Homework 7 Solutions

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 Taking Gradients

1. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a function defined by $f(\mathbf{x}) = \sin(x_1) + 2x_2^2$.
   (a) Find $\nabla f(\mathbf{x})$.
   $\nabla f(\mathbf{x}) = \begin{bmatrix}\cos(x_1) & 4x_2\end{bmatrix}^\top$
   (b) Find a critical point of f.
   $\begin{bmatrix}\frac{\pi}{2} & 0\end{bmatrix}^\top$ is a critical point.
   (c) Take another point $\mathbf{y}$ near the critical point you've found, and calculate $\nabla f(\mathbf{y})$. Does $\nabla f(\mathbf{y})$ point towards or away from your critical point? What does this imply about what type of critical point you've found?
   At $\begin{bmatrix}\frac{\pi}{2} & 1\end{bmatrix}^\top$, the value of the gradient is $\begin{bmatrix}0 & 4\end{bmatrix}^\top$, which points away from the critical point. This implies our critical point is a minima or a saddle point. We can confirm it is a saddle point by checking $\begin{bmatrix}-1 & 0\end{bmatrix}^\top$, for which the value of the gradient is $\begin{bmatrix}0.54 & 0\end{bmatrix}^\top$, pointing towards our critical point.

2. Let $\mathbf{v} = \begin{bmatrix}3 & -2\end{bmatrix}^\top$, and let $g(\mathbf{x}) = \mathbf{v}^\top \mathbf{x}$.
   (a) Find $\nabla g(\mathbf{x})$.
   By rewriting $g(\mathbf{x}) = 3x_1 - 2x_2$, it becomes clear that $\nabla g(\mathbf{x}) = \mathbf{v}$.
   (b) Does g have any critical points?
   Since $\nabla g(\mathbf{x})$ is a constant not dependent on $\mathbf{x}$, there is no value of $\mathbf{x}$ for which $\nabla g(\mathbf{x}) = \mathbf{0}$, so g has no critical points.

2 Taking Jacobians

1. Let $f : \mathbb{R}^3 \to \mathbb{R}^2$ be a function defined by $f(\mathbf{x}) = \begin{bmatrix}2x_1^2 + 3x_2 + 5 & \sin(x_3) - e^{x_1}\end{bmatrix}^\top$.
   (a) What are the dimensions of $\mathbf{J}_f$?
   $\mathbf{J}_f$ will be a 2×3 matrix.

(b) Compute $\mathbf{J}_f$.

$\mathbf{J}_f = \begin{bmatrix} 4x_1 & 3 & 0 \\ -e^{x_1} & 0 & \cos(x_3) \end{bmatrix}$

2. Let $\mathbf{A}$ be an arbitrary $n \times d$ matrix and $\mathbf{v}, \mathbf{w}$ arbitrary d-dimensional vectors.

   (a) Let $g(\mathbf{v}) = \mathbf{A}\mathbf{v}$. Find $\mathbf{J}_g(\mathbf{v})$. What are the dimensions of $\mathbf{J}_g$?
   $\mathbf{A}$ represents a linear map $g : \mathbb{R}^d \to \mathbb{R}^n$, so $\mathbf{J}_g$ will be another $n \times d$ matrix; in fact, $\mathbf{J}_g = \mathbf{A}$.

   (b) Let $h(\mathbf{v}) = \mathbf{v} + \mathbf{w}$. Find $\mathbf{J}_h(\mathbf{v})$.
   $\mathbf{J}_h = \mathbf{1}$.

3 Extending the Gradient

We can extend the notion of the gradient to functions $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ which map matrices to scalars. Let $\mathbf{A}$ be an arbitrary $m \times n$ matrix, and $\mathbf{v}$ be an n-dimensional vector. First, rewrite $\mathbf{A}$ as a vector $\mathbf{a}$ such that $A_{ij} = a_k$ where $k = (i-1)n + j$. Then let $g : \mathbb{R}^{mn} \to \mathbb{R}$ such that $f(\mathbf{A}) = g(\mathbf{a})$. We can then define $\nabla f(\mathbf{A})$ to be a matrix with entries $\nabla f(\mathbf{A})_{ij} = \nabla g(\mathbf{a})_k$.

1. Let $f(\mathbf{A}) = \mathbf{v}^\top \mathbf{A}\mathbf{v}$. Find $\nabla f(\mathbf{A})$.
   We have $f(\mathbf{A}) = \sum_{i=1}^n \sum_{j=1}^n v_i A_{ij} v_j$, so $\nabla f(\mathbf{A})_{ij} = v_i v_j$.

