Neural Networks and Convolutional Neural Networks

COL 774: Assignment 2

Released on: 31st August, 2024

Due Date [Part 1]: 7.00PM, 7th September, 2024 [Saturday]

-
- This assignment has two main parts - 1. Neural Network and 2. Convolutional Neural Networks.
- All the announcements related to the assignment will be made on Piazza. The access code is wbd8avkblhb. You are strongly advised to have a look at the relevant Piazza post regularly.
- You are advised to use vector operations (wherever possible) for best performance as the evaluation will be timed.
- You should use Python 3 for all your programming solutions.
- Your assignments will be auto-graded, make sure you test your programs before submitting. We will use your code to train the model on the train set and predict on the test set.
- Input/output format, submission format and other details are included. Your programs should be modular enough to accept specified parameters.
- You may submit the assignment **individually or in groups of 2**. No additional resource or library is allowed except for the ones specifically mentioned in the assignment statement. If you still wish to use some other library, please consult on Piazza. If you choose to use any external resource or copy any part of this assignment, you will be awarded D or F grade or **DISCO**.
- You should submit work of your own. You should cite the source in the report, if you choose to use any external resource for the competitive part i.e. Part 1 Section (c) and (d). You will be awarded D or F grade or **DISCO** in case of plagiarism.
- **Late submission policy**: For each late day from the deadline, there will be a penalty of 7%. So, if you submit, say, 4 days after the deadline, you will be graded out of 72% of the weightage of the assignment. Any submission made after 7 days would see a fixed penalty of 50%.
- For doubts, use Piazza. Send an email to Kaustubh R Borgavi for doubts that may disclose implementation details. You are also heavily encouraged to report your best objective function score for Part 1 Section (C) and (D) through mailing.

History of updates

- **(03/09/2024): Xavier Initialisation:** Updated the suggested `numpy` call for Xavier initialisation to include the variance term of `2/n_i`. Check Part 1.a: Initialisation of weights section for more details.
- **(03/09/2024): Saving weights.pkl:** Corrected the names of the keys mentioned for saving the `bias` terms for `weights.pkl`. Check `Coding Guidelines` for more details.

1 Neural Networks (50 points, Due date:)

Problem Statement: In this part of the assignment, you will be implementing a Neural Network for the task of Indian language classification. For this task we will be using a dataset which contains cropped images of handwritten words written in 8 different Indian languages. You will be designing a neural network for classifying the images into their respective language classes. Some examples of the images from the dataset are shown in Fig. 9. You will be using the train set images [400 images/class] to train the model and the validation set images [100 images/class] to check the performance of your model. We will be evaluating the weights of your Neural Network for Part (a) and (b) by comparing them with the given weights for grading. You can use the given checker script here to check if your weights are matching for these two parts. Part (c) is a competitive part where the loss on the training set will be used for grading. The lower the loss obtained, the higher the marks you get. Lastly, Part (d) is also competitive where the Accuracy obtained using your Neural network on the held-out test set will be used for grading. You will have to design the Neural network to obtain the best accuracy for this part.

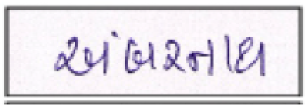
Figure 1: Gujarati

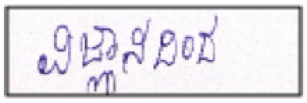
Figure 2: Kannada

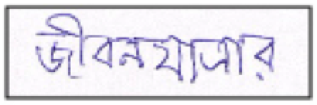
Figure 3: Bengali

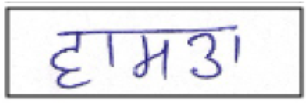
Figure 4: Gurmukhi

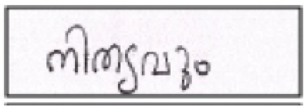
Figure 5: Malayalam

Figure 6: Odia

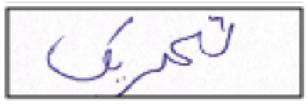
Figure 7: Urdu

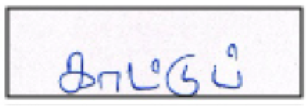
Figure 8: Tamil

Figure 9: Glimpse at the dataset

(a) **Training a neural network for binary classification [12.5 points]**: In this part, you will be designing a Neural Network for the task of binary classification of two languages. The `Binary dataset` shared here contains the languages: *Bengali* and *Kannada*. You will be implementing the forward and the backward propagation for the the given Neural Network for the task of binary classification and checking if the weights match with the given weights using the checker script. You are allowed to use only `numpy` and usage of any DL frameworks such as `TensorFlow` or `PyTorch` is NOT allowed for this part of the assignment.

