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)

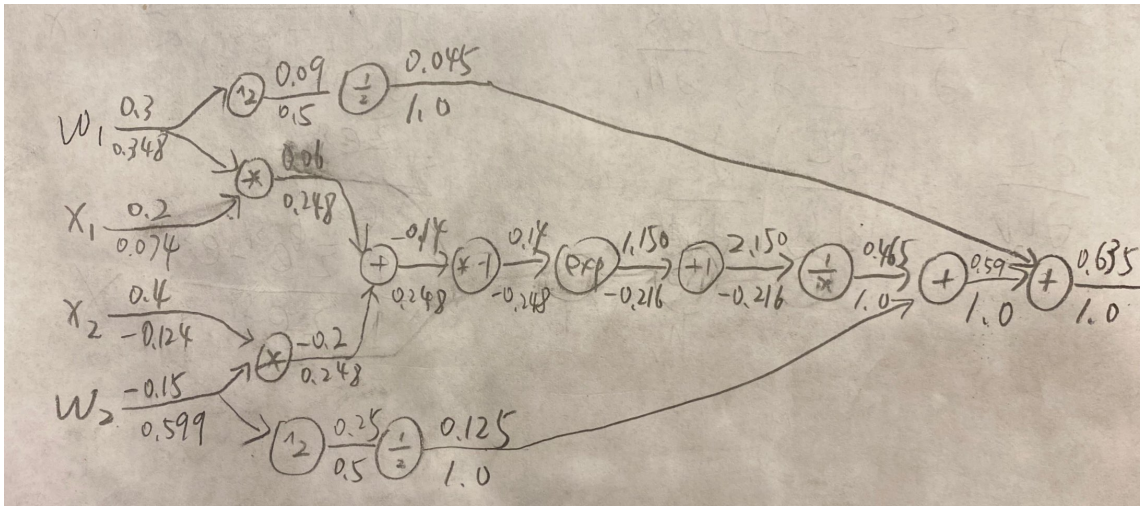
1. The goal of this problem is to minimize a function given a certain input using gradient descent by breaking down the overall function into smaller components via a computation graph. The function is defined as:

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

   (a) Please calculate $\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}$.
   Solution:

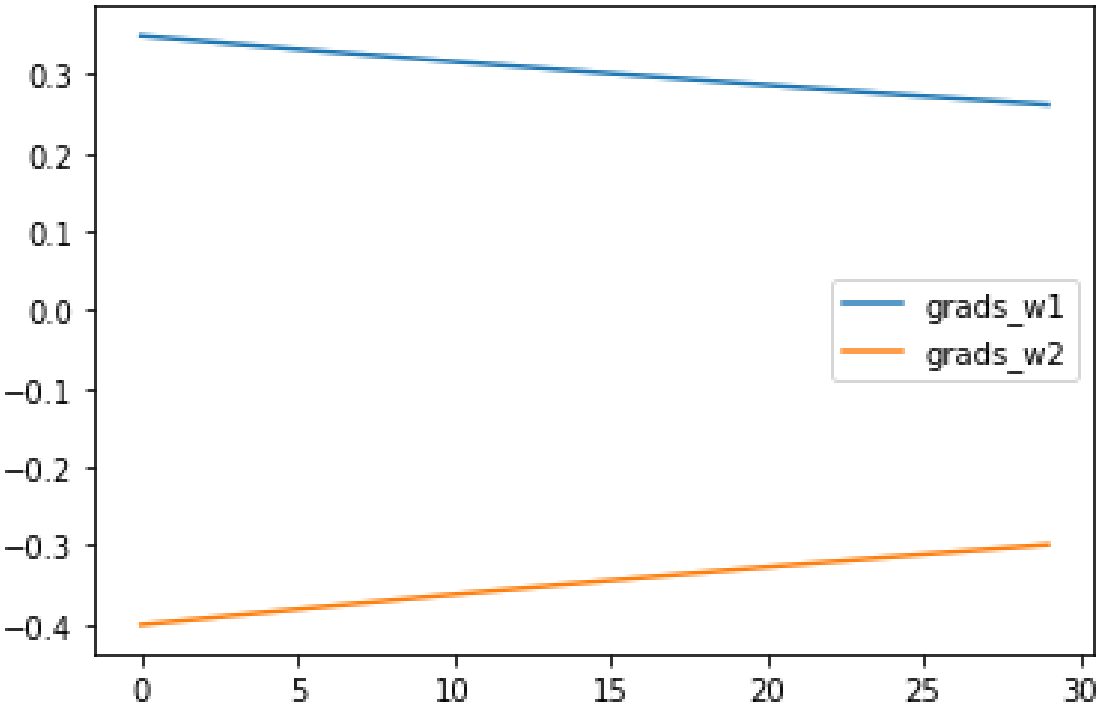
$$\frac{\partial f}{\partial w_1} = \frac{x_1 \cdot e^{-(w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_1 x_1 + w_2 x_2)})^2} + w_1$$
$$\frac{\partial f}{\partial w_2} = \frac{x_2 \cdot e^{-(w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_1 x_1 + w_2 x_2)})^2} + w_2$$
$$\frac{\partial f}{\partial x_1} = \frac{w_1 \cdot e^{-(w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_1 x_1 + w_2 x_2)})^2}$$
$$\frac{\partial f}{\partial x_2} = \frac{w_2 \cdot e^{-(w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_1 x_1 + w_2 x_2)})^2}$$