2. Let $f(\mathbf{A}) = ||\mathbf{A}\mathbf{v}||_2^2 = \sum_{i=1}^m (Av)_i^2$.

   a. Rewrite $f(\mathbf{A})$ as a vectorized function $g(\mathbf{a})$.
   $g(\mathbf{a}) = \sum_{i=1}^m \left(\sum_{j=1}^n a_{(i-1)n+j} v_j \right)^2$

   b. Express g as the composition of two functions $g_1 : \mathbb{R}^m \to \mathbb{R}$ and $g_2 : \mathbb{R}^{mn} \to \mathbb{R}^m$ such that $g = g_1 \circ g_2$.

$$g_1(\mathbf{x}) = \sum_{i=1}^m x_i^2$$

$$g_2(\mathbf{a})_i = \sum_{j=1}^n a_{(i-1)n+j} v_j$$

   c. Using the chain rule, compute $\nabla g(\mathbf{a})$, and use it to derive $\nabla f(\mathbf{A})$.
   We have

$$\nabla g_1(\mathbf{x}) = 2\mathbf{x}$$

   and, noting that $g_2 : \mathbb{R}^{mn} \to \mathbb{R}^m$, then for $k' \in mn$

$$J_{g_2}(\mathbf{a})_{ik'} = \frac{\partial g_{2i}}{\partial a_{k'}} = \begin{cases} v_{k' \bmod n} & \text{if } \lfloor \frac{k'}{n} \rfloor = i - 1 \\ 0 & \text{otherwise} \end{cases}$$

   so using the chain rule $\nabla(g_1 \circ g_2)(\mathbf{a}) = \mathbf{J}_{g_2}(\mathbf{a})^\top \nabla g_1(g_2(\mathbf{a}))$ we have

$$\begin{aligned} \nabla g(\mathbf{a}) &= \mathbf{J}_{g_2}(\mathbf{a})^\top \nabla g_1(g_2(\mathbf{a})) \\ &= 2\mathbf{J}_{g_2}(\mathbf{a})^\top g_2(\mathbf{a}) \\ \nabla g(\mathbf{a})_{k'} &= 2\sum_{i=1}^m \mathbf{J}_{g_2}(\mathbf{a})_{ik'} g_2(\mathbf{a})_i \\ &= 2\sum_{i=1}^m \frac{\partial g_{2i}}{\partial a_{k'}} \sum_{j=1}^n a_{(i-1)n+j} v_j \end{aligned}$$

There is only one value of i for which $\lfloor \frac{k'}{n} \rfloor = i - 1$, which we will denote i^*. Therefore

$$\nabla g(\mathbf{a})_{k'} = 2v_{k' \bmod n} \sum_{j=1}^{n} a_{(i^*-1)n+j} v_j$$

We can now translate this back to the original function f by mapping $i = \lfloor \frac{k'}{n} \rfloor + 1$ and $j = k' \bmod n$ and setting

$$\nabla f(\mathbf{A})_{ij} = 2v_j \mathbf{A}_i^\top \mathbf{v}$$

where $\mathbf{A}_i^\top$ denotes the i-th row of $\mathbf{A}$.

4 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. The code component 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:

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

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:

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

Our neural network generates predictions using two layers of weights $\mathbf{V}, \mathbf{W}$ with biases $\mathbf{b}, \mathbf{c}$ and one nonlinearity σ. The procedure is as follows:

$$\mathbf{h} = \mathbf{V}\mathbf{x} + \mathbf{b}$$

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

$$\mathbf{y}' = \mathbf{W}\mathbf{h}' + \mathbf{c}$$

The intermediate vectors $\mathbf{h}$ and $\mathbf{h}'$ have the same dimension, which is known as the *hidden dimension*. Note that this dimension is a hyperparameter. Let's call this dimension d. Then $\mathbf{V} \in \mathbb{R}^{d \times 28^2}$, $\mathbf{b} \in \mathbb{R}^d$. Our output vector $\mathbf{y}'$ 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 $\mathbf{W} \in \mathbb{R}^{10 \times d}, \mathbf{c} \in \mathbb{R}^{10}$.

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

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

which is linear in $\mathbf{x}$ 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 $\mathbf{y}$ as *one-hot vectors*, which are 1 in the index corresponding to the label and 0 elsewhere. Our loss function is thus

$$L(\mathbf{y}') = ||\mathbf{y} - \mathbf{y}'||^2 = \sum_{i=1}^{10} (y_i - y_i')^2$$

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

1. Derive $\nabla L(\mathbf{y})$.
   We can rewrite $L(\mathbf{y}') = \sum_{i=1}^{10}(y_i - y_i')^2 = \sum_{i=1}^{10} y_i^2 - 2y_i y_i' + y_i'^2$, and so the i-th entry of $\nabla L(\mathbf{y}')$ is given by $2y_i' - 2y_i$. Therefore $\nabla L(\mathbf{y}') = 2(\mathbf{y}' - \mathbf{y})$.

