

UNIVERSITY OF PENNSYLVANIA
ESE 546: PRINCIPLES OF DEEP LEARNING
FALL 2020
[10/19] HOMEWORK 3
DUE: 10/27 TUE 1.30P ET

Changelog

- 10/20 11am: Minor change on line 27 on how to compute the size of the vocabulary.

Instructions

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.
- Please 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.
- Clearly indicate the name and Penn email ID of all your collaborators on your submitted solutions.
- For each problem in the homework, you should mention the total amount of time you spent on it.
- 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 4 Code” where you will upload your solution for the respective problems. **For each programming problem, you should create a fresh Google Colab notebook.** This notebook should contain all the code to reproduce the results of the problem and you will upload the `.ipynb` file obtained from Colab. Name your notebook to be “`pennkey_hw3_problem4.ipynb`”, e.g., I will name my code for Problem 4 as “`pratikac_hw3_problem4.ipynb`”.
- **Remember that you should include all the relevant plots in the PDF, without doing so you will not get full credit. Your PDF solutions should be completely self-contained, we should not have to look at/run the Colab notebook to check your solutions.**

Credit The points for the problems add up to 80. You only need to solve for 50 points to get full credit, i.e., your final score will be $\min(\text{your total points}, 50)$.

1 **Problem 1 (30 points).** This problem will show that co-coercivity of the gradient and its Lipschitz
 2 continuity are equivalent. Assume that the function $f(x)$ is convex.

3 (a) (10 points) For a differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, show that co-coercivity of ∇f implies
 4 Lipschitz continuity of ∇f . I.e., show that for all x, y

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \frac{1}{L} \|\nabla f(x) - \nabla f(y)\|^2 \Rightarrow \|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|.$$

5 Hint: use the Cauchy-Schwartz inequality.

6 (b) (10 points) Show the converse, i.e., Lipschitz continuity of ∇f implies co-coercivity.

7 (c) (10 points) For a twice-differentiable function $f(x)$ with L -Lipschitz gradients and strong-
 8 convexity parameter m , show that

$$m \leq \|\nabla^2 f(x)\|_2 \leq L$$

9 for all x .

10 **Problem 2 (10 points).** In Chapter 7 of the lecture notes, we analyzed the objective

$$\min_w \mathbb{E}_R [\|y - (R \odot X) w\|_2^2] \quad (1)$$

11 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 $w \in \mathbb{R}^d$
 12 are the weights. The Dropout mask $R \in \{0, 1\}^{n \times d}$ consists of entries that are Bernoulli random
 13 variables with parameter $1 - p$. Complete the calculation of Section 7.3.2 to show that

$$\min_w \mathbb{E}_R [\|y - (R \odot X) w\|_2^2] = \min_{\tilde{w}} \|y - X \tilde{w}\|_2^2 + \left(\frac{p}{1 - p} \right) \tilde{w}^\top \text{diag}(X^\top X) \tilde{w}.$$

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

15 **Problem 3 (10 points).** Let the population risk of a regression model be

$$R(f) = \int |f(x) - y|^2 P(x, y) \, dx \, dy.$$

16 Show that the model that minimizes the population risk, we called this the optimal classifier in
 17 Chapter 7, is given by

$$f^* = \operatorname{argmin}_f R(f) = \mathbb{E}[y \mid x].$$

18 **Problem 4 (30 points, You can try to do this problem on your laptop when you are debugging**
 19 **but Colab with GPU will train much faster).** We will implement a recurrent neural network for
 20 predicting the next character in a given sentence. We will use Leo Tolstoy's War and Peace as our
 21 training dataset. You can get the text file of the entire book from [https://github.com/mmckay/nyu-econ-](https://github.com/mmckay/nyu-econ-370/blob/master/notebooks/data/book-war-and-peace.txt)
 22 [370/blob/master/notebooks/data/book-war-and-peace.txt](https://github.com/mmckay/nyu-econ-370/blob/master/notebooks/data/book-war-and-peace.txt)

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

```
25 m = length(set(all_chars))  
26
```

27 to get the correct length of the one-hot vector. This is known as the size of the vocabulary in NLP.
 28 Do not worry about cleaning the dataset; neural networks are surprisingly good at handling unclean
 29 data. You will write a function that takes a part of the text input, say a sequence of 25 characters, and
 30 converts it into this embedding.
 31

(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 $x_1, x_2 \dots$. Each element here is the embedding of character or 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 function

$$h_{t+1} = \tanh(w_h h_t + w_x x_{t+1} + b_h) \quad (2)$$

$$\mathbb{R}^{64} \ni \hat{y}_t = \text{softmax}(w_y h_t + b_y) \quad (3)$$

$$\ell(\hat{y}_t, y_t) = \ell(\hat{y}_t, x_{t+1}) = -\log(\hat{y}_t)_{x_{t+1}} \quad (4)$$

$$\ell = \frac{1}{T} \sum_{t=1}^T \ell(\hat{y}_t, y_t). \quad (5)$$

32 Notice that we are going to predict softmax output logits $\hat{y}_t \in \mathbb{R}^{64}$ over the entire vocabulary. The
33 loss is simply the cross-entropy loss of predicting the correct next character.

34 Code the RNN using PyTorch and train it. Plot the training error as a function of the number of
35 weight updates. The validation error of an RNN is calculated by simply setting x_{t+1} in (2) to zero
36 “rolling forward” the RNN to predict the future outputs. Report the validation error over a future
37 sequence of 25 characters every 1000 weight updates.

38 Some tips: You should use the inbuilt nn.RNN module to build the recurrent network. You should
39 also use torch.optim.Adam as the optimizer instead of SGD (we will study this soon) with a learning
40 rate of 10^{-3} , set the batch-size of 1 (i.e., think of the entire book/dataset as one really long sen-
41 tence), and the length of the sequence in the mini-batch to be $T = 25$. You may have to clip the
42 gradients before calling optim.step() using the function torch.nn.utils.clip_grad_norm_. For more
43 tips read the code at https://github.com/pytorch/examples/tree/master/time_sequence_prediction and
44 https://github.com/pytorch/examples/blob/master/word_language_model/main.py carefully.

45 War and Peace is a really long book, feel free to use a subset of the book to train/validate your model.
46 Similarly, feel free to modify the suggested hyper-parameters.