Deep Learning / Spring 2023

Homework 1

Please upload your assignments on or before February 17, 2023.

- You are encouraged to discuss ideas with each other. But you **must acknowledge** your collaborator, and you **must compose your own** writeup and code independently.
- We **require** answers to theory questions to be written in LaTeX. (Figures can be hand-drawn, but any text or equations must be typeset.) Handwritten homework submissions will not be graded.
- We **require** answers to coding questions in the form of a Jupyter notebook. It is **important** to include brief, coherent explanations of both your code and your results to show us your understanding. Use the text block feature of Jupyter notebooks to include explanations.
- Upload both your theory and coding answers in the form of a **single PDF** on Gradescope.

---

1. **(3 points)** *Linear regression with non-standard losses.* In class we derived an analytical expression for the optimal linear regression model using the least squares loss. If X is the matrix of n training data points (stacked row-wise) and y is the vector of their corresponding labels, then:

   a. Using matrix/vector notation, write down a loss function that measures the training error in terms of the ℓ_1-norm. Write down the sizes of all matrices/vectors.

   b. Can you simply write down the optimal linear model in closed form, as we did for standard linear regression? If not, why not?

   c. In about 2-3 sentences, reflect on how you solved this question, and relate any aspects of this to the 3-step recipe for ML that we discussed in class.

2. **(4 points)** *Expressivity of neural networks.* Recall that the functional form for a single neuron is given by $y = \sigma(\langle w, x \rangle + b, 0)$, where x is the input and y is the output. In this exercise, assume that x and y are 1-dimensional (i.e., they are both just real-valued scalars) and σ is the unit step activation. We will use multiple layers of such neurons to approximate pretty much any function f. There is no learning/training required for this problem; you should be able to guess/derive the weights and biases of the networks by hand.

   a. A *box* function with height h and width δ is the function $f(x) = h$ for $0 < x < \delta$ and 0 otherwise. Show that a simple neural network with 2 hidden neurons with step activations can realize this function.

Draw this network and identify all the weights and biases. (Assume that the output neuron only sums up inputs and does not have a nonlinearity.)

b. Now suppose that f is *any arbitrary, smooth, bounded* function defined over an interval $[-B, B]$. (You can ignore what happens to the function outside this interval, or just assume it is zero). Use part a to show that this function can be closely approximated by a neural network with a hidden layer of neurons. You don't need a rigorous mathematical proof here; a handwavy argument or even a figure is okay here, as long as you convey the right intuition.

c. Do you think the argument in part b can be extended to the case of d-dimensional inputs? (i.e., where the input x is a vector – think of it as an image, or text query, etc). If yes, comment on potential practical issues involved in defining such networks. If not, explain why not.

d. In about 2-3 sentences, reflect on how you solved this question, and relate any aspects of this to the 3-step recipe for ML that we discussed in class.

3. **(3 points) Calculating gradients**. Suppose that z is a vector with n elements. We would like to compute the gradient of $y = \text{softmax}(z)$. Show that the Jacobian of y with respect to z, J, is given by the

$$J_{ij} = \frac{\partial y_i}{\partial z_j} = y_i(\delta_{ij} - y_j)$$

where δ_{ij} is the Dirac delta, i.e., 1 if $i = j$ and 0 else. *Hint: Your algebra could be simplified if you try computing the log derivative,* $\frac{\partial \log y_i}{\partial z_j}$.

4. **(2 points)** *Improving the FashionMNIST classifier.* Recall that in the first recitation, we trained a simple logistic regression model to classify MNIST digits. Repeat the same experiment, but now use a (dense) neural network with three (3) hidden layers with 256, 128, and 64 neurons respectively, all with ReLU activations. Display train- and test- loss curves, and report test accuracies of your final model. You may have to tweak the total number of training epochs to get reasonable accuracy. Finally, draw any 3 image samples from the test dataset, visualize the predicted class probabilities for each sample, and comment on what you can observe from these plots.

5. **(3 points)** *Implementing back-propagation in Python from scratch.* Open the (incomplete) Jupyter notebook provided as an attachment to this homework in Google Colab (or other Python IDE of your choice) and complete the missing items. In the second demo, we worked with autodiff. Autodiff enables us to implicitly store how to calculate the gradient when we call backward. We implemented some basic operations (addition, multiplication, power, and ReLU). In this homework problem, you will

implement backprop for more complicated operations directly. Instead of using autodiff, you will manually compute the gradient of the loss function for each parameter.