2. Derive $\nabla L(\mathbf{W})$.
   Similar to 3.2, we will use the chain rule, defining

$$l_1(\mathbf{x}) = \sum_{i=1}^{10} (y_i - x_i)^2$$

$$\nabla l_1(\mathbf{x}) = 2(\mathbf{y} - \mathbf{x})$$

   and defining an invertible mapping $o(i, j) = k$ for $i \in [1, 10]$, $j \in [1, d]$, $k \in [1, 10d]$ to vectorize $\mathbf{W} \in \mathbb{R}^{j \times m}$ as $\mathbf{w} \in \mathbb{R}^k$,

$$l_2(\mathbf{w})_i = y_i' = c_i + \sum_{j=1}^{d} w_{o(i,j)} h_j'$$

$$\mathbf{J}_{l_2}(\mathbf{w})_{ik} = \frac{\partial l_{2i}}{\partial w_k} = \begin{cases} h_j' & \text{if } o(i,j) = k \\ 0 & \text{otherwise} \end{cases}$$

   We then apply the chain rule:

$$\begin{aligned} \nabla L(\mathbf{w}) &= \mathbf{J}_{l_2}(\mathbf{w})^\top \nabla l_1(l_2(\mathbf{w})) \\ &= 2\mathbf{J}_{l_2}(\mathbf{w})^\top (\mathbf{y} - l_2(\mathbf{w})) \\ &= 2\mathbf{J}_{l_2}(\mathbf{w})^\top (\mathbf{y} - \mathbf{y}') \\ \nabla L(\mathbf{w})_k &= 2\sum_{i=1}^{10} \mathbf{J}_{l_2}(\mathbf{w})_{ik}(y_i - y_i') \\ &= 2\sum_{i=1}^{10} \frac{\partial l_{2i}}{\partial w_k}(y_i - y_i') \end{aligned}$$

   Since o is invertible, there are unique values i^* and j^* such that $k = o(i^*, j^*)$. Only $\frac{\partial l_{2i^*}}{\partial w_k}$ will be non-zero, so we have

$$\nabla L(\mathbf{w})_k = 2h_{j^*}'(y_{i^*} - y_{i^*}')$$

   De-vectorizing this, we arrive at $\nabla L(\mathbf{W})_{ij} = 2h_j'(y_i - y_i') = h_j' \nabla L(\mathbf{y}')_i$.

3. Derive $\nabla L(\mathbf{c})$.
   Our definition of l_1 is the same as in the last problem. We do not need to vectorize $\mathbf{c}$, as it is already

a vector; so we have

$$\begin{aligned}
l_2(\mathbf{c})_i &= y'_i = c_i + \sum_{j=1}^{d} W_{ij} h'_j \\
\mathbf{J}_{l_2}(\mathbf{c}) &= \mathbf{I}_{10} \\
\nabla L(\mathbf{c}) &= \mathbf{J}_{l_2}(\mathbf{c})^\top \nabla l_1(l_2(\mathbf{c})) \\
&= \nabla l_1(\mathbf{y}') \\
&= \nabla L(\mathbf{y}')
\end{aligned}$$

4. Derive $\nabla L(\mathbf{h}')$.
   $l_2(\mathbf{h}') = \mathbf{y}' = \mathbf{W}\mathbf{h}' + \mathbf{c}$ so we have $\mathbf{J}_{l_2}(\mathbf{h}') = \mathbf{W}$. Therefore,

$$\nabla L(\mathbf{h}') = \mathbf{W}^\top \nabla l_1(\mathbf{y}') = \mathbf{W}^\top \nabla L(\mathbf{y}')$$

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 $\nabla L(\mathbf{h})$.
   We now define $l_1(\mathbf{h}') = L(\mathbf{h}')$ (since we know its gradient already) and $l_2(\mathbf{h}) = \sigma(\mathbf{h})$. Then $\mathbf{J}_{l_2}(\mathbf{h})$ is a diagonal matrix with ones on the diagonal where the corresponding entries in $\mathbf{h}$ are positive, and zeroes otherwise. Let's use $\mathbf{\Sigma_x}$ to denote a matrix with this property for some vector $\mathbf{x}$. So $\mathbf{J}_{l_2}(\mathbf{h}) = \mathbf{\Sigma_h}$, and $\nabla L(\mathbf{h}) = \mathbf{\Sigma_h} \nabla l_1(\mathbf{h}') = \mathbf{\Sigma_h} \nabla L(\mathbf{h}')$.

6. Derive $\nabla L(\mathbf{V})$.
   Let $l_1(\mathbf{h}) = L(\mathbf{h})$ and $l_2(\mathbf{V}) = \mathbf{h} = \mathbf{V}\mathbf{x} + \mathbf{b}$. Following a similar process to our derivation of $\nabla L(\mathbf{W})$, we arrive at $\nabla L(\mathbf{V})_{ij} = x_j \nabla l_1(\mathbf{h})_i = x_j \nabla L(\mathbf{h})_i$.

7. Derive $\nabla L(\mathbf{b})$.
   Similar to $\nabla L(\mathbf{c})$ we have $\nabla L(\mathbf{b}) = \nabla L(\mathbf{h})$.

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.