   (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.
   Solution:
   The computation graph is shown as below. All number above the lines are values in forward pass. All numbers below the lines are values in backward pass.



(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.

Solution:



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.*

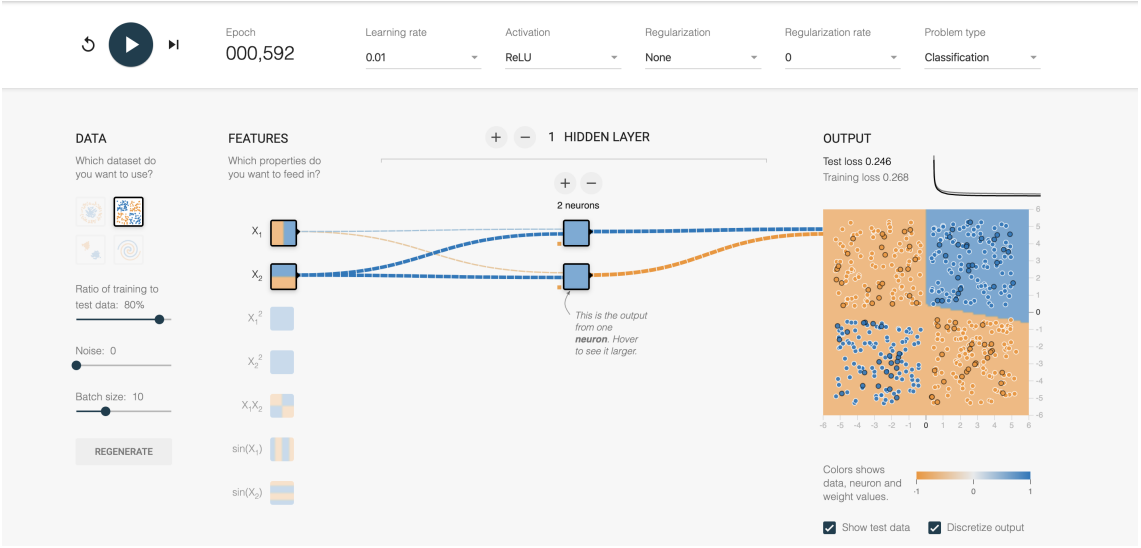
   (a) Can a linear classifier, without any hidden layers, solve the XOR problem?
   Solution: No. Since there's only one layer,it is only capable of distinguish all data with a line. It is apparently not possible to divide the data in XOR problem with a line.

[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

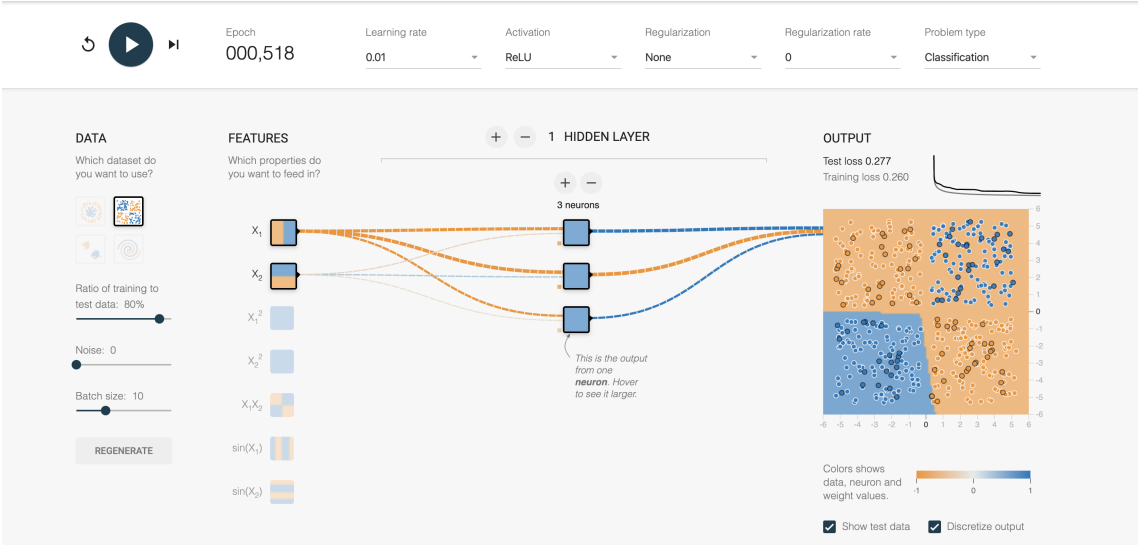
(b) With one hidden layer and $\text{ReLU}(x) = \max(0, x)$, how many neurons in the hidden layer do you need to solve the XOR problem? Describe the training loss and estimated prediction accuracy when using 2, 3 and 4 neurons. Discuss the intuition of why a certain number of neurons is necessary to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

