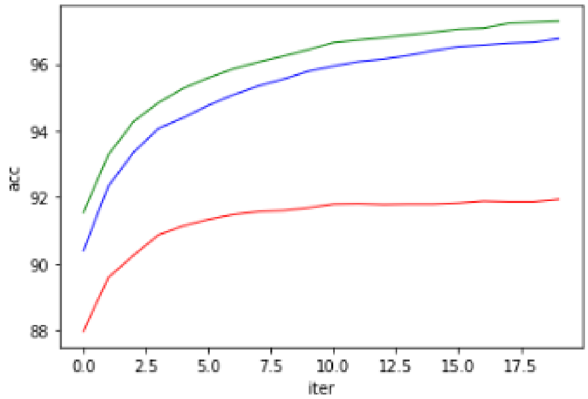

Figure 3: accuracy

In the figure, the red line is linear, the blue line is single hidden layer and the green line is two hidden layer.

(e) Try using other hyperparameters and select a set of best hyperparameters using **development accuracy**. Once you pick the best model and hyperparameters, include the development accuracy of each epoch into the above figure (make a new figure) and report the **test accuracy** of the selected model and hyperparameters.

```
 20%|| 1/5 [00:28<01:55, 28.84s/it]epoch 0 train loss = 0.001 dev accuracy = 98.17%
 40%|| 2/5 [00:57<01:26, 28.87s/it]epoch 1 train loss = 0.001 dev accuracy = 98.17%
 60%|| 3/5 [01:26<00:57, 28.92s/it]epoch 2 train loss = 0.001 dev accuracy = 98.17%
 80%|| 4/5 [01:55<00:28, 28.83s/it]epoch 3 train loss = 0.001 dev accuracy = 98.17%
100%|| 5/5 [02:24<00:00, 28.76s/it]epoch 4 train loss = 0.001 dev accuracy = 98.17%
best dev accuracy = 98.17%
```

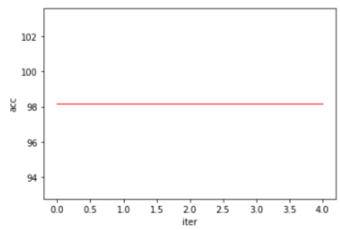

Figure 4: best training

The best parameters I try are that batch size is 20, learning rate is 0.001 and the epoch is 5. The single hidden layer's performance is the best.