EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Homework 2

This homework is due on Saturday, September 17, 2022, at 10:59PM.

1. Why learning rates cannot be too big

To understand the role of the learning rate, it is useful to understand it in the context of the simplest possible problem first.

Suppose that we want to solve the scalar equation

$$\sigma w = y \tag{1}$$

where we know that $\sigma > 0$. We proceed with an initial condition $w_0 = 0$ by using gradient descent to minimize the squared loss

$$L(w) = (y - \sigma w)^2 \tag{2}$$

which has a derivative with respect to the parameter w of $-2\sigma(y - \sigma w)$.

Gradient descent with a learning rate of η follows the recurrence-relation or discrete-time state evolution of:

$$\begin{aligned} w_{t+1} &= w_t + 2\eta\sigma(y - \sigma w_t) \\ &= (1 - 2\eta\sigma^2)w_t + 2\eta\sigma y. \end{aligned} \tag{3}$$

(a) **For what values of learning rate $\eta > 0$ is the recurrence (3) stable?**

(HINT: Remember the role of the unit circle in determining the stability or instability of such recurrences. If you keep taking higher and higher positive integer powers of a number, what does that number have to be like for this to converge?)

Solution: We can rewrite the update rule as

$$\begin{aligned} w_{t+1} - \frac{y}{\sigma} &= (1 - 2\eta\sigma^2)(w_t - \frac{y}{\sigma}) \\ w_{t+1} &= \frac{y}{\sigma} + (1 - 2\eta\sigma^2)^{t+1}(w_0 - \frac{y}{\sigma}) \end{aligned} \tag{4}$$

To make the recurrence (3) stable with $\eta > 0$, we need $|1 - 2\eta\sigma^2| < 1$. This gives $\eta < \frac{1}{\sigma^2}$.

(b) The previous part gives you an upper bound for the learning rate η that depends on σ beyond which we cannot safely go. **If η is below that upper bound, how fast does w_t converge to its final solution $w^* = \frac{y}{\sigma}$? i.e. If we wanted to get within a factor $(1 - \epsilon)$ of w^*, how many iterations t would we need?**

Solution:

$$|w_T - w^*| < \epsilon|w^*| \tag{5}$$

Use the derived update rule in (4). We have

$$
\begin{aligned}
|w_T - \frac{y}{\sigma}| &< \epsilon|\frac{y}{\sigma}| \\
|(1-2\eta\sigma^2)^T| &< \epsilon \\
T &> \frac{log(\epsilon)}{log(|1-2\eta\sigma^2|)},
\end{aligned} \tag{6}
$$

(c) Suppose that we now have a vector problem where we have two parameters $w[1], w[2]$. One with a large σ_ℓ and the other with a tiny σ_s. i.e. $\sigma_\ell \gg \sigma_s$ and we have the vector equation we want to solve:

$$
\begin{bmatrix} \sigma_\ell & 0 \\ 0 & \sigma_s \end{bmatrix} \begin{bmatrix} w[1] \\ w[2] \end{bmatrix} = \begin{bmatrix} y[1] \\ y[2] \end{bmatrix}. \tag{7}
$$

We use gradient descent with a single learning rate η to solve this problem starting from an initial condition of $\mathbf{w} = \mathbf{0}$.

For what learning rates $\eta > 0$ will we converge? Which of the two σ_i is limiting our learning rate?

Solution: Similarly, we can rewrite the loss function and update rule w.r.t the vector form.

$$
\begin{aligned}
L(\mathbf{w}) &= ||\mathbf{y} - \Sigma\mathbf{w}||^2, \Sigma = \begin{bmatrix} