UNIVERSITY OF PENNSYLVANIA

ESE 546: PRINCIPLES OF DEEP LEARNING

FALL 2024

[10/14] HOMEWORK 3

DUE: 11/04 MON 11:59 PM ET

Read the following instructions carefully before beginning to work on the homework.

- You will submit solutions typeset in LaTeX on Gradescope (strongly encouraged). You can use hw_template.tex on Canvas in the "Homeworks" folder to do so. If your handwriting is unambiguously legible, you can submit PDF scans/tablet-created PDFs.
- Clearly indicate the name and Penn email ID of all your collaborators on your submitted solutions.
- Start a new problem on a fresh page and mark all the pages corresponding to each problem. Failure to do so may result in your work not graded completely.
- For each problem in the homework, you should mention the total amount of time you spent on it. This helps us keep track of which problems most students are finding difficult.
- You can be informal while typesetting the solutions, e.g., if you want to draw a picture feel free to draw it on paper clearly, click a picture and include it in your solution. Do not spend undue time on typesetting solutions.
- You will see an entry of the form "HW 3 PDF" where you will upload the PDF of your solutions. You will also see entries like "HW 3 Problem 3 Code" where you will upload your solution for the respective problems.
- **For each programming problem/sub-problem, you should create a fresh .py file**. This file should contain **all** the code to reproduce the results of the problem/sub-problem, e.g., it should save the plot that is required (correctly with all the axes, title and legend) as a PDF in the same directory. You will upload the .py file as your solution for "HW 3 Problem 3 Code". Name your file as pennkey_hw3_problem3.py, e.g., I will name my code as pratikac_hw3_problem3.py. Note, we will not accept .ipynb files (i.e., Jupyter notebooks), you should only upload .py files. If you are using Google Colab to do your homework (and I suggest that you don't...), you can export the notebook to a .py file.
- **This is very important**. Note that the instructors will download your code and execute it themselves, so your code should be such that it can be executed independently without any errors to create all output/plots required in the problem.

Credit The points for the problems add up to 80. There are no extra points in this homework, i.e., your final score will be min(your total points, 80).

Problem 1 (10 points). In Chapter 7 of the lecture notes, we analyzed the objective

$$\min_{w} \mathop{\mathrm{E}}_{R}\left[\|y - (R \odot X)\, w\|_2^2\right] \tag{1}$$

where $X \in \mathbb{R}^{n \times d}$ are the input data written together as a matrix, $y \in \mathbb{R}^n$ are the targets and $w \in \mathbb{R}^d$ are the weights. The Dropout mask $R \in \{0,1\}^{n \times d}$ consists of entries that are Bernoulli random variables with parameter $1-p$. Compete the calculation of Section 7.3.2 to show that

$$\min_{w} \mathop{\mathrm{E}}_{R}\left[\|y - (R \odot X)\, w\|_2^2\right] = \min_{\tilde{w}} \|y - X\tilde{w}\|_2^2 + \left(\frac{p}{1-p}\right)\, \tilde{w}^\top \mathrm{diag}(X^\top X)\tilde{w}.$$

where $\tilde{w} = (1-p)w$.

Problem 2 (10 points). Consider a dataset $\left\{(x^i, y^i)\right\}_{i=1,\dots,n}$ where $x^i \in \mathbb{R}^d$ and $y^i \in \mathbb{R}$. If we arrange all the data in a matrix $X \in \mathbb{R}^{n \times d}$ and targets as a vector $Y \in \mathbb{R}^n$ we can perform linear regression by solving

$$w^* \in \mathop{\mathrm{argmin}}_{w \in \mathbb{R}^d} \|Y - Xw\|_2^2 \tag{2}$$

when we have $d \leq n$, i.e., the number of data is more than the number of parameters to be estimated. If we have fewer data than the number of parameters $d > n$, the solution to the above problem is not unique. In this case, we pick one of the many solutions, say the one with the smallest ℓ_2 norm, using

$$\begin{aligned} w^* = \min_{w \in \mathbb{R}^d} \frac{1}{2}\|w\|_2^2 \\ \text{such that } Y = Xw. \end{aligned} \tag{3}$$

(a) (8 points) Derive the solution w^* for the problem in Eq. (3). Assume $XX^\top$ is invertible.

