CSE343/ ECE363/ ECE563: Machine Learning W2022
Assignment: Neural Networks & CNN

Max Marks: Programming: 105 (+15 Extra credit), Theory: 5(UG/PG) + 10 (for PG only)

Due Date: 3/5/2022, 11:59 PM

Instructions

- This is a individual assignment.
- Try to attempt all questions.
- The theory questions should be your individual effort. Copying/Plagiarism will be dealt with strictly.
- Start early, solve the problems yourself.
- Late submission penalty: Refer policy on course website.
- Your submission would be a single .zip file (rollno_HW1.zip) file, that would contain two items (codes + .pdf file). Zip your saved models and include them in the codes folder. You have to include all your plots, results, analysis, conclusion, and solutions for the theory questions in the pdf file. **Code should be well documented. Use comments for python scripts or markdown in Jupyter Notebooks.**
- Anything not written in the report will fetch 0 marks.
- It is preferred that you write LaTeX reports.
- Remember to **Turn in** after uploading on Google Classroom. No excuses or issues would be taken regarding this after the deadline.
- Start the assignment early. Resolve all your doubts from TAs in their office hours **at least two days before the deadline.**

PROGRAMMING QUESTIONS

1. (70 points) **Backpropagation - To be attempted individually**
   You have to implement a general algorithm for training Neural Networks using only the Numpy library. Use Q1.py for implementing the algorithm.

   1. (2) The network should have the following parameters
      - **n_layers:** Number of Layers (int)
      - **layer_sizes:** array of size n_layers which contains the number of nodes in each layer. (array of int)

- **activation:** activation function to be used (string)
- **learning_rate:** the learning rate to be used (float)
- **weight_init:** initialization function to be used
- **batch_size:** batch size to be used (int)
- **num_epochs:** number of epochs to be used for training (int)

Rubric:

- 2 points if all parameters used and initialized with appropriate values.

2. (2+2+2+2+2 = 10) Implement the following activation functions with their gradient calculation too: **ReLU, Sigmoid, Linear, Tanh, Softmax**.
   Rubric:
   - 2 points for each correct implementation.

3. (1+1+1 = 3) Implement the following weight initialization techniques for the hidden layers:
   - **zero:** Zero initialization
   - **random:** Random initialization with a scaling factor of 0.01
   - **normal:** Normal(0,1) initialization with a scaling factor of 0.01

   Rubric:
   - 1 point for each correct implemenation.

4. (9+2+2+2 = 15) Implement the following functions with zero bias for each neuron and **cross entropy loss** as the loss function. You can create other helper functions too.
   - **fit():** accepts input data & input labels and trains a new model.
   - **predict_proba():** accepts input data and returns class wise probability. Hint: Use softmax.
   - **predict():** accepts input data and returns the prediction using the trained model.
   - **score():** accepts input data and their labels and returns accuracy of the model on the corresponding input data and labels.

   Rubric:
   - 5 points for gradient calculation implemented correctly using cross entropy loss function at the final layer
   - 4 points for correctly updating weights and biases using gradient descent.
   - 2 point for each correctly implemented function.

   If the model has failed to learn anything, i.e. the accuracy stays at random or 10%, then 0 points for the fit function. Marks have still been given for every model trained and comparing with scikit-learn, even though its no use comparing untrained models.

5. (28) Use the FASHION MNIST dataset for training and testing the neural network model created above.

(a) (16)Use the following architecture, training method, and setup to train your network:

- **Architecture:** #input, 256, 128, 64, #output
- **Learning Rate:** 0.1
- **Number of Epochs:** 100
- **Batch Size:** 32
- Use **normal** weight initialization
- Train using **stochastic gradient descent** and **random shuffle** on each iteration of SGD
- Dataset should be 80% training, 10% validation, and 10% testing

Keep the following deliverables in mind:

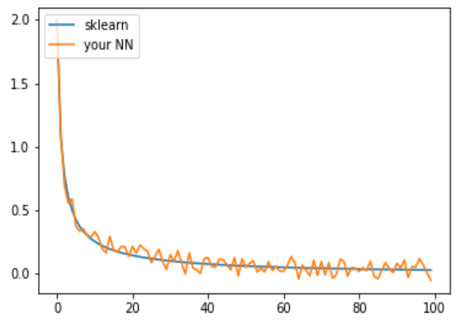
- (2*4=8) Train your neural network using all four activation functions: **ReLU, Sigmoid, Linear, Tanh**. Calculate training and validation loss after each iteration and save them to plot graphs later. 1 marks for validation and training loss for each of 4 activation functions.
- (0.5*16 = 8) Save the learnt weights and biases using Pickle or Joblib after every 50 epochs for all the four activation functions(So, there should be 2 files - one for weights and one for biases, for 2 instances - after 50 epochs and after 100 epochs for four activation functions, which makes a total of 16 files) Use the following naming convention for model files : **(iteration)_(Activation Function)_(weights/biases).pkl** Rubric: 0.5 if each file exists, ask student to open and print any 1 pickle file.

(b) ((0.5+0.5)*4=4) Plot training loss vs epoch curve and validation loss vs epoch curve for ReLU and sigmoid activation functions. all graphs should be decreasing exponential graphs.

(c) (2*4=8) Test your saved models on the test set for all activation functions. Compare accuracies after 50 and 100 iterations. models should be successfully loaded and realistic values should be generated.

6. (12) Implement the same architecture with the same hyperparameters using sklearn's MLP classifier (You may reduce the learning rate). Use sklearn's fit, predict and score methods to get the accuracy on test data. You can use the **"compare_with_mlp"** function provided in Q1.py as reference (it provides code for training sklearn MLP and ploting loss curve):

   - Use Random normal weight initialization and save the model at the 50th and 100th iteration.
   - (1*8 = 8)For each of these activation functions (ReLU, Sigmoid, Linear, Tanh) plot the loss of your model (cross-entropy loss) with that of Scikit-Learn's MLP using the compare_with_mlp method in the same graph. You should have 1 graph for each of the 4 activation functions, with 2 plots per graph. A sample graph is provided below :
     The loss should look reasonable (not too smooth) and should match that of



sklearn

- (4) Load models after 50th and 100th iterations (yours and scikit-learn's) from saved weights and biases for all activation functions and do the following:
  - (2) Obtain accuracy of all loaded models on the test set.
  - (2) Report the best model overall in terms of testing accuracy.

  Accuracy should not be abrupt, look for realistic results.

2. (35 points) **Convolutional Neural Networks** Note: You are only allowed to use the PyTorch framework in Python. Explicitly mention each parameter considered (Number of filters, filter size, type of pooling).

   - For this problem, use the existing VGG16 model in Pytorch (pretrained on ImageNet) on the FASHION MNIST dataset and fine tune it using the following steps:-
     - Divide the training batch of the dataset into train set and validation set (80:20 split)
     - Freeze all the weights of the lower convolutional layers.
     - Replace the upper dense layers of the network with custom fully connected layers of your own. Please note that the number of outputs of your model would be equal to the number of classes of the dataset.
     - Train only the custom added layers of this model using the training set, therefore, optimizing the VGG16 model for your dataset.
   - Perform the following analysis:
     1. (10) Plot train and validation loss and accuracy w.r.t number of epochs and state your observations. Use Wandb library for the same. Justify the choice of preprocessing and hyperparameters used, if any.
     2. (15) Report the accuracy and confusion matrix on the test set using the fine tuned model.
     3. (10) Report class wise accuracy and confusion matrices and analyse your findings.
        - 10 points for correct implementation of model. Dividing into train test + Freeze weights + Replace with custom + Training only custom - (1+3+3+3)
        - Loss should be decreasing and while accuracy increasing for both train and validation set with increasing number of epochs. 10 points for converging loss and accuracy plots.

  - 5 points for accuracy and confusion matrix on test set (2.5 each).
  - 7 points for class-wise accuracy and confusion matrices (3.5 each) and 3 points for analysis on which classes are identified more accurately than others and why.

