Homework 7

Math 198: Math for Machine Learning

Due Date:
Name:
Student ID:

Instructions for Submission

Please include your name and student ID at the top of your homework submission. You may submit handwritten solutions or typed ones (LaTeX preferred). If you at any point write code to help you solve a problem, please include your code at the end of the homework assignment, and mark which code goes with which problem. Homework is due by start of lecture on the due date; it may be submitted in-person at lecture or by emailing a PDF to both facilitators.

1 Practice with Differentiation

1. Let $\mathbf{A}$ be an arbitrary matrix and $\mathbf{v}, \mathbf{w}$ arbitrary vectors.
   (a) Let $f(\mathbf{v}) = \mathbf{A}\mathbf{v}$. Find $\mathbf{J}_f(\mathbf{v})$. What are the dimensions of $\mathbf{J}_f$?
   (b) Let $g(\mathbf{A}) = \mathbf{A}\mathbf{v}$. Find $\mathbf{J}_g(\mathbf{A})$. What are the dimensions of $\mathbf{J}_g$?
   (c) Let $h(\mathbf{v}) = \mathbf{v} + \mathbf{w}$. Find $\mathbf{J}_h(\mathbf{v})$.
2. Let $\mathbf{A} \in \mathbb{R}^{m \times k}$, $\mathbf{B} \in \mathbb{R}^{n \times m}$, and $\mathbf{v} \in \mathbb{R}^k$.
   (a) Let $f(\mathbf{v}) = \mathbf{A}\mathbf{v}$. Without explicitly computing it, what are the dimensions of $\mathbf{J}_f(\mathbf{v})$?
   (b) Let $g(\mathbf{v}) = \mathbf{B}\mathbf{A}\mathbf{v}$. Without explicitly computing it, what are the dimensions of $\mathbf{J}_g(\mathbf{v})$?
   (c) Make a dimensionality argument for the values of $\mathbf{J}_f(\mathbf{v})$ and $\mathbf{J}_g(\mathbf{v})$, again without computing them.
   (d) Compute $\mathbf{J}_g(\mathbf{v})$ and confirm your answer matches (c).
   (e) Let $\mathbf{C} \in \mathbb{R}^{n \times n}$, and let $h(\mathbf{v}) = \mathbf{C}\mathbf{B}\mathbf{A}\mathbf{v}$. What is $\mathbf{J}_h(\mathbf{v})$?

2 Gradient Descent

Note: The final question of this homework will require you to work with the attached file `hw7.ipynb` using Jupyter, a web-based Python development environment. This question is optional, but highly recommended unless you experience technical difficulties, as you'll only have to fill in a few lines of code. If you are familiar with the terminal and know which package manager you have installed, you should be able to just install jupyter from command-line, i.e. `pip install jupyter`. (Further installs are handled in the code file.) For a familiar "download-and-install" method, you can set up Anaconda, although this will also install a variety of unrelated packages. There are also a few online Jupyter servers available at Jupyter's website; select "Try Classic Notebook", then when your window launches, click File → Open, then Upload all the files and folders in the attached code folder.

In this problem, you will use Gradient Descent to train a neural network to recognize hand-written digits from the MNIST dataset. The images in this dataset are black-and-white and 28x28 pixels. Therefore, we can represent them as matrices:

$$\mathbf{x}_i = \begin{bmatrix} p_{1,1} & \cdots & p_{1,28} \\ \vdots & \ddots & \vdots \\ p_{28,1} & \cdots & p_{28,28} \end{bmatrix}$$

Note that $p_{j,k}$ represents the intensity of the pixel at position (j,k), which is clipped to the range $[0, 256]$. A value of 0 means the pixel is white, whereas 256 represents black, with the shades of grey in between. We then reshape these matrices into vectors, so each input is represented as a vector:

$$\mathbf{x}_i = \begin{bmatrix} p_{1,1} & \ldots & p_{1,28} & p_{2,1} & \ldots & p_{28,28} \end{bmatrix}^\top$$

