COMS 4771 Machine Learning *Fall 2025*

Homework 6

1 Gradient descent

Consider the OLS objective $J(w) = \|Aw - b\|^2$, where the SVD of $A \in \mathbb{R}^{n \times d}$ is $A = \sum_{i=1}^{r} \sigma_i u_i v_i^\mathsf{T}$ for some $r \geq 1$, and $b \in \mathbb{R}^n$. Let $w^{(1)}, w^{(2)}, \ldots$ denote the iterates produced by gradient descent with J as the objective function and $\eta > 0$ as the step size, starting from $w^{(0)} = 0$.

Assume the largest singular value of A is $\sigma_1 = 1$.

1. What is behavior of $v_1^\mathsf{T} w^{(t)}$ as t increases when $\eta < 0.5$?
2. What is behavior of $v_1^\mathsf{T} w^{(t)}$ as t increases when $\eta = 0.5$?
3. What is behavior of $v_1^\mathsf{T} w^{(t)}$ as t increases when $0.5 < \eta < 1$?

In the three cases above, the asymptotic behavior is the same, but the sequences themselves are qualitatively different.

2 Straight-line program

Consider the two-layer neural network function $f_\theta \colon \mathbb{R}^2 \to \mathbb{R}$ given by

$$f_\theta(x) := C\sigma(Ax + b) + d, \quad \text{for all } x \in \mathbb{R}^2,$$

where $\theta = (A, b, C, d) \in \mathbb{R}^{2\times 2} \times \mathbb{R}^2 \times \mathbb{R}^{1\times 2} \times \mathbb{R}^1$ are the parameters of the function, and $\sigma \colon \mathbb{R} \to \mathbb{R}$ is the logistic function $\sigma(z) := \text{logistic}(z)$, which is applied coordinate-wise to vector arguments. Write a straight-line program for computing f_θ on a given input $x = (x_1, x_2)$, using only the following standard library functions:

- `logistic(x)` which computes the logistic function on its argument
- `prod(x,y)` which computes the product of its two arguments
- `sum(x,y)` which computes the sum of its two arguments

Name your variables `v1`, `v2`, etc.

Now consider the computation graph associated with the straight-line program. Write two distinct topological orderings of the nodes (ignoring the nodes corresponding to the inputs and parameters).

3 Automatic differentiation

Consider the following straight-line program for a function $F\colon \mathbb{R} \to \mathbb{R}$:

```
v1 := square(v0)
v2 := exp(v1)
v3 := prod(v1,v2)
v4 := sum(v2,v3)
```

Assume the standard library provides the functions:

- `square(x)` computes the square of x, with partial derivative $\frac{\partial}{\partial x}\,\mathrm{square}(x) = 2x$
- `exp(x)` computes e^x, with partial derivative $\frac{\partial}{\partial x}\exp(x) = \exp(x)$
- `prod(x,y)` computes $x \times y$, with partial derivatives $\frac{\partial}{\partial x}\,\mathrm{prod}(x,y) = y$, $\frac{\partial}{\partial y}\,\mathrm{prod}(x,y) = x$
- `sum(x,y)` computes $x + y$, with each partial derivative equal to 1

1. Draw the computation graph for this straight-line program.
2. Suppose the input is $v_0 = 1/2$. Write out the transcript of the computations in the backward pass, showing the partial derivatives of F with respect to each variable $(v_4, v_3, v_2, v_1, v_0)$.
3. Repeat the previous part with $v_0 = -1$.

4 Neural networks

For this problem, you will train neural networks on synthetic and OCR data using PyTorch. Template code (`hw6-template.py`) is provided on Courseworks. A PyTorch tutorial is available here: `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`

4.1 One-layer neural network on XOR data

First, start with a one-layer neural network $f_\theta\colon \mathbb{R}^2 \to \mathbb{R}$ given by

$$f_\theta(x) := W_1 x + b_1, \quad \text{for all } x \in \mathbb{R}^2,$$

where $\theta = (W_1, b_1) \in \mathbb{R}^{1\times 2} \times \mathbb{R}^1$ are the parameters of the function. (Use an identity activation function for the output node.) This is just an affine function! Use it as a binary classifier via

$$x \mapsto \mathbb{1}\{f_\theta(x) > 0\}.$$

Implement such a neural network in PyTorch, as well as a gradient descent procedure to fit the parameters to the training data $(x_1, y_1), \ldots, (x_4, y_4) \in \mathbb{R}^2 \times \{0, 1\}$,

$$\begin{aligned}(x_1, y_1) &= ((-1, -1), 0),\\ (x_2, y_2) &= ((+1, -1), 1),\\ (x_3, y_3) &= ((-1, +1), 1),\\ (x_4, y_4) &= ((+1, +1), 0).\end{aligned}$$