3. (15 points) (*Extra Credit*) After training the VGG16 on the Fashion MNIST dataset. Use that same model to visualize it's Convolution Layers. For that clone this GitHub repository and Perform the following tasks:-

   1. (5) Use the *cnn_visualization.py* file to visualize all the classes in Fashion MNIST dataset and state your observations in the report along the visualizations generated, any layer can be used to visualize.
   2. (5) Use the *vanilla_backprop.py* file to visualize all the classes in Fashion MNIST dataset and state your observations in the report along the visualizations generated, any layer can be used to visualize.
   3. (5) Use the *cnn_visualization.py* file to visualize a randomly generated image(Array of random integers) and show visualizations for all the layers and state your observations.

   Note:-

   - Parameters filter_pos and cnn_layer can be changed.
   - Any function with or without PyTorch hooks can be used.
   - Number of iterations can also be tweaked to get a more meaningful visualizations.

   - 5 points for each correct output generation and analysis.
   - partial marking done if analysis incorrect or complete output (e.g. each layer visualization) not generated.

THEORY QUESTIONS

1. (5 points) The KL divergence between two discrete probability distributions P and Q is given by $KL(P||Q) = \sum_z P(z) \log \frac{P(z)}{Q(z)}$. Show that the cross-entropy loss used for multi-class classification is the same as the KL divergence under certain assumptions of the posterior distribution $P(y|x)$, where y are the class labels, while x are the data samples. What are these assumptions?

   Let $P(z)$ be the probability of the target variable and $Q(z)$ be the probability of the predicted variable. The cross-entropy loss and KL divergence loss for distributions P and Q can be written as:

$$CE = -\sum_z P(z) \log Q(z) \tag{1}$$

$$KL(P||Q) = \sum_z P(z) \log P(z) - \sum_z P(z) \log Q(z) \tag{2}$$

   Substituting Equation (1) in (2),

$$KL(P||Q) = \sum_z P(z) \log P(z) + CE$$

   Entropy is the randomness of the information being processed, given as: $\sum_z P(z) \log P(z)$. Assuming that we have sufficiently large dataset ($\mathcal{D}$) that can reflect the actual distribution of the problem ($\mathcal{P}$), i.e. $P(\mathcal{D}) = P(\mathcal{P})$, we can say that entropy of P is constant.

$$\therefore KL(P||Q) = CE + \text{Constant}$$

   Thus, we can say that minimizing KL divergence is same as minimizing cross-entropy loss and therefore they are same.

2. (10 points) A common problem while building CNN or Convolutional Neural Networks is the selection of an appropriate kernel(or filter) size. Larger kernels tend to focus more on globally distributed information, while smaller ones focus on local information more. Inception Net was a great invention in this regard. It solved this problem by applying multiple kernels at the same network depth, and stacking the output feature maps. One thing that stood out was the usage of 1x1 convolutions to reduce the feature map size before applying a larger kernel, which led to an overall reduction in the number of computations.
   Given the 2 networks below, calculate and compare the total number of computations performed to calculate the final feature map. Also calculate the size of the feature map after every convolution operation.
   Network 1:

   - Input : Feature map of size 28 x 28 x 192
   - Convolution operation : Filter size 5 x 5, same padding, 32 filters

   Network 2:

- Input : Feature map of size 28 x 28 x 192
- Convolution operation : Filter size 1 x 1, no padding, 16 filters.
- Convolution operation : Filter size 5 x 5, same padding, 32 filters.

Rubric:
For network 1:

- Computations = (5x5x192)x(28x28x32) = 120422400
- Close to 120 million computations
- Assuming stride=1, output size = (28-5+2)+1 = 26x26x32

For network 2:

- Computations for first convolution = (1x1x192)x(28x28x16) = 2408448
- Output size after first convolution = (28-1+0)+1 = 28*28*16
- Computations for second convolution = (5x5x16)x(28x28x32) = 10035200
- total $\approx$ 12 million parameters
- output size = (28-5+2)+1 = 26x26x32

Thus adding a 1x1 convolution layer reduces the total number of computations by 10 fold, even though output size is the same.