Homework 5 – Deep Neural Networks (CS/DS 541, Murai, Fall 2023)

You may complete this homework assignment in teams of up to 3 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.

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.

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. You can use any suitable synthetic data generation method or existing dataset for this problem. 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 Neural Machine Translation [20 points]

In this project, you will train a seq2seq model to translate from English to French (see `eng-fra.txt.zip` on Canvas) using 2 different strategies. You can use either TensorFlow (`https://www.tensorflow.org/text/tutorials/nmt_with_attention`) or PyTorch (`https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`). In either case, you can download a GRU-based encoder-decoder jupyter notebook as starter code (which gives you the advantage that the text preprocessing steps are already implemented), but the pytorch starter is easier to follow since it uses less of the advanced features of the framework.

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

1. You must set up the training pairs so that the model can learn how to translate from English to French. The pytorch starter translates from French to English, while the TensorFlow starter is using a different dataset than the one on Canvas.

2. Randomly partition the data into 80% training, and 20% testing.

3. **Version 1: supervised training.** Train the network for at least 80 epochs.

4. **Version 2: pretraining as autoencoder.** Use unsupervised training to pretrain the encoder. Specifically, set the pairs so that you have two of the same phrase (I am test\tI am test), in order to train autoencoder. Try this:

   - Train as an autoencoder

- Save only the Encoder network and freeze its parameters (they must not be updated during training)
- Train a new Decoder for translation from there

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.**