DS-UA 301
Advanced Topics in Data Science
Instructor: Parijat Dube
Due: March 3, 2024

Homework 3

General Instructions

This homework must be turned in on Gradescope by 11:59 pm on the due date. It must be your own work and your own work only—you must not copy anyone's work, or allow anyone to copy yours. This extends to writing code. You may consult with others, but when you write up, you must do so alone. Your homework submission must be written and submitted using Jupyter Notebook (.ipynb). **No handwritten solutions will be accepted**. You should submit:

1. One Jupyter Notebook containing all of your solutions in this homework.
2. One .pdf file generated from the notebook.

Please make sure your answers are clearly structured in the Jupyter Notebooks:

1. Label each question part clearly. Do not include written answers as code comments. The code used to obtain the answer for each question part should accompany the written answer.
2. All plots should include informative axis labels and legends. All codes should be accompanied by informative comments. All output of the code should be retained.
3. Math formulas can be typesetted in Markdown in the same way as LaTeX. A Markdown Guide is provided on Brightspace for reference.

For more homework-related policies, please refer to the syllabus.

Problem 1 - *Softmax Activation Function* 10 points

Consider the softmax activation function in the output layer, in which real-valued outputs $v_1, \ldots, v_k$ are converted into probabilities as follows:

$$o_i = \frac{\exp(v_i)}{\sum_{j=1}^{k} \exp(v_j)} \; \forall i \in \{1, \ldots, k\}.$$

1. Show that the value of $\frac{\partial o_i}{\partial v_j}$ is $o_i(1 - o_i)$ when $i = j$ and $-o_i o_j$ when $i \neq j$. **(5)**
2. Assume that we are using cross-entropy loss $L = -\sum_{i=1}^{k} y_i \log(o_i)$, where $y_i \in \{0, 1\}$ is the one-hot encoded class label over different values of $i \in \{1, \ldots, k\}$. Use the result in part 1 to show the correctness of the following equation: **(5)**
$$\frac{\partial L}{\partial v_i} = o_i - y_i.$$

Problem 2 - *Neural Network Training and Backpropagation* 25 points

In this problem, you will first review this notebook on training a 2-layered neural network (one input and one hidden layer) using sigmoid activations and backpropagation algorithm with and without regularized cost function. The notebook implements functions for forward propagation, cost calculation, and backpropagation.

Next, in the template provided, you will make a copy of this notebook and modify it to train a 3-layered neural network with two hidden layers using the same dataset. The number of hidden units in the first and second hidden layers is 20 and 20. The activation function you will use in hidden layers is scaled sigmoid, given by:

$$\hat{\sigma}(z) = \frac{1}{1+e^{-2z}}$$

You will need to make changes to the following functions in the original notebook.

1. `sigmoid()` to return scaled sigmoid. **(1)**
2. `forward_propagate()` to account for 2 hidden layers (the original has one hidden layer). **(3)**
3. `cost()` to calculate predictions from the 3-layered neural network and hence the cost. You need to make changes to both versions of the `cost()` function, with and without regularization, as in the original notebook. **(4)**
4. `sigmoid_gradient()` to return gradient of scaled sigmoid function. **(2)**
5. `backprop()` to compute the gradients. Your function should return both the cost and the gradient vector, as in the original notebook. Also, you will need to implement two versions of these functions, with and without regularization, as in the original notebook. **(8)**

   Then you will

6. Train your 3-layered neural network by minimizing the objective function, as in the original notebook, keeping the hyperparameters (`learning_rate, method, jac, options`) unchanged. **(3)**
7. Make forward predictions from your trained model and compute the accuracy. **(2)**
8. How does your model accuracy compare with the accuracy of the 2-layered neural network in the original notebook? **(2)**

Problem 3 - *Weight Initialization, Dead Neurons, Leaky ReLU* 25 points

Read the two blogs referenced below on weight initialization. You will reuse the code in the GitHub repo linked in the blog to explain vanishing and exploding gradients. You can use the same 5-layer neural network model as in the repo and the same dataset.

1. Explain the vanishing gradients phenomenon using *RandomNormal* initialization with three different values of standard deviation. You will conduct two groups of experiments: train the model with *tanh* and *sigmoid* activation functions. For each group of experiments, you should have one plot containing 3 subplots of gradients at each of the 5 layers in the neural network, which is similar to the plots in the blog post. Summerise and explain your observations. **(6)**
2. Next, show how *Xavier (aka Glorot normal) initialization* of weights helps in dealing with this problem. For each of the activation functions in the previous part, you should plot the gradients at each of the 5 layers in the neural network. Compare the plots with the previous part, and briefly discuss your observations. **(4)**
3. The dying ReLU is a kind of vanishing gradient, which refers to the problem when ReLU neurons become inactive and only output 0 for any input. In the worst case of dying ReLU, ReLU neurons at a certain layer are all dead, i.e., the entire network dies and is referred to as the dying ReLU neural network in Lu et al (reference below). A dying ReLU neural network collapses to a constant function.

Show this phenomenon using any one of the three 1-dimensional functions on page 13 of Lu et al. Use a ReLU network with 10 hidden layers, each of width 2 (hidden units per layer). Use a minibatch size of 64 and draw training data uniformly from $[-\sqrt{7}, \sqrt{7}]$. Perform 1000 independent training simulations each with 3,000 training points. Out of these 1000 simulations, what fraction resulted in neural network collapse? Is your answer close to over 90% as was reported in Lu et al.? **(10)**

4. Instead of ReLU consider Leaky ReLU activation as defined below:

$$\phi(z) = \begin{cases} z & \text{if } z > 0 \\ 0.01z & \text{if } z \leq 0. \end{cases}$$

Run the 1000 training simulations in the previous part with Leaky ReLU activation and keep everything else the same. Again calculate the fraction of simulations that resulted in neural network collapse. Did Leaky ReLU help in preventing dying neurons? If so, why do you think it helps? **(5)**

References:

- Andre Perunicic. Understand neural network weight initialization.
  Available at https://intoli.com/blog/neural-network-initialization/
- Daniel Godoy. Hyper-parameters in Action Part II — Weight Initializers.
  Available at https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404
- Initializers - Keras documentation. https://keras.io/initializers/.
- Lu Lu et al. Dying ReLU and Initialization: Theory and Numerical Examples.
  Available at https://arxiv.org/pdf/1903.06733.pdf.

Problem 4 - *Batch Normalization, Dropout, MNIST* 20 points

Batch normalization and Dropout are used as effective regularization techniques in training neural networks. However, it's unclear which one should be preferred and whether their benefits add up when used in conjunction. In this problem, we will compare batch normalization, dropout, and their conjunction using MNIST and LeNet-5 (see e.g., http://yann.lecun.com/exdb/lenet/). LeNet-5 is one of the earliest convolutional neural networks developed for image classification and its implementation in all major frameworks is available. You can refer to the lecture slides for the definition of standardization and batch normalization.

DS-UA 301
Advanced Topics in Data Science
Instructor: Parijat Dube
Due: March 3, 2024

Homework 3

1. Read two papers referenced below and explain the terms *co-adaptation* and *internal covariate shift*. Use examples if needed. **(2)**

2. Batch normalization is traditionally used in hidden layers, for the input layer, standard normalization is used. In standard normalization, the mean and standard deviation are calculated using the entire training dataset whereas in batch normalization these statistics are calculated for each mini-batch.

   Train LeNet-5 for 10 epochs with standard normalization of input and batch normalization for hidden layers. You may use SGD as the optimizer, and a batch size of 128. What are the learned batch normalization parameters for each layer? Plot the distribution of learned batch normalization parameters for each layer using violin plots. You should have one figure for each batch normalization parameter, and within each figure, you will have a violin plot for the distribution of that parameter on each layer. **(5)**

3. Next, instead of standard normalization, use batch normalization for the input layer as well. Train the network again with the same hyperparameters. Plot the distribution of learned batch norm parameters for each layer (including input) using violin plots similar to the previous part. However, now for each figure you should have one more violin plot compared with the previous part, which corresponds to the distribution of the batch normalization parameters on the input layer.