Our neural network generates predictions using two layers of weights $\mathbf{W}_1, \mathbf{W}_2$ with biases $\mathbf{b}_1, \mathbf{b}_2$ and one non-linearity σ. The procedure is as follows:

$$\mathbf{h} = \mathbf{W}_1 \mathbf{x}_i + \mathbf{b}_1$$

$$\mathbf{h}' = \sigma(\mathbf{h})$$

$$\mathbf{y}_i' = \mathbf{W}_2 \mathbf{h}' + \mathbf{b}_2$$

The intermediate vectors $\mathbf{h}$ and $\mathbf{h}'$ have the same dimension, which is known as the *hidden dimension*[1]. Note that this dimension is a hyperparameter. Let's call this dimension h. Then $\mathbf{W}_1 \in \mathbb{R}^{h \times 28^2}$, $\mathbf{b}_1 \in \mathbb{R}^h$. Our output vector $\mathbf{y}_i'$ is ten-dimensional, with the indices representing the possible labels. Each entry in this vector can thus be interpreted as the network's "confidence" in the corresponding label. So $\mathbf{W}_2 \in \mathbb{R}^{10 \times h}, \mathbf{b}_2 \in \mathbb{R}^{10}$.

Before we get to the problems, let's cover a couple questions. First of all, why include biases? The biases are an example of basic feature augmentation (effectively we are adding an extra 1 to each input), which improves performance, as the biases can correct for bias in the input (i.e. most of the pixels are black, since the handwriting is white). Additionally, why use a nonlinearity? Without σ, our prediction would be

$$\mathbf{y}_i' = \mathbf{W}_2(\mathbf{W}_1 \mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2$$

which is linear in $\mathbf{x}_i$ and therefore our network wouldn't be able to learn arbitrary functions $f : \mathbb{R}^{28^2} \to \mathbb{R}^{10}$ like we'd like it to. Finally, why are our outputs vectors, and not just the class labels themselves? Consider what would happen if the network was equally sure that the image was of the number 1 and the number 7. What should it output? Ideally it could output a mix of those two labels and be penalized less than guessing completely wrong. However, it's not clear how the network could do this – it could output 4, which is halfway between 1 and 7, but this would still be a completely wrong prediction. Instead, we encode the labels $\mathbf{y}_i$ as *one-hot vectors*, which are 1 in the index corresponding to the label and 0 elsewhere. Our loss function is thus

$$L(\mathbf{y}_i') = ||\mathbf{y}_i - \mathbf{y}_i'||^2$$

which is minimized when the prediction is correct, but will still award "partial credit".

1. Derive $\frac{\partial L}{\partial \mathbf{y}_i'} = \nabla L(\mathbf{y}_i')$.
2. Derive $\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{y}_i'} \frac{\partial \mathbf{y}_i'}{\partial \mathbf{W}_2}$.
3. Derive $\frac{\partial L}{\partial \mathbf{b}_2}$.
4. Derive $\frac{\partial L}{\partial \mathbf{h}'} = \frac{\partial L}{\partial \mathbf{y}_i'} \frac{\partial \mathbf{y}_i'}{\partial \mathbf{h}'}$.

[1] More generally, the dimension of the *hidden layer*.

5. Our nonlinearity σ is the ReLU function, which is defined as:

$$\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ otherwise} \end{cases}$$

   When applied to a vector, ReLU operates element-wise. Derive $\frac{\partial L}{\partial \mathbf{h}}$.

6. Derive $\frac{\partial L}{\partial \mathbf{W}_1}$.

7. Derive $\frac{\partial L}{\partial \mathbf{b}_1}$.

8. Using the comments as guides, fill in the appropriate sections of `hw7.ipynb` with your weight updates. Using the preset learning rate, you should achieve x accuracy after n epochs (passes through the training set). Feel free to mess with the learning rate and hidden dimension to try to achieve better performance, although note that if your training accuracy is significantly higher than your validation accuracy, the neural net is likely overfitting to the training data.