Solution:

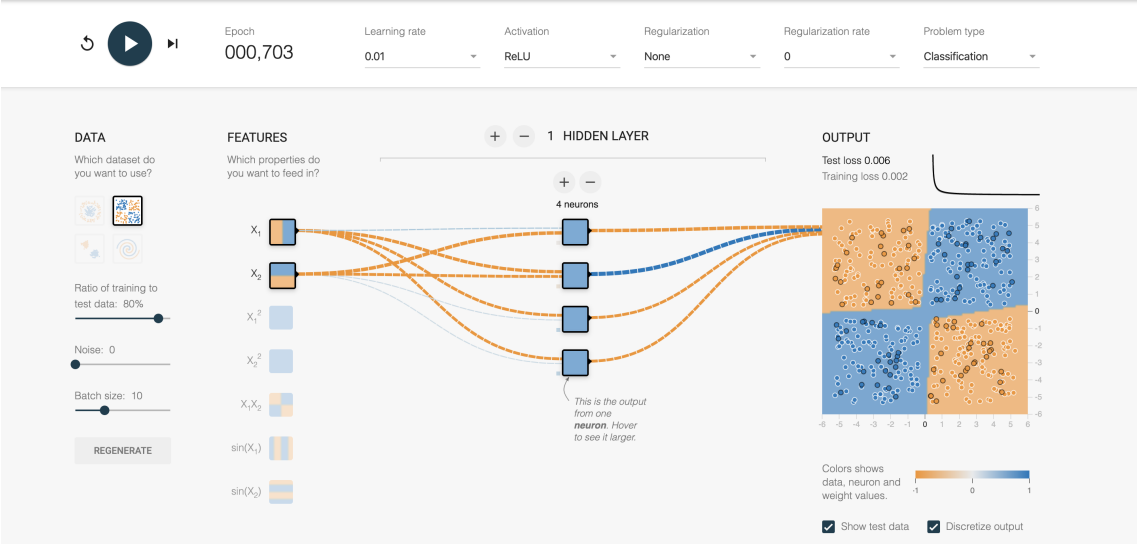
When using 2 neurons, the training loss is 0.268, the estimated prediction accuracy is $\frac{78}{100} = 0.78$. The picture is shown as below.



When using 3 neurons, the training loss is 0.260, the estimated prediction accuracy is $\frac{73}{100} = 0.73$. The picture is shown as below.



When using 4 neurons, the training loss is 0.002, the estimated prediction accuracy is $\frac{100}{100} = 1.00$. The picture is shown as below.



I think that there are 2 status for x1 and 2 status for x2. Since a layer of neurons can only perform 1 manipulation, we need $2 = 4$ neurons to represent the 4 conditions when x1 xor x2. Therefore, we can use the four neurons in the hidden layers to make to right prediction.