   Besides, compare the train/test accuracy and loss for the two cases, by plotting them over epochs. You should have one plot for accuracy, which includes two subplots: standard normalization for the input layer (the experiment in part 2) and batch normalization for the input layer (the experiment in part 3). For each subplot, you should have two line plots for training and testing accuracy as a function of epochs. For the sake of comparison, please make sure both subplots share the exact same scale on the y-axis. In a similar manner, you should have another plot (which includes two subplots) for training and testing loss. Briefly summarize your observations. Did batch normalization for the input layer improve performance? **(5)**

4. Using the same hyperparameters, train the network without batch normalization but this time use dropout. For hidden layers, use a dropout probability of 0.5 and for the input layer, take it to be 0.2. Compare train/test accuracy using dropout to the previous two experiments using batch normalization in parts 2 and 3 by adding a subplot in the previous accuracy plot. Again, please make sure all three subplots share the same scale on the y-axis. Briefly summarize your observations. Did dropout help improve performance? **(4)**

5. Now, still using the same set of hyperparameters, train the network using both batch normalization(including input layer) and dropout. How does the performance of the network compare with the cases with dropout alone (in part 4) and with batch normalization alone (in part 3)? You should have one accuracy plot similar to previous parts, but this time contains three subplots, each corresponding to experiments in parts 3, 4, and 5. Briefly summarize your observations. **(4)**

References:

- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R.Salakhutdinov . Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Available at at https://www.cs.toronto.edu/ rsalakhu/papers/srivastava14a.pdf.
- S. Ioffe, C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Available at https://arxiv.org/abs/1502.03167.

Homework 3

Problem 5 - *Learning Rate, Batch Size, FashionMNIST* 20 points

Recall the cyclical learning rate policy discussed in Lecture 4. The learning rate changes in a cyclical manner between lr_{min} and lr_{max}, which are hyperparameters that need to be specified. For this problem, you first need to read carefully the article referenced below as you will be making use of the code there (in Keras) and modifying it as needed. For those who want to work in Pytorch, there are open-source implementations of this policy available which you can easily search for and build over them. You will work with the FashionMNIST dataset and LeNet-5.

1. Fix batch size to 64 and start with 11 candidate learning rates: $10^{-9}, 10^{-8}, \ldots, 10^{1}$. Train your model for 5 epochs for each learning rate. Plot the training loss as a function of the learning rate. You should see a curve like Figure 2 in the referenced post below. Based on your plot, identify the values of lr_{min} and lr_{max}. **(4)**

2. Use the cyclical learning rate policy (with exponential decay) and train your network using batch size 64 and lr_{min} and lr_{max} values obtained in part 1. Plot train/validation loss and accuracy curve over the number of epochs (similar to Figure 4 in reference). **(8)**

3. We want to test if increasing batch size for a fixed learning rate has the same effect as decreasing learning rate for a fixed batch size. Fix the learning rate to lr_{max} and train your network starting with batch size 32 and incrementally going up to 4096 (each time by the power of 2; i.e. $2^5, 2^6, \ldots, 2^{12}$). You can choose a step size (in terms of the number of epochs) to increment the batch size. Plot the training loss vs. $\log_2(batch\_size)$. Is the generalization of your final model similar to or different from than cyclical learning rate policy? Briefly discuss your observations. **(8)**

References:

1. Leslie N. Smith Cyclical Learning Rates for Training Neural Networks.
   Available at https://arxiv.org/abs/1506.01186.

2. Keras implementation of cyclical learning rate policy.
   Available at https://www.pyimagesearch.com/2019/08/05/keras-learning-rate-finder/.