ECE/CS/ME 539 – Fall 2024 — Homework 11

Exercise 1

(3.5 points) Please download the starter code `TimeSeries_starter.ipynb` from Canvas. In this activity, we will implement an auto-regressive model to predict the future price of bitcoin from its recent history. We will consider an auto-regressive model for the price change at time t given by

$$\Delta_t = w_0 + \sum_{i=1}^{T} w_i x_{t-i}$$

where x_t is the bitcoin price at time t, w_i are the model weights and Δ_t is the predicted difference between the last price x_{t-1} and the price we want to predict x_t. As can be seen in the equation above, this is a simple linear model whose input features are the prices of bitcoin over the prior T days.

(a) The code provides the daily price of bitcoin from October 2013 until January 2021. Start by analyzing the data. What are the minimum and maximum prices within this time period? On which days did they occur?

(b) The code implements a `TimeSeriesDataset` class which draws sub-sequences of a prespecified length from the original time series starting at a specific time index. Create an instance of this class using the bitcoin price data. Then, index the dataset object to get the prices starting on 2017-12-01 for a total of 30 days.

(c) Let's now train the model assuming $T = 9$. In other words, our goal is to predict the price on the 10th day from the sequence of prices in the prior 9 days. Start by creating a linear model using `nn.Linear`, and the dataset class with the appropriate sequence length. How many input and output channels should the linear model have? What should the sequence length for the dataset class be (recall that you'll need to use the generated sequence to define both the input to the model and its target)?

(d) Complete the `train()` function. This function should loop through all sub-sequences in the dataset, compute the model's prediction for the price change Δ_t, and update the weights to minimize the mean absolute error loss between the predictions and the ground-truth price changes.

(e) Once the `train()` function is complete, train the model for 300 epochs with a learning rate of 0.0005 and batch size of 32 using the Adam optimizer. You should obtain an error of approximately 128 per BTC.

(f) Using the trained model, predict the price of bitcoin on the provided data. Recall that the model predicts the price change Δ_t from x_{t-1} to x_t. Analyze the plots of the predicted and ground truth bitcoin price, as well as the histogram of the errors. Discuss the results.

Exercise 2

(3 points) Please download the starter code `SimpleRNN_starter.ipynb` from Canvas. In this activity, we will (1) implement a simple RNN model and (2) train it for the next word prediction model. The code downloads the short story *"The Time Machine"* by H. G. Wells, made public by *Project Gutenberg*.

1. Start by constructing the vocabulary from the text corpus with all unique words plus an additional `"UNK"` token.
   (a) How many words were found?
   (b) What are the 10 most common words found in the text?
2. The `TextCorpusDataset` class implements an indexable torch dataset.
   (a) Build the dataset using the vocabulary from part 1 and a sequence length of 50.
   (b) How many sequences can we draw from this dataset?
   (c) If we sample the 1st sequence by running `dataset[0]`, we will get words 0-49 of the original corpus. Which words will we get when running `dataset[4]`?
3. Implement the RNN model. Suppose that, at time t, we are given an input vector x_t and the hidden state of the model at the previous step h_{t-1}. Then, an RNN updates its state

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

   where W_{xh} and W_{hh} are weight matrices and b_h the bias of the model. Furthermore, at each time step t, the model computes its output as follows

$$o_t = W_{ho}h_t + b_o$$

   (a) Implement the RNN equations above using `nn.Linear()`. Assume that we will feed one-hot embeddings for each word (i.e., the dimension of x is equal to the vocabulary size).
   (b) Build the `SimpleRNN` model with hidden dimension of 256.
   (c) Feed the sentence "Today is too darn cold" to the model. What did this untrained model predict? Use `dataset.convert2idx()` method and `F.one_hot()` to convert the sentence into each word 1hot embeddings. Finally, convert the model's predictions back to words.
4. Train the RNN model. Given a long sequence, RNNs are known to have a problem with exploding or vanishing gradients which makes them hard to train. To prevent exploding gradients, one strategy is to "unroll" the RNN for short sequences.
   (a) Complete the `train_on_sequence()` function by defining the short input and target sequences to train the model on.
   (b) Train the model for 100 epochs with a learning rate of 0.001, batch size of 32 and an unroll sequence of 5.
   (c) What is the perplexity of the trained model? Compared to the autoregressive MLP model obtained in last class, which one performed the best? Why?
   (d) Using the generate function, test the model ability to generate new text, by providing your own prompt and checking how the model completes it.

Exercise 3

SMS Spam Detection (3.5 points)
In class, we explored the use of simple RNNs for language modeling (a next token prediction task). RNNs can also be used for other tasks. In this assignment, we will train a simple RNN model for spam detection on text messages.

(a) Start by downloading and preparing the dataset by running the starter code `SpamDetection_starter.ipynb`. Each entry in the list data will be an SMS, where the first word denotes whether the SMS is spam (spam) or not (ham).

(b) Split the data into 80% training and 20% testing.

(c) Create a vocabulary of 10,000 unique words and add tokens `"/UNK"` and `"/PAD"`.

(d) Create the dataset class to sample SMS messages and the corresponding ground-truth label (whether it is spam or not). To facilitate training, pad or truncate all messages to a length of exactly 30 words. To pad the message, simply add `"/PAD"` tokens to the beginning of the message.

(e) Create a simple RNN binary classifier, which parses a sequence of word tokens and computes its prediction using a linear layer operating on the final state (i.e., the state after parsing the last word).

(f) Fit the model using the Adam optimizer for 20 epochs with a learning rate of 0.001, and batch size of 32. Measure the accuracy on the training and testing data.

(g) For each of the first 10 SMS in the test set, print the message, the model prediction, and the ground truth.