   i. **Data preprocessing**: You are provided with a script `preprocessing.py` to load the data from the given dataset for feeding it to your model. Place the script in the same directory where the dataset is present in your systems. You are NOT supposed to make any changes to the given data loading and preprocessing pipeline as the weights may not agree with the given weights otherwise.

   ii. **Neural Net Design:** Consider the neural network architecture as shown in the Appendix section for the task of binary classification. Implement the given neural network design and the back-propogation algorithm to train the network from scratch using `numpy` only. As shown in the diagram, use `Sigmoid` as the activation function for the units in **intermediate layers** and also in final **output layer**. As mentioned, using DL frameworks such as `PyTorch` or `TensorFLow` for implementating the Neural Networks is **NOT** allowed for this part of the Assignment.

   iii. **Initialization of Weights** : We will be using a standard way to initialise weights known as the **Xavier initialization** in all parts of this assignment. You must use `numpy.random.randn(`n_i`,` n_o`) * np.sqrt(2/(`n_i`))` to initialise the weights where n_i is the number of input neurons and n_o is the number of output neurons. Ensure that you initialise the weights only once before the beginning of the training of the Neural network. You must use `numpy.random.seed(0)` to set the seed for the weights to align after training. The code to set the seed should only run once for the entire training phase. The biases for each layer should be initialised to zeros. You must use `np.zeros` to do so. Ensure that the weights and biases when initialised have the data type of `float64`.

iv. **Optimiser Design:** Implement the Mini-batch gradient descent optimiser using `numpy` only for training the given neural network. Use `batch size` as 256. Refer the Appendix section for the list of hyperparameters to be used for training. For this section, you are supposed to use the same set of hyperparameters and the same NN design as mentioned in the Appendix.

v. **Loss Function:** Implement the binary cross entropy loss [logistic loss] for training the network for the task of two-class classification. The details of the binary cross entropy loss function are given below:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] \tag{1}$$

Here i corresponds the training samples. N is the number of samples in the batch. Here, y_i is the ground binary value and $\hat{y}_i$ is the prediction of the network post `sigmoid` activation such that $\hat{y}_i \in [0, +1]$.

vi. **Evaluation:** Once the training is completed [Check all hyperparameters in Appendix for training], use the checker script from here to ensure if your weights are matching with the given weights.

(b) **Training a neural network for multi-class classification [12.5 points]**: In this part, you will be designing a Neural Network for the task of multi-class classification. The `Multi-class dataset` shared here contains 8 Indian language images. You will be implementing the forward and the backward propagation for the the given Neural Network for the task of 8-class classification and checking if the weights match with the given weights using the checker script. You are allowed to use only `numpy` and usage of any DL frameworks such as `TensorFlow` or `PyTorch` is NOT allowed for implementing the Neural Networks and training them.

i. **Data preprocessing**: You are provided with a script `preprocessing.py` to load the preprocessed data from the given dataset to feed it to your model. Place the script in the same directory where the dataset is present in your systems. You are NOT supposed to make any changes to the given data loading and preprocessing pipeline as the weights may not agree with the given weights otherwise.

ii. **Neural Net Design:** Consider the neural network architecture as shown in the Appendix section for the task of 8-class classification. Implement the given neural network design and the back-propogation algorithm to train the network from scratch using `numpy` only. As shown in the diagram, Use `Sigmoid` as the activation function for the units in **intermediate layers** and `Softmax` as the activation in final **output layer**. Use the same `numpy.random.randn` function to initialise the weights and `np.zeros` to initialise the biases as done in Part-(a). As mentioned, using DL frameworks such as `PyTorch` or `TensorFLow` for implementating the Neural Networks is **NOT** allowed for this part of the Assignment.

iii. **Loss Function:** For this section, we will be using the cross entropy loss function as shown below:

$$CE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}\log(\hat{y}_{ij}) \tag{2}$$

$$\hat{y}_{ij} = \frac{\exp z_{ij}}{\sum_{j'=1}^{k}\exp z_{ij'}} \tag{3}$$

Here i corresponds the training samples while j corresponds to their corresponding class labels. n is the number of samples in the batch and k is the number of labels/classes being used. Here, y_{ij} is the ground truth one-hot encoded vector and will have one value (corresponding to true class index) as **+1** and **0** for all other values (corresponding to other class indices). $\hat{y}_{ij}$ is the probability vector from the network post `softmax` activation such that $\hat{y}_{ij} \in [0, +1] : \sum_j \hat{y}_{ij} = 1$. z_{ij} is the logit of the j^{th} class from the final layer before the `softmax` activation for the i^{th} sample.

iv. **Evaluation:** Once the training is completed [Check all hyperparameters in Appendix for training], use the checker script from here to ensure if your weights are matching with the given weights.