(b) (2 points) Write down all solutions of Eq. (2). Hint: think of the null-space of the matrix X.

Problem 3 (60 points, You can try to do this problem on your laptop when you are debugging but Colab with GPU will train much faster). We will implement a model to predict the next character in a given sentence. We will use a few different sources of data to build this model (think carefully about how you will create a train and test set)

(1) The complete works of Shakespeare, which you can get at https://www.gutenberg.org/ebooks/100.txt.utf-8 which is hosted at https://www.gutenberg.org/ebooks/100.

(2) Leo Tolstoy's War and Peace as our training dataset. You can get the text file of the entire book from https://www.gutenberg.org/ebooks/2600.txt.utf-8 which is hosted at https://www.gutenberg.org/ebooks/2600.

(3) Any English book of your choice which is sufficiently long, e.g., its text file is at least 1 MB. You can find text files for many books at Project Gutenberg https://www.gutenberg.org.

(a) (5 points) First observe that characters in the text can be letters, numbers, punctuation, and newlines. We will represent each character using its one-hot encoding; you should do

```
m = length(set(all_chars))
```

to get the correct number of unique characters. This is known as the size of the vocabulary in NLP. Do not worry about cleaning the dataset; neural networks are surprisingly good at handling unclean data. You will write a function that takes a part of the text input, say a sequence of 32 characters, and converts it into this embedding.

(b) (25 points) We now have written our problem in the standard form for an RNN where we are given a long sequence of data $x_1, x_2 \ldots$. Each element here is the embedding of a character or a punctuation mark. You will train an RNN with one hidden layer which predicts the one-hot vector of the next *character* as output, given the past sequence. This corresponds to the operations

$$h_{t+1} = \tanh\left(w_h h_t + w_x x_{t+1} + b_h\right) \tag{4}$$

$$\mathbb{R}^{75} \ni \hat{y}_t = \text{softmax}(w_y h_t + b_y) \tag{5}$$

$$\ell(\hat{y}_t, y_t) = \ell(\hat{y}_t, x_{t+1}) = -\log(\hat{y}_t)_{x_{t+1}} \tag{6}$$

$$\ell = \frac{1}{T}\sum_{t=1}^{T} \ell(\hat{y}_t, y_t). \tag{7}$$

Notice that we are going to predict softmax output logits $\hat{y}_t \in \mathbb{R}^{75}$ over the entire vocabulary. The loss is simply the cross-entropy loss of predicting the correct next character.

Code the RNN using PyTorch and train it. Plot the training error as a function of the number of weight updates. The validation error of an RNN is calculated the same way as the training error, it is the cross-entropy loss on data that was not a part of the training set. Report the validation error over a future sequence of 32 characters every 1000 weight updates.

You should also report a few examples of the RNN for generating new sentences: to do so initialize the hidden state to zero and roll the RNN forward by setting x_{t+1} in Eq. (4) to the prediction of previous step $\hat{y}_t$. You will notice that although the RNN obviously does not do a good job of generating correct words, it gets syntactic things like punctuation more or less correct.

Some tips: You should use the inbuilt nn.RNN module to build the recurrent network. You should also use torch.optim.Adam as the optimizer instead of SGD (we will study this soon) with a learning rate of 10^{-3}. Set the length of the sequence in the mini-batch to be $T = 32$. You may have to clip the gradients before calling optim.step() using the function torch.nn.utils.clip_grad_norm_. For more tips read the code at https://github.com/pytorch/examples/tree/master/time_sequence_prediction and https://github.com/pytorch/examples/blob/master/word_language_model/main.py carefully.

(c) (30 points) You will now train a self-attention-based neural network for the same problem. You can use the nn.Transformer module for this part. Again read the Pytorch example code linked to above carefully to understand its different parts. You will need to modify this code by quite a bit before you can use it for this problem. You should use a context length of 128 characters in this case.

Plot the training error as a function of the number of weight updates. The validation error of a self-attention-based model is also calculated the same way as the training error, it is the cross-entropy loss on data that was not a part of the training set. Report the validation error over a future sequence of 128 characters every 1000 weight updates.

Again, you should also report a few examples of generating new sequences of length about 1024, starting from a sequence of length 16. You should describe how you implemented this mechanism.