HOMEWORK SET 4

CSCI5527 Deep Learning (Spring 2026)

Due 11:59 pm, Apr 26 2026

Instruction Your writeup, either typeset or scanned, should be a single PDF file. For problems requiring coding, organize all codes for each top-level problem (i.e., Problem 1, Problem 2, etc) into a separate Jupyter notebook file (i.e., `.ipynb` file). Your submission to Gradescope should include the single PDF and all notebook files—**please DO NOT zip them**! No late submission will be accepted. For each problem, you should acknowledge your collaborators—**including AI tools**, if any.

About the use of AI tools You are strongly encouraged to use AI tools—they are becoming our workspace friends. It takes a bit of practice to ask the right and effective questions/prompts to these tools; we highly recommend that you go through this popular free short course **ChatGPT Prompt Engineering for Developers** offered by https://learn.deeplearning.ai/ to get started.

If you use any AI tools for any of the problems, you should include screenshots of your prompting questions and their answers in your writeup. The answers provided by such AI tools often contain factual errors and reasoning gaps. **So, if you only submit an AI answer with such bugs for any problem, you will obtain a zero score for that problem**. You obtain the scores only when you explain the bugs and also correct them in your own writing. You can also choose not to use any of these AI tools, in which case we will grade based on the efforts you have made.

Notation Please refer to our supplementary notes on high-dimensional calculus

Important notes Please provide detailed steps with justification for all problems; jumping to the final results leads to a zero score. Also, **if we ask you to use certain facts/tools to obtain something, you have to use these facts/tools (perhaps plus others); otherwise, you get a zero score**, e.g., if you prove everything from scratch when we ask you to use an existing theorem.

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

(a) Implement an LSTM cell, and compare its behavior with the PyTorch built-in at https://pytorch.org/docs/stable/generated/torch.nn.LSTMCell.html#torch.nn.LSTMCell. For the implementation, please follow the exact formulas on the previous website; for comparison, you can follow these steps: (i) follow the example at the bottom of the `torch.nn.LSTMCell` page to create the output; (ii) read the randomly initialized weights inside the rnn module in that example (note that these weights are the module variables as described in their documentation) and use these weights for your own implementation of LSTM; (iii) execute the example based on your implementation of the LSTM module and compare the outputs from the two different versions. (1/15)

(b) Read the instructions on 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% for training, 20% for validation, and 20% for test. $(1/15)$

(c) 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. $(1/15)$

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

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

(a) Take 10 random digit images from MNIST, one for each class, and then vectorize them. These are the source vectors. Then follow the same procedure to independently form 10 target vectors. Write code to calculate the (cross-)attention between the 10 target vectors and the 10 source vectors, i.e., for each target vector, follow the standard dot-product attention to form a weighted sum of the source vectors, and then reshape and display your weighted sum as an image—of the same size as the original MNIST images. $(0.5/15)$

In addition, display the 10×10 attention matrix, which visualizes the combination weights. What do you observe from the attention matrix? $(0.5/15)$

(b) For any pair of given vectors $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n$ where m may or may not equal n in practice, the attention score simply measures "similarity" between them. In classical machine learning, kernels are a systematic way of measuring similarities: for $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n$, $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle$, where $\Phi(\cdot)$ is the feature mapping induced by the kernel K. Recall that in kernel methods, we do not need to define the mapping Φ but instead just need to define K directly. For example, the famous Gaussian (or radial basis function, RBF) kernel is: $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2 / \sigma^2)$. What are the relative pros and cons of using attention vs. using the classical kernel to measure similarities? $(1/15)$

(c) Moving to self-attention, for each input, we have a learnable (query, key, value) triple, and the output will be a weighted sum of all the values based on the attention mechanism. Given a data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ where each row represents a token, the output can be succinctly written as

$$\boldsymbol{X}_{new} = \text{softmax}(\boldsymbol{X}\boldsymbol{W}^Q(\boldsymbol{W}^K)^\intercal \boldsymbol{X}^\intercal)\boldsymbol{X}\boldsymbol{W}^V, \tag{1}$$

where the three matrices $\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V \in \mathbb{R}^{d \times D}$ are learnable. What is the computational complexity of (1) for a forward pass? What happens when N is large? $(1/15)$

Given any $N \times N$ permutation matrix $\boldsymbol{\Pi}$ (https://en.wikipedia.org/wiki/Permutation_matrix), $\boldsymbol{\Pi}\boldsymbol{X}$ permutes the rows of $\boldsymbol{X}$, i.e., the input tokens. Prove that the output will change into $\boldsymbol{\Pi}\boldsymbol{X}_{new}$, i.e., the output tokens will be permuted in the same way as the input. $(0.5/15)$

(d) Each encoder layer in the Transformer model consists of a multi-head self-attention layer followed by a shallow feedforward network applied to each position separately and identically. Implement an encoder layer; you may want to consult the relevant sections of the original paper https://arxiv.org/abs/1706.03762 or other online resources for details. $(1/15)$ (**Optional**) You may want to compare your implementation against the PyTorch built-in https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html, but it takes some effort to figure out how to read the weights inside a Transformer layer.

(e) Check out this tutorial on French-to-English translation using a RNN-based Seq2Seq model: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html and then: (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. (1.5/15)

Problem 3 (Graph neural networks (GNNs); 5/15)

(a) From applied domains that you are familiar with, describe a problem that can be formulated as a task on graphs. What are the graph nodes and edges, and what exactly is the task in graph language? (1/15)

(b) Real-world graphs are typically very sparse, and hence the node-list or edge-list representation is preferred over the affinity matrix representation. Popular GNN libraries such as PyTorch Geometric (PyG, https://pytorch-geometric.readthedocs.io/en/latest/) and Deep Graph Library (DGL, https://www.dgl.ai/) both use the edge-list format by default, i.e., by specifying the list of all edges as a $2 \times |E|$ tensor, where $|E|$ is the number of edges, and each column contains the index of the source and end nodes (i.e., directed edges by default), respectively.

We will use PyG here. Please walk through their quick introduction https://pytorch-geometric.readthedocs.io/en/latest/get_started/introduction.html, and perform the following tasks—**you can use any of their built-in functions unless explicitly told not**: (1) load the `KarateClub` dataset (the directed version) from their built-in dataset; (2) find all distance-1 neighbors of Node 10; (3) find the largest connected subgraph inside the graph; (4) implement one-layer plain graph convolution based on distance-1 neighborhood, apply it to the graph, and display your resulting features. For simplicity, we assume that both trainable weight matrices are $\boldsymbol{I}$. Here, you can use the built-in neighbor function to find the neighbors, but the convolution operation must be your own implementation. (2/15)

(c) Load the `WikipediaNetwork` dataset under `torch_geometric.datasets`, and randomly divide the dataset into $80\% : 20\%$ for training and test, respectively. Details of the datasets, graph structures, and node features can be found from https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.WikipediaNetwork.html#torch_geometric.datasets.WikipediaNetwork, particularly in the "Wikipedia graphs" bullet point paragraph (page 11) of the original paper https://arxiv.org/pdf/1909.13021.pdf. Design your own graph neural network architecture and predictor, and report the regression performance that you obtain. You should measure your performance by the mean absolute error. (2/15)