(c) **Choosing the best optimiser [12.5 points]:** In this part, the objective is to minimize the training loss of the given Neural Network [check Appendix section] within 15 minutes of runtime for 8-class classification. This part is competitive and the training loss will be used for grading. You can explore some of the suggestions given below for possible ways to reduce the loss in least number of iterations:

- Momentum
- RMSprop
- Adam

For this part, refer to the Neural network design given for for Part (c) in the Appendix section. For this part, you can start with using the standard values for the hyperparameters for the chosen optimiser(as specified on the web) and later tune them in order to get the best performance. After the implementation, experiment which training setting helps in converging faster. You are free to tune the hyperparameters to achieve the lowest possible loss value within 15 minutes by maintaining the following attributes same:

- Multi-class architecture (As specified in the Appendix section).
- `Softmax` activation in the output layer.
- Cross-entropy loss.

Report the best setting you get(the best is the one that reduces the loss to a minimum in the given time constraint) on the train set. You can also refer to the checker script given for Part (c) to check the loss of your model on the train set. While evaluation, we will be using the train set to compute the loss value using the last epoch saved weights that we obtain after the training of your model. Mention in the report how did you arrive at these parameters with sufficient justification. Once you achieve the best setting, fix the number of epochs required to achieve the least loss possible in the submission script so that the training concludes at that point. Make sure the execution of the script does not exceed 15 minutes and overwrites the weights after each epoch. At the end of training or at the of 15 minutes (whichever is sooner) we will be using the saved weights of the last epoch to compute the loss value on the train set. Use the `time` library in `Python 3` to measure the time of your executions[Refer coding guidelines and submission instructions for more details]. Please note that this is a competitive part, and your code would not be allowed to run for more than 15 minutes. The format to save the weights is specified in the `Coding guidelines` section. The grading for this Part is relative among all students based on the loss value obtained on the train set.

(d) **Designing NN for best accuracy [12.5 points]:** In this part, the objective is to design a NN for 8-class classification which can give the best accuracy on the held-out test set within 15 minutes of runtime. This part is also competitive and the grading will be relative based on the accuracy achieved on the held-out test set. For the given `Multi-class dataset` used in Part (b) and (c), you can experiment with different Neural network designs such as experimenting by varying the number of hidden layers, try different number of units in each layer with different loss functions, activation functions and gradient descent variants etc. Comment on your observation and submit your best performing architecture. **Note:** Use the validation set to check for performance in terms of the accuracy. Visualise the loss graphs for both training and validation to comment on the convergence of the chosen architecture. You can compare your chosen architecture design and training setting with other variants tried during experimentation to better support your findings. Make sure the training and testing of the model does not exceed 15 minutes and the script overwrites the weights after each epoch. Once, the training is completed, the script should write the predictions using the last epoch saved weights as a `predictions.pkl` files [Refer `Coding guidelines` for more details.] At the end of training or at the of 15 minutes (whichever is sooner) we will be using the predictions saved using the saved weights of the last epoch to evaluate the accuracy on the held out set [Refer coding guidelines and submission instructions for more details]. Please note that this is a competitive part, and your code would not be allowed to run for more than 15 minutes. Please use a time measuring library such as `time` library of `Python 3` to keep track of time and overwrite the weights after each epoch. The grading for this Part is relative among all students.

- **A note of caution:** The report must clearly specify how optimal parameters were reached in part (c) and (d). Any cutting corners will be treated as disciplinary misconduct and will be dealt with accordingly.

Appendix

I Neural Network Designs for subsections (a) and (b)

i. `Neural network for Binary and Multi-class classification`

625 → 512	"fc1": Input layer
Sigmoid	
512 → 256	"fc2": layer 1
Sigmoid	
256 → 128	"fc3": layer 2
Sigmoid	
128 → 1	"fc4": Output layer
Sigmoid	

Table 1: NN for binary classification [Part (a)]

625 → 512	"fc1": Input layer
Sigmoid	
512 → 256	"fc2": layer 1
Sigmoid	
256 → 128	"fc3": layer 2
Sigmoid	
128 → 8	"fc4": Output layer
Softmax	

Table 2: NN for Multi-class classification [Part (b)]

