Project 2nd Module Instructions

Due March 24th 2023, before 11:59 pm

Submit your solutions electronically on gradescope as instructed.

General Instructions

- The project module directory structure is as follows:

```
Project_Module2
├── data
│   └── (data files)
├── centralized.py
├── distributed.py
├── multiSVM.py
├── plots
├── utils
│   ├── mnist_reader.py
│   └── plotter.py
└── logs
```

You will only need to make changes to the files `centralized.py`, `distributed.py`, `multiSVM.py` and the file `./utils/plotter.py` to visualize your logs; and create `centralized3.py` that is based on `centralized.py`. Please do not change the location of any files and maintain the same directory structure. Your generated logs from running the code will be stored in the `logs` folder and generated plots (after running `./utils/plotter.py`) would be stored in the `plots` folder. You do not have to write commands to load any data as that has already been done in the code files.

- We will also make use of the following python libraries: `scipy, scikit-learn, matplotlib`, which can be installed using:

```
pip install -U scikit-learn scipy matplotlib
```

The code for loading and using these libraries has already been provided in the main file.

- **Plotting:** The output of your code would be stored as log files (of loss, testing accuracy, etc.) in the `logs` folder. The name given to each file will contain the hyper-parameter configurations used for running the code, so different logs files are easy to distinguish. Using the name for these log files, you may run the `plotter.py` file in the `utils` folder to visualize these logs. The plots will be stored in the `plots` directory. Please ensure that you provide the appropriate configuration (log names) in the `plotter.py` file to generate the appropriate plots. For plots comparing different schemes (say, learning rates), you must provide the curves in the same plot. You may find additional instructions on what you need to change in the `plotter.py` file.

- You must upload a pdf of your report to Gradescope by **Friday, 24th March 11:59 p.m.**

- **Submission Format:** Create a PDF report containing any comments and plots supporting your answers. You must also embed your completed code files (`centralized.py`, `centralized3.py`, `distributed.py` and `multiSVM.py`) to your report. Please be sure to include the name of all the group members on the first page of your report and include a small section towards the end on the contribution of each group member in the project.

Introduction to Neural Networks

We introduce neural networks by first introducing the simplest component of these networks, *the artificial neuron.* An artificial neuron takes several inputs, $x_1, \ldots, x_n$, multiplies the inputs by weights $w_1, \ldots, w_n$, adds a bias b to the weighted sum, and outputs a single binary output based on an activation function f:

$$y = f\left(\sum_j w_j x_j\right) \tag{1}$$

An example for f would be the rectifier function where

$$y = \left(\sum_j w_j x_j\right)^+ = \max\left\{0, \sum_j w_j x_j\right\} \tag{2}$$

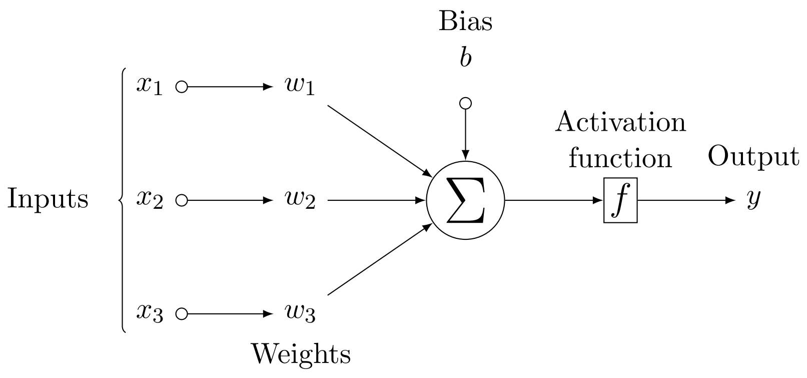

Figure 1: Artificial Neuron

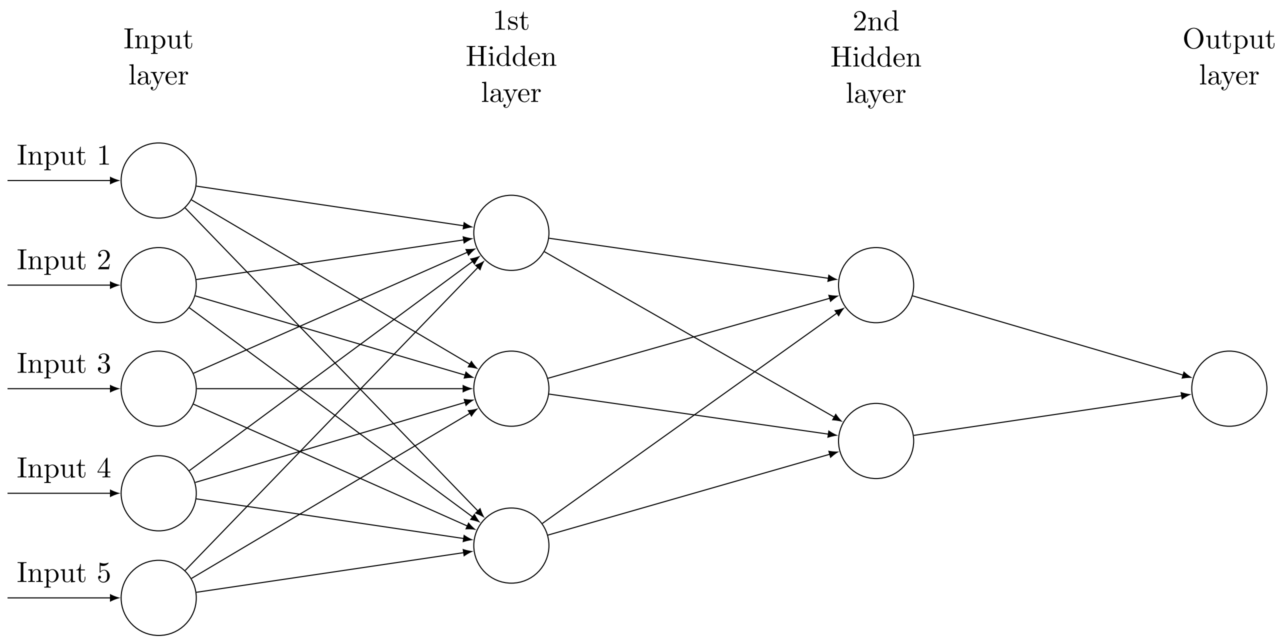

Figure 2: Neural Network with two hidden layers

In the below network, the first column of neurons, we have the input layer. The second column of neurons, the first hidden layer, is making three simple decisions, by weighing the inputs. Each of the neurons in the second hidden layer is making a decision by weighing up the results from the first layer of decision-making. In this way a neuron in the second layer can make a decision at a more complex and more abstract level than neurons in the first layer. In this way, a many-layer network of perceptrons can engage in sophisticated decision making. *Note that the output layer can have more than one neuron, e.g. in multiclass classification.*

Gradient Descent and Backpropagation

The goal is to learn the weights and biases that minimize a training loss function. The idea is to use the backpropagation algorithm to find the gradients of the loss function with respect to the neural network weights and biases. The loss minimization can then be performed using gradient based methods. Please refer the resource [1] that uses terminology that you became familiar with throughout the course, and provides an accessible description of the backpropagation algorithm.

1 Centralized Neural Network training

In this exercise, you will work through multi-class classification using neural networks. Our data consists of inputs $x_i \in \mathbb{R}^d$ and labels $y_i \in \{0, 1, \ldots, 9\}$ for $i \in \{1, 2, \ldots, N\}$. We will work with the Fashion-MNIST dataset, which you must be familiar with from Module 1.
You are provided `centralized.py` which contains a complete, functioning code for training a

centralized neural network with 1 hidden layer and 10 output nodes (for each class). As such, the goal of this problem is to make you familiar with the training code. Given $x_i \in \mathbb{R}^d$, the neural network outputs $y = [y_{i,0}, y_{i,1}, \ldots, y_{i,9}]^T$ where $y_{i,k}$ is the probability that sample x_i is from class $k \in \{0, \ldots, 9\}$. Hence the decision rule for x_i is given by:

$$\hat{y}_i = \text{argmax}_{k \in \{0,\ldots,9\}} y_{i,k}$$

Finally, we define the cross entropy loss and the accuracy over the training set as:

$$L_{CE}^{\text{train}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=0}^{9} \mathbb{1}_{y_i=k} \log(y_{i,k})$$

$$A^{\text{train}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\hat{y}_i=y_i}$$

with similar definition for L_{CE}^{test} and A^{test} on the test dataset.

