Homework 5 – Deep Neural Networks (CS/DS 541, Murai, Spring 2024)

You may complete this homework assignment in teams of up to 4 people.

1 Facial Age Estimation using VGG16 [25 points]

In this project, you will train a VGG16 network to estimate how old each person looks in a set of face images (see `facesAndAges.zip` on Canvas, which contains 7500 grayscale images of size 48x48). You can use either TensorFlow (`https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16`) or PyTorch (`https://pytorch.org/vision/stable/models.html`). You can either train a VGG16 from scratch (which gives you the advantage that the input size can be tailored exactly to the dimensions of your dataset), *or* you can use a pre-trained VGG16 (with fixed input size of 224x224; you'll need to transform your dataset accordingly) with the strong advantage of harnessing the information stored in ImageNet. In either case, you will need to account for the fact that the images in your dataset (see below) are grayscale (1 input channel), whereas VGG16 expects RGB (3 channels); the easiest strategy is to replicate the grayscale values 3x. You will also need to modify the network so that the last layer outputs a real number, not a 1000-dim vector.

In particular:

1. Randomly partition the data into 80% training+validation, and 20% testing.
2. Either initialize the weights randomly, or initialize them according to their pre-trained values using ImageNet.
3. Either train the network from scratch, or perform supervised pre-training (train last few layers and possibly fine-tune the remaining layers).
4. After optimizing hyperparameters on the validation set, report your root mean squared error (RMSE) loss on the test set. Report your training, validation, and testing loss (RMSE) in the PDF file you submit. For full credit, you should get < 13 years RMSE on the 20% test partition.

2 Comparing Vanilla RNN with Variants in Sequence Modeling [25 points]

You will implement and train three different neural networks for sequence modeling: a Vanilla RNN (a simple RNN with shared weights), and two variants of a NN with a similar architecture to the Vanilla RNN but which do not share weights. You will compare their performance on a sequence prediction task and analyze the differences between them.

Prediction task: this is a many-to-one regression task, i.e., a sequence of inputs is used to predict a single output. In particular, the i-th input sequence $\mathcal{X}_i = (\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,\ell_i})$ has length ℓ_i, where $\mathbf{x}_{i,j} \in \mathbb{R}^{10}$; the i-th output is a scalar $y_i \in \mathbb{R}$.

Dataset: You will use a synthetic dataset containing sequences of variable lengths stored in the zip file `homework5_question2_data.zip`. Each sequence consists of input features and corresponding target values. The sequences are generated such that they represent a time-dependent process. Note that ℓ_i may be different than ℓ_j for $i \neq j$. So the (pickled) numpy object $\mathbf{X}$ is actually a list of sequences.

Tasks:

1. Implement a Vanilla RNN: Implement a Vanilla RNN architecture (needless to say, weights are shared across time steps). A pytorch starter code is provided in `homework5_starter.py`. **Important: You are not allowed to use an RNN layer implementation from any library**.

2. Implement a NN with Sequences Truncated to the Same Length: Implement a NN where sequences are truncated to have the same length before training. In other words, if the shortest sequence in the dataset has length L, all sequences should be truncated to length L before training.

3. Implement a NN with Sequences Padded to the Same Length: Implement another variant of NN where sequences are padded to have the same length before training. Use appropriate padding techniques to ensure that all sequences have the same length, and implement a mechanism to ignore the padding when computing loss and predictions.

4. Train and Compare the Models:

   (a) Train all three models (Vanilla RNN, Truncated NN, Padded NN) on the provided dataset.

   (b) Use a suitable loss function for sequence prediction tasks, such as mean squared error (MSE) or cross-entropy.

   (c) Train each model for a fixed number of epochs or until convergence.

   (d) Monitor and record performance metrics, such as training loss, on a validation set during training.

5. Evaluate and Compare the Models:

   (a) Evaluate the trained models on a separate test dataset.

   (b) Compare the performance of the three models in terms of MSE, convergence speed, and overfitting tendencies.

   (c) Analyze the results and discuss the advantages and disadvantages of each approach in terms of modeling sequences with varying lengths.

Additional Information:
You can choose the specific hyperparameters for your models, such as the number of hidden units, learning rate, batch size, and sequence length. Feel free to use any deep learning framework or library you are comfortable with, and provide clear code documentation. Note: Be sure to clearly explain your implementation, provide code comments, and present your results in a well-organized manner in the report.

3 Fine-tune a RoBERTa-based model [25 points]

In this project, you will first train a classification head using a pre-trained RoBERTa model on a dataset of social media tweets to classify tweets as containing medical information or not. You are provided with a dataset of social media tweets, where each tweet is labeled as either containing medical information (class 1) or not containing medical information (class 0). Then you will fine-tune the entire model to improve its accuracy. The preprocessing of the dataset, by tokenizing the tweets and converting them into a format suitable for RoBERTa, is already provided in the starter code: `https://colab.research.google.com/drive/1bQOE87rY5DGjYlpGCWmY5jbBL-9N5uyQ?usp=sharing`.

You will need to make the following changes to the existing code:

1. Split the dataset into training, validation, and test sets.

2. Train a classification head for a pre-trained RoBERTa model on the training set and evaluate its performance on the validation set. You can use the `ClassificationModel` from the `simpletransformers` library (see `https://simpletransformers.ai/docs/classification-models/`). Note 1: DO NOT

spend too much time tuning the batch size. We will leave that for the next item. Note 2: DO NOT train the entire model, only the classification head. To specify a custom set of parameter groups, refer to `https://simpletransformers.ai/docs/tips-and-tricks/#train-custom-parameters-only`.

3. Experiment with different values for batch size (consider powers of two, e.g., 2, 4, 8, ...). What is the maximum batch size you can use before the program crashes due to out of memory exceptions?

4. Set the batch size to the value found in the previous item. Train the model until it reaches approximate convergence. (Think about how you can demonstrate that.)

5. Evaluate the final model on the test set and report the performance metrics (e.g., accuracy, precision, recall, F1-score).

6. Now we will fine-tune the entire model. Experiment with two values of learning rates. (Note: Since this is fine-tuning, choose values that are relatively small compared to those typically used in training.) For each learning rate, determine how many epochs you can run before the model starts to overfit.

7. Based on the item above, discuss the performance of the model, any challenges faced during fine-tuning, and potential improvements that can be made to further improve accuracy.

Additional note: Use the Hugging Face transformers library to load and fine-tune the RoBERTa model.

In addition to your Python code (`homework5_WPIUSERNAME1.py` or `homework5_WPIUSERNAME1_WPIUSERNAME2.py` for teams), create a PDF file (`homework5_WPIUSERNAME1.pdf` or `homework5_WPIUSERNAME1_WPIUSERNAME2.pdf` for teams) containing the screenshots described above. **Please submit both the PDF and Python files in a single Zip file.**