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.*

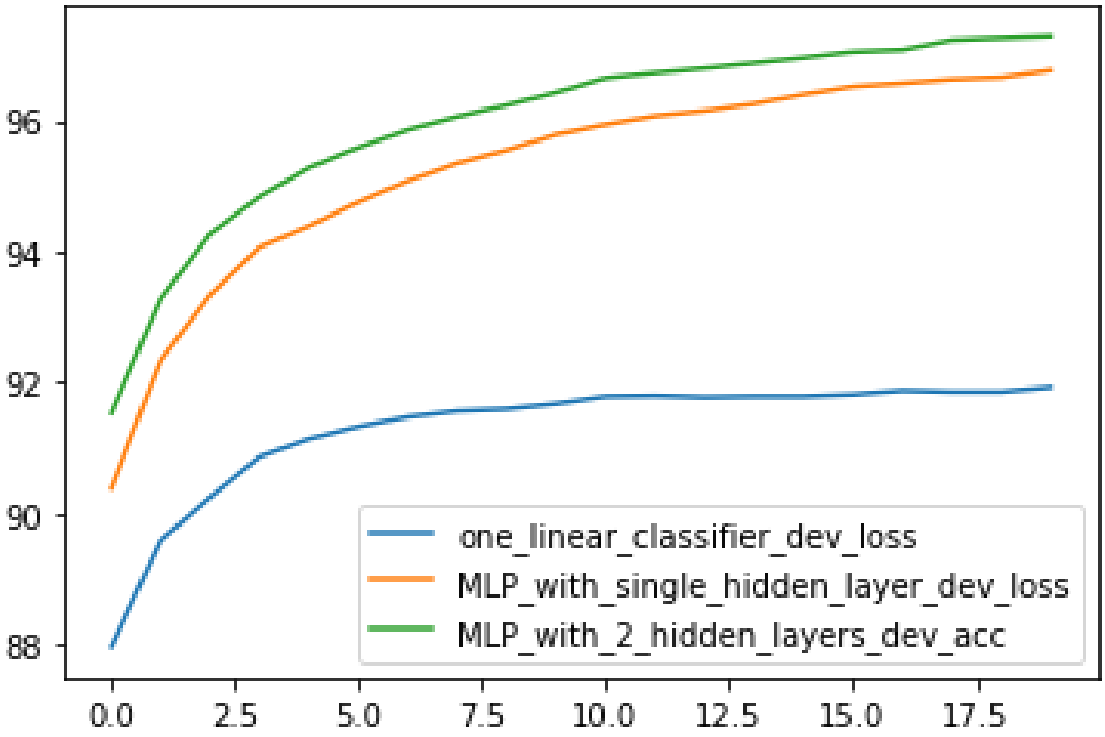
   (a) Implement the loss and gradient of a linear classifier (python function `linear_classifier_forward_and_backward`).

   (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`).

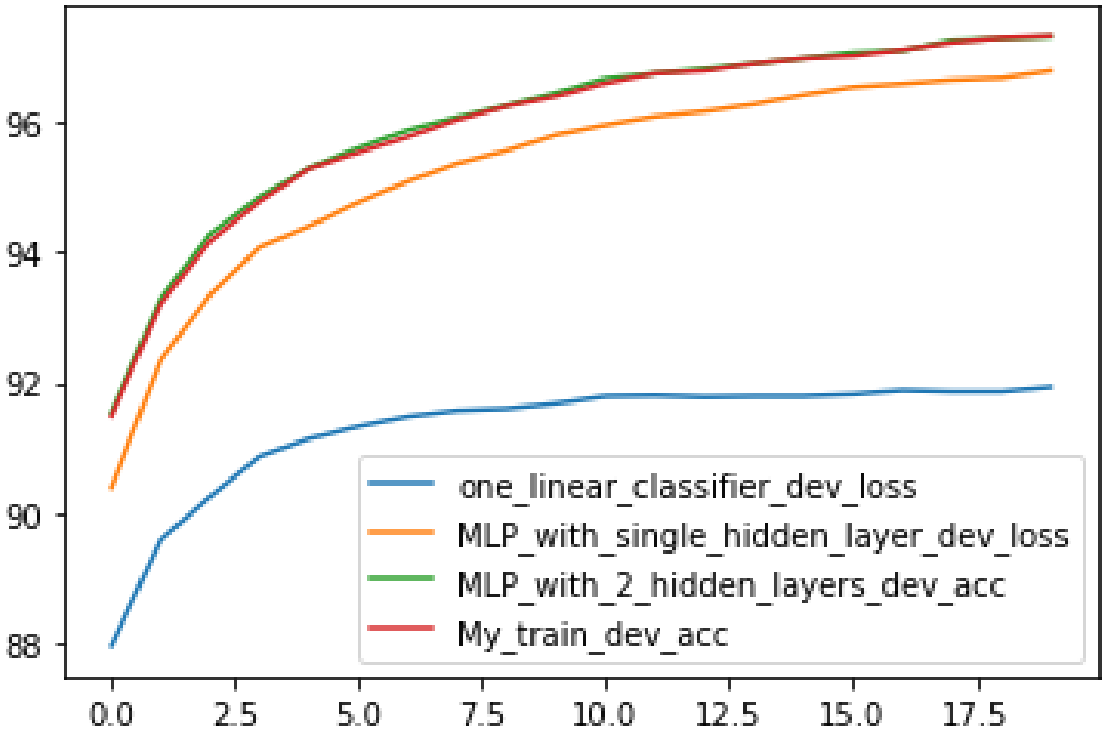
   (d) Plot the development accuracy of each epoch of three models in a single figure using the following hyperparameters: the batch size is 50, the learning rate is 0.005 and the number of epochs is 20.

   Solution:



(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.

Solution: The best parameter I currently find is BS = 100, LR = 0.01, NB_EPOCH = 20. The development accuracy is 97.30%, higher than the original MLP_with_2_hidden_layers_dev_loss, which is 97.29%.
The picture is shown as below:



The test accuracy is 97.18%