625 → 512	"fc1": Input layer
Sigmoid	
512 → 256	"fc2": layer 1
Sigmoid	
256 → 128	"fc3": layer 2
Sigmoid	
128 → 32	"fc4": layer 3
Sigmoid	
32 → 8	"fc5": Output layer
Softmax	

Table 3: NN for Multi-class classification [Part (c)]

ii. `List of Hyperparameters for Parts [a] and [b]`
   A. Batch size: 256
   B. Learning rate: 0.001
   C. Epochs: 15

II **Coding guidelines:**

- For parts (a) and (b) :
  - Don't shuffle/change order of the data files.
  - No other transformation must be done to the data other than the ones specified in the Data preprocessing instructions before feeding to the network as it may cause the weights to not match for Parts [a] and [b].
  - Don't apply early stopping criterion. Run for full specified number of iterations/epochs.
  - Be careful to code Mini-batch Gradient Descent to be closely consistent with the theoretical framework taught in class.
  - Make sure you do not use any random function other than `numpy.random.randn` with seed set as 0. Also make sure that the usage is as mentioned in the question.
  - Make sure there are no other libraries used other than `numpy, PIL, cv2, math, time, pandas, matplotlib, os`. **Usage of any other library is prohibited.**
  - In order for you to check your implementation, we have provided the weight files and checker script here. You can use the checker code to make sure the weights agree with the evaluation weights for Parts [a] and [b].
  - Your code must not print anything on the screen.
  - **Guidelines for savings weights:** The weights of the Neural Network should be saved as a Python `dict` of `dict` which contains two keys called `weights` and `bias` and value corresponding to each key is another `dict`. The `dict` corresponding to the `weights` key has keys as follows: `fc1, fc2, fc3, fc4 ... fcN` where N is the number of layers in the Network. Each key holds the corresponding weight matrix for that layer of the Neural Network. Similarly the `dict` corresponding to the `bias` key has keys as follows: `b1, b2, b3, b4 ... bN` where N is the number of layers in the Network. Each key holds the corresponding weight matrix for that layer of the Neural Network. Save this `dict` of `dict` as a pickle file named as `weights.pkl`. The checker script will also follow the same convention for checking the weights.

- For parts (c) and (d):
  - Your code must not print anything on the screen.
  - For Part [c], use the NN architecture as specified in the Appendix.
  - Make sure there are no other libraries used other than `numpy, PIL, cv2, math, time, pandas, matplotlib, os`. **Usage of any other library is prohibited.**
  - While using matplotlib for plots, make sure you never use plt.show() and rather save the plots.
  - Design code with emphasis on Vector Operations and minimal use of loops to ensure run-time efficiency.
  - Ensure that the training does not take more than 15 **minutes** for each submission of Part (c) and (d). Please use a time measuring library to keep track of time. Ensure your program runs using less than 8GB of RAM within 15 minutes on an *Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz* (see this link for theoretical peak GFLOPS of this processor). If your program is running within 10 minutes on your systems, things should be fine. The weights obtained at the end of training or at the end of 15 minutes (whichever comes earlier) will be used as the best weights for the evaluation on the held out set for these two parts. **Note:** Make sure you save the weights after the end of each epoch such that the `weights.pkl` holds the weights and biases of the immediate last epoch before training was concluded.
  - **Guidelines for savings weights:** The weights of the Neural Network should be saved as a Python `dict` of `dict` which contains two keys called `weights` and `bias` and value corresponding to each key is another `dict`. The `dict` corresponding to the `weights` key has keys as follows: `fc1, fc2, fc3, fc4 ... fcN` where N is the number of layers in the Network. Each key holds the corresponding weight matrix for that layer of the Neural Network. Similarly the `dict` corresponding to the `bias` key has keys as follows: `fc1, fc2, fc3, fc4 ... fcN` where N is the number of layers in the Network. Each key holds the corresponding bias matrix for that layer of the Neural Network. Save this `dict` of `dict` as a pickle file named as `weights.pkl`. The checker script will also follow the same convention for checking the weights.
  - For Part (c), model convergence in the least number of iterations is an important factor, hence after experimentation with different optimiser variations, fix the number of epochs in the submission script so that the training concludes once the model has reached the best weights in least amount of time in comparison to other variations. As once the training is completed, these best weights will be used to compute the loss on the train set. You can refer to the `Part-C` directory here for checking your weights format and the loss value on the train set.
  - **Guidelines to submit the predictions [For Part (d)]:** The `preprocessing.py` script is designed to load the train/valid set and its corresponding `.csv` file from the dataset folder. For Part (d), for training the model, call the `preprocess` function with the `'train'` argument so that it picks the training dataset and once the training is completed, call the `preprocess` function once again with the `'valid'` argument so that the validation/test data is loaded and model saves the predictions obtained on this set using the last epoch saved weights post training. Ensure that the training of the model and the saving of predictions post training is completed within 15 minutes. At the time of evaluation. we will using the held-out test instead of the val set to save the predictions using your script. While saving the predictions, ensure to save only the predicted labels on the test set as integer classes [refer class idx in `train.csv` or `val.csv`] in the form of a 1-D `numpy` array. For ex: If `pred` is the 1-D array storing the predictions, then `pred[i]` is the predicted class idx for the test sample `i`. The length of the `pred` array is the number of samples in the test set. Save this numpy array as a pickle file by the name `predictions.pkl`. While implementing the assignment, you can use the `val` set directory path and `val.csv` given to you to check if the `predictions.pkl` file is being saved correctly. While evaluation, we will be giving the hidden test set inplace of the given val set.

III Submission instructions: `Neural Network`

Submit your code in the form of the following executable python files called `part_a_binary.py`, `part_b_multiclass.py`, `part_c_multi_best.py`, `part_d_multi_best.py` and a well-written report explaining your experiments, results and observations named as `Entry_number_1_Entry_number_2_report.pdf`.

Submit all the python files as a zip named as `Entry_number_1_Entry_number_2_A2.zip` on **Moodle** and the report is to be submitted on **Gradescope**.

- I/O format for Part (a) and (b): `python part_a_binary.py --dataset_root --save_weights_path`
  The Python file for Part(a) and (b) submissions should take the following arguments as input from the command line: `--dataset_root --save_weights_path`. Here, `--dataset_root` corresponds to the root directory of the dataset containing the `train, val` splits along with their corresponding `.csv` files. This will be used by the preprocessing script to load the data for the Neural Network. The `--save_weights_path` argument holds the path where the weights and biases will be saved as `weight.pkl`. Refer Coding guidelines section to know more details about the format to save the weights in the `weights.pkl` file. The saved weights after the last epoch will be used to check if they match with our weights and the grading will be done accordingly.
  The same format will be followed for Part (b) as well.
- For Part (c): `python part_c_multi_best.py --dataset_root --save_weights_path`
  The arguments passed here hold the same meaning as the ones explained for Part(a) and (b). The weights saved at `save_weights_path` will be used to compute the loss for the competitive part.
- For Part (d): `python part_d_best.py --dataset_root --save_weights_path --save_predictions_path`
  The arguments passed here hold the same meaning as the ones explained for Part(a) and (b). The predictions saved at `--save_predictions_path` will be used to compute the accuracy on the held-out test set for the competitive part.
- Report: `Entry_number_1_Entry_number_2_A2.zip`
  Here you have to submit a `pdf` file with all details of experimentaions for Parts (c) and (d). Report results and observations of all variations experimented justifying how the the best set of hyperparameters and training settings were obtained. Plots/tables must be included wherever necessary.

IV **Note on grading:**

- For part-a and part-b, you can get 0 (if submission throws an error), partial marks (if the code runs successfully but the weights are incorrect within some predefined threshold) and full (the submission works as expected). We will be training your models to check if the obtained weights match the given weights for grading Parts (a) and (b). Make sure to initialise the weights in the same way as mentioned as any deviations would lead to incorrect results.
- For Part (c), marks will be given based on loss value on the best weights at the end of training or at the end of 15 minutes (whichever is sooner). Hence, fix the number of epochs in the submission scripts accordingly to obtain the best weights within the given time constraint. The lesser the loss obtained on train set in the given amount of time, higher the marks you get. Use a suitable time measuring library such as `time` from `Python` to ensure the execution within the time constraints. [Refer `coding guidelines` for more details.]
- For Part (d), marks will be given based on the accuracy obtained using the predictions saved at the end of training or at the end of 15 minutes (whichever is sooner). The accuracy will be computed on the hidden test set using the saved predictions using `predictions.pkl` file. The higher the value of accuracy obtained of the held-out test set in the given amount of time, higher the marks you get. Use a suitable time measuring library such as `time` from `Python` to ensure the execution within the time constraints. [Refer `coding guidelines` for more details.]
- For Part (c) and Part (d), there will be relative marking among all the students based on their respective submissions.
- For Part (c) and Part (d) marking will be done for code as well as the report.
- Please refer to the submission instructions, any submissions not following the guidelines will fail the auto-grading and be awarded 0. There will be NO DEMO for this assignment.
- Please post on Piazza if any coding guidelines or submission instructions are unclear to you.