ECE 651 Fall 2024: Homework Assignment #2

Instructions:

1. Through this homework assignment, you will learn how to build a neural network step by step and train the model to make it work in test data. The training process involves the forward propagation, backward propagation, and gradient descent.
2. Through Problem 2, you will start to use a deep learning platform for training, preferably Pytorch.
3. Please submit **1 typed** report combining all your solutions and source files via Blackboard by Friday **October 4, 2024**. Late submissions of assignments will be handled according to the requirements specified in the Syllabus. Please **do not** submit zip files.
4. You will be graded by the quality of the source code and the report quality.

Problem 1. In this problem, we design a simple shallow neural network (NN) for image classification. A partial Python code is provided to help go through the coding for a simple NN step by step. There are only three training samples, with the input being the images of the three letters, A, B, C. Each image is a matrix of size 6×5, whose 30 entries are either 1 (being part of the letter image, indicated by yellow color in the example) or 0 (background, indicated by purple color in the example). Each image is vectorized into a length-30 vector. The output y for each input is a one-hot vector indicating one of the three letters.

Given a small training dataset, a simple 2-layer shallow neural network is adopted. Generally speaking, you will need to decide the size the of input and output layers of the neural network, based on the types of input and output data. You may design the size of the hidden layer, and tune it as a hyper-parameter. In the ipynb code provided, the hyper-parameters are:

1st layer: Input layer(1, 30)
2nd layer: Hidden layer (1, n_h) (use $n_h = 5$)
3rd layer: Output layer(1, 3)
The output Y collecting the labels of all three samples is of size (3, 3).

For simplicity, we train the NN using batch gradient descent, that is, all samples are used in every iteration of weight updating.

The activation functions for all layers are sigmoid. This is a design choice that can be changed.

The loss function in Step 5.4 is preferred to be the cross-entropy loss. For simplicity, we use the sum of squares loss, derive the gradient, and implement the corresponding back propagation formula for training.

Requirements

Fill in the code and make it runnable. Note that all the fillable spaces are marked by "None". You need to replace all "None" by appropriate terms to make the code runnable. The final output will be a sequence of cost as well as a plot of cost decreasing as the iteration increases.

Recommendations and bonus points

You may want to try out different design options and hyper-parameters based on your understanding of NN design considerations. Tunable hyper-parameters may include:

1. the number of neurons in the hidden layer
2. **the number of layers** (you may need to introduce new parameters in the code for added layers)
3. **activation functions for the hidden layers**
4. the learning rate in the gradient descent algorithm
5. **the loss/cost function**, such as using cross-entropy loss, adding a L_2-loss on the weights in the objective function for training, etc. You will need to re-derive the gradients during backprop.

If you try out one or more of the above options 2, 3, or 5 marked in bold, and provide comparative training performance, then you will receive up to 20% bonus points. Please concisely summarize your work and findings.

As a side note, the cost function for training a neural network does not have to be the same as the figure of merit for performance evaluation. For example, you can use the cross-entropy loss to train the network, but use the sum-of-squares loss or 0-1 loss as the error metric for evaluating the learning accuracy. Why and when should this happen?

Problem 2 Use a programming platform (i.e., Pytorch) to simplify the building of neural networks by abstracting away the low-level code. For example, you can build a neural network with the following structure:

```
DNN = models.Sequential([layers.Flatten(input_shape=(32,32,3)),
                         layers.Dense(3000, activation='relu'),
                         layers.Dense(1000, activation='relu'),
                         layers.Dense(10, activation='softmax')
                         ])
```

Try to adjust the input shape, the number of neurons, the number of layers, and the activation function to make it consistent with Problem 1. Train the model and compare with Problem 1. Summarize your observations regarding their performance and difference.