Assignment 3
CS7.601 (Monsoon 2023)
Deep Learning: Theory and Practices

Submission Deadline:

Q1 and Q2: 11:55 PM, November 7th 2023

Q3: 11:55 PM, November 12th 2023

Max. Marks : 24

Instructions

1. Please submit your code for all the questions in **Jupyter Notebooks**.
2. Only the following libraries are allowed:
   - Numpy
   - Pandas
   - Matplotlib
   - Pytorch (version ≥ 2.0)
   - tqdm
   - scikit-learn
3. **Only submissions made on Moodle will be considered for evaluations.**

1 Transformers [4 Marks]

You have to train a transformer model to learn a transformation which takes a sentence and outputs a transformed sentence. Both the sentence and the transformed sentence are 8 characters long and only consist of lowercase English alphabets. You have been provided with "train_data.csv" which consists of 7000 sentence-transformed sentence examples and "eval_data.csv" which consists of 2000 sentence-transformed sentence examples. Do not use the samples from "eval_data.csv" for training the model.

1.1 Evaluation

For both the datasets, output the number of sentences with i correct character predictions where i ranges from 0 to 8. Use the "check" function provided below to calculate how many character predictions are correct.

```
def check(pred: str, true: str):
correct = 0
for a, b in zip(pred, true):
    if a == b:
        correct += 1
return correct
```

Save the model and implement a predict function which takes in a string and returns the transformed string. Your model will be tested on unseen dataset during evaluations so make sure it works properly.

2 Denoising autoencoders [4 marks]

In this question, you will be training a autoencoder to remove noise from an image. Create the noisy images by adding Gaussian noise with mean 0 and standard deviation 97 to every pixel of an image. Clip the pixel values of the images between 0 and 255. When displaying the images, you can round the pixel values to the nearest integer.
Pytorch provides train and test splits for the MNIST dataset. You are not suppose to use the test split for training in any form, including early stopping. If required, create an eval split from the train split, but having an eval split is not compulsory in this question.

2.1 Evaluation

Plot the MSE-epoch curve for the train dataset. Report the MSE of the trained model over the test dataset. Additionally, display original image, noisy image and reconstructed image for 10 samples from the test dataset. An example image is shown below.

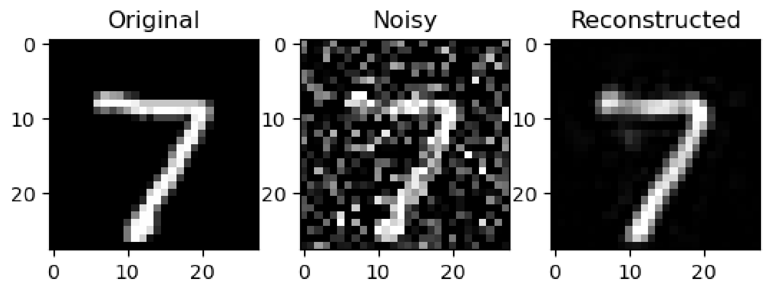

Figure 1: Example for 1 sample of the test dataset. Report 10 such examples from the test dataset

3 Encoder-Decoder with Attention [16 marks]

Implement the following paper: data2vis. The train, dev, and test splits are provided as train.sources, train.targets, dev.sources, dev.targets, and test.sources, test.targets. A dataset named progression.json has been provided for inference.

3.1 Architecture

The following architecture was used by the paper, you can modify it if desired.

- Embedding dimension: 512.
- Encoder: Bidirectional LSTM, 2 layers, 512 hidden units.
- Decoder: LSTM, 2 layers, 512 hidden units.
- Droupout: probability 0.5 before every encoder/decoder layer, only on the inputs, not the hidden/cell states.
- Attention type: Dot product.

- Attention vector dimension: 512.
- Max source sequence length: 500 (first 500).
- Max target sequence length: 500 (first 500).
- Max decode sequence length: 2000 (during inference, first 2000).
- Width for beam search: 15 (beam search is only used during inference).
- Optimizer: Adam, lr=1e-4
- Batch size: 32
- Number of steps: 20000 steps (not epochs, this means a total of 20000 mini-batches were used for training).

3.2 Evaluation

Marks will be provided based on the code but the model should still perform decently, it doesn't need to be perfect. The following things are expected:

- Average log perplexity of 0.032 on the test dataset.
- Generate 15 different possible encoding strategies for progression.json and save them.
- Visualize and save all of the encodings using online editor. If some of the produced encodings have mistakes, you can correct them manually for visualization purposes.

An example vegalite specification named example_vegalite.json has been provided as well. The data has been loaded already. The only thing you need to do is replace the provided mark and encoding with your model's outputs.