HOMEWORK SET 5

CSCI5527 Deep Learning (Fall 2022)

Due 11:59 pm, Dec 18 2022

Instruction Your writeup, either typeset or scanned, should be a single PDF file. For problem requiring coding, please organize all codes for each problem into a separate Jupyter notebook file (i.e., `.ipynb` file). Your submission into Canvas/Gradescope should include the single PDF and all the notebook files—**Please do not zip them**! No late submission will be accepted. For each problem, your should acknowledge your collaborators if any. For problems containing multiple subproblems, there are often close logic connections between the subproblems. So whenever possible, try to build on previous ones, rather than work from scratch.

Notation We will use small letters (e.g., u) for scalars, small boldface letters (e.g., $\boldsymbol{a}$) for vectors, and capital boldface letters (e.g., $\boldsymbol{A}$) for matrices. $\mathbb{R}$ is the set of real numbers. $\mathbb{R}^n$ is the space of n-dimensional real vectors, and similarly $\mathbb{R}^{m\times n}$ is the space of $m \times n$ real matrices. The dotted equal sign $\doteq$ means defining.

Problem 1 (Recurrent neural networks; $4/15$) In this problem, we will solve a simple text-based sentiment analysis problem. The dataset can be found here https://www.kaggle.com/kazanova/sentiment140. This github site https://github.com/bentrevett/pytorch-sentiment-analysis includes detailed tutorials on performing sentiment analysis using basic and advanced RNN models in PyTorch on the classical IMDb dataset. Please go over the tutorials and feel free to adapt the codes there.

(a) Read the instruction from the Kaggle website and load the data from the sentiment140 dataset. We will use the text field to predict the target, i.e., polarity. The text field is not as clean as the IMDb dataset, e.g., the "@ xxxx" part is probably not useful for sentiment analysis. Perform data clean-up when necessary. There is a single data file in the dataset. Please split it into 60% training, 20% validation, and 20% test. $(1/15)$

(b) Design and train a sentiment analysis model on the data. Again, feel free to start with the above-mentioned sentiment analysis tutorial and adapt the models there. $(2/15)$

(c) A "85%" test: you'll get 1 point if your classification accuracy exceeds 85%. $(1/15)$

Problem 2 (Attention and Tranformers; $6/15$) In this problem, we will practice the basics of the attention mechanism and try to get a precise understanding of the building components in Transformers.

(a) Take 10 random digit images from MNIST, one for each class, and then vectorize them. These are the source vectors. Then take another target vector from class "5". Write code to calculate the (cross-)attention weights $\boldsymbol{w}$ and the induced weighted sum over the source vectors—you're free to choose any attention variant we talked of in class or in the literature, e.g., dot-product attention, multiplicative attention, etc. Reshape and display your weighted sum as an image—same size as the original MNIST images. Also, display the attention matrix. Does "5" have a high correlation with "5" in the source vectors? $(1.5/15)$

(b) Each encoder layer in the Transformer model consists of multi-head self-attention layer followed by a shallow feedforward network applied to each position separately and identically. Implement an encoder layer; you probably need to check out the related sections of the original paper https://arxiv.org/abs/1706.03762 or other online resources to figure out the details. PyTorch has a built-in implementation https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html; unfortunately, due to the randomness in the internal weights at initialization, it is not easy to benchmark your implementation against the built-in. (2/15)

(c) Check out this tutorial on French-to-English translation using a RNN-based Seq2Seq model: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html. Now: (1) Replace the embeddings with pre-trained word embeddings such as word2vec or GloVe; (2) Replace the model with a Transformer-based encoder-decoder model (the WMT 2014 English-to-French translation task in the original Transformer paper https://arxiv.org/abs/1706.03762 might be helpful also); you can use PyTorch built-in Transformer implementation https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html, retrain the model, and compare your result with that obtained from the RNN-based Seq2Seq model. (2.5/15)

Problem 3 (Generative models; 5/15) Finally, we're here to generate new fashionable items! In other words, we will train generative models based on the famous Fashion-MNIST dataset https://github.com/zalandoresearch/fashion-mnist, which is available as a PyTorch standard dataset https://pytorch.org/docs/stable/torchvision/datasets.html#fashion-mnist. There are numerous implementations of the following algorithms on the Internet; you can occasionally consult these online resources when feeling uncertain, but your implementations must be your own work.

(a) Train a GAN and generate 10 new items after training. For the GAN, you can use either the original form, or any modified variation, e.g., W-GAN. (2/15)

(b) Modify the above implementation into a conditional GAN, i.e., with class labels as input augmentation for both generator and discriminator. Repeat the training and generation, and show at least 1 new item from each class and visually compare your new results with those from (a). (1/15)

(c) Implement and train a variational autoencoder (VAE), and also generate 10 random samples from it. As is standard in VAE, let's assume the approximate posterior $q(\boldsymbol{z}|\boldsymbol{x})$ and the conditional $p(\boldsymbol{x}|\boldsymbol{z})$ take multivariate Gaussian form with diagonal covariance structure. Sec. 3 and Appendix C of the original paper https://arxiv.org/abs/1312.6114 may help you to clear up doubts. Make sure to implement the reparametrization trick so that auto differentiation can be performed. (2/15)