(a) Run the code with the given hyper-parameters. Include the output of the code in the final report.

(b) Try out different values (individually) for the number of nodes in the hidden layer, the learning rate η, the weight decay factor λ, and the batch size. Comment on your general observations from these experiments. You may include any supporting plots regarding the loss/test accuracy in the report. Make sure to report maximum test accuracy and minimum training loss you have seen over all iterations.

(c) Modify the gradient descent updates to include momentum for every layer. For instance, for the first layer weights (similarly for bias) change:

`params["W1"] = params["W1"] - learning_rate * params_grads["dW1"]` to the following two lines:

```
v_w1 = momentum_coef*v_w1 + params_grads["dW1"]
params["W1"] = params["W1"] - learning_rate * v_w1
```

where momentum coefficient is chosen as 0.1. You might need to do other changes in the code such as initializing v_w1 as 0 and defining momentum_coef as an hyperparameter. How does your result compares to the case without momentum, in terms of maximum test accuracy and lowest training error?

(d) Create a copy of `centralized.py` and call it `centralized3.py`. In this part you will extend the neural network in parts a,b with one hidden layer to a neural network with two hidden layers (you will add another set of parameters W_3, b_3). Note that now the activation function after second layer will be ReLU instead of softmax, and you will use softmax after the third layer. Fix the best set of parameters for the first hidden layer, η, λ and batch size from the previous part, and try out different values for hidden layer 2. Comment on your general observations from these experiments. You may include any supporting plots regarding the loss/test accuracy in the report. Make sure to report maximum test accuracy and minimum training loss you have seen over all iterations. Compare the maximum test accuracy and minimum training loss you have observed to the ones in the previous part.

(e) We will now compare neural network training with a multi-class SVM training. A complete and functioning code implementing the Kernel-SVM algorithm using the *Radial Basis Function* kernel has been provided in the `multiSVM.py` file. Experiment with different values of the regularization coefficient in the code, differing in orders of magnitude. What do you observe works best? Compare the test accuracy to that of neural network training.
Note: This part requires using the SVM implementation from the SKLEARN *library. You may read more about the method used in [2].*
Note 2: This part may take some time.

2 Distributed Neural Network training (Federated learning)

Mobile devices have access to large datasets that are suitable for learning models which can improve the user experience on the device. However, this data is often privacy sensitive or large in size, which makes aggregating the data to process at a central system infeasible. An alternative would be to leave the training data on the mobile devices and instead learn a shared model by aggregating locally-computed updates (in our case, gradient updates). This training scenario is termed as *Federated learning*.
In this exercise, you will work through multi-class classification using neural networks in a federated setting. Specifically, we will consider that the original dataset is distributed among some number of worker nodes (specified as `num_workers` in the file `distributed.py`). At each training iteration, we will compute the gradient for each node locally (that is, using just the node's own local data). The gradient updates from all the client are then averaged, and this average gradient vector is made known to each client. The nodes then take a step along the direction of this average. Note that this is different from the case where nodes use gradients updates formed only by their local datasets. In our case, the nodes are working *collaboratively* to learn a single model.You may also refer to Algorithm 1 in [3] for details. For this exercise use the one hidden layer neural network without momentum.

(a) Complete the method `uniformSampling( )` in distributed.py to distribute the training samples over the clients (workers) uniformly at random.

(b) Complete `trainDistributed()` method in distributed.py to train the neural network in a distributed setting.

(c) Try out different values for the hyper-parameters (batch size, learning rate, epochs, learning rate decay) including the number of clients. Comment on your observations and include any relevant plots in your answer.

References

[1] Neural Networks and Deep Learning (Chapter2) http://neuralnetworksanddeeplearning.com/chap2.html

[2] sklearn - Support Vector Classification https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[3] McMahan, B., Moore, E., Ramage, D., Hampson, S. and y Arcas, B.A., 2017, April. Communication-efficient learning of deep networks from decentralized data. In Artificial intelligence and statistics (pp. 1273-1282). PMLR.