This is the "XOR" pattern previously given as an example of a non-linearly separable data set. This data set is provided by the `XOR_data()` function in the template code. Run 25 iterations of gradient descent with step size $\eta = 1.0$ on the empirical logistic loss risk:

$$J(\theta) = \frac{1}{4}\sum_{i=1}^{4}\Big(y_i \ln(1 + \exp(-f_\theta(x_i))) + (1 - y_i)\ln(1 + \exp(f_\theta(x_i)))\Big).$$

In PyTorch, this objective function is implemented as `torch.nn.BCEWithLogitsLoss`. Use the default PyTorch initialization scheme to set the initial parameters for gradient descent, but do so *immediately after* setting the random number seed using `torch.manual_seed(0)`. (This will help reproducibility!) Record the objective values and error rates prior to training, and also after every iteration of gradient descent. Report the initial and final values of each, and separately plot the objective values and error rates as a function of the iteration number.

4.2 Two-layer neural network on XOR data

Next, consider the two-layer neural network function $f_\theta \colon \mathbb{R}^2 \to \mathbb{R}$ given by

$$f_\theta(x) := W_2\sigma(W_1 x + b_1) + b_2, \quad \text{for all } x \in \mathbb{R}^2,$$

where $\theta = (W_1, b_1, W_2, b_2) \in \mathbb{R}^{2\times 2} \times \mathbb{R}^2 \times \mathbb{R}^{1\times 2} \times \mathbb{R}^1$ are the parameters of the function, and $\sigma \colon \mathbb{R} \to \mathbb{R}$ is the ReLU function $\sigma(z) := \max\{0, z\}$, which is applied coordinate-wise to vector arguments. (Again, use an identity activation function for the output node.)

Implement this network in PyTorch and repeat the tasks from the previous part.

4.3 Three-layer neural network on handwritten digits

Next, consider a three-layer neural network $f_\theta \colon \mathbb{R}^{64} \to \mathbb{R}$ with the following architecture:

- Layer 1: fully-connected (linear) layer with 64 inputs and 64 outputs, with a bias term. Use the ReLU activation function.
- Layer 2: fully-connected (linear) layer with 64 inputs and 32 outputs, with a bias term. Use the ReLU activation function.
- Layer 3: fully-connected (linear) layer with 32 inputs and 1 output, with a bias term. Use identity (i.e., no) activation function.

In other words,

$$f_\theta(x) := W_3\sigma(W_2\sigma(W_1x + b_1) + b_2) + b_3, \quad \text{for all } x \in \mathbb{R}^{64},$$

where $\theta = (W_1, b_1, W_2, b_2, W_3, b_3) \in \mathbb{R}^{64\times64} \times \mathbb{R}^{64} \times \mathbb{R}^{32\times64} \times \mathbb{R}^{32} \times \mathbb{R}^{1\times32} \times \mathbb{R}^1$ are the parameters of the function, and $\sigma\colon \mathbb{R} \to \mathbb{R}$ is the ReLU function $\sigma(z) := \max\{0, z\}$, which is applied coordinate-wise to vector arguments.

Implement this network in PyTorch and repeat the tasks from the previous part with a few changes.

- Instead of the XOR data set, use the "digits" data set that is prepared using the `digits_data()` function in the template code. This data set is obtained using scikit-learn: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html`. The first 180 examples are reserved as test examples; the remaining 1617 examples are used as training examples. The binary label to predict is 1 for odd digits, and 0 for even digits.
- You will have to "reshape" the inputs, as each image is given as an 8×8 array, but your network takes 64-dimensional vectors as inputs. (The template code shows how to do this.)
- In your gradient descent implementation, use a step size of $\eta = 0.1$ (instead of 1.0), and use 500 iterations (instead of 25).
- Also report the *test* error rate of the network using the final parameters.

5 Calibration

Let (X, Y) be a random example taking values in $\mathbb{R}^d \times \{0, 1\}$. Suppose $\hat{p}\colon \mathbb{R}^d \to [0, 1]$ is perfectly calibrated: i.e., $\Pr(Y = 1 \mid \hat{p}(X) = p) = p$ for all p in the range of $\hat{p}$. Is it true that $\hat{p}$ is also *marginally calibrated*? (Here, by *marginally calibrated*, we mean that $\hat{p}$ satisfies $\mathbb{E}[\hat{p}(X)] = \Pr(Y = 1)$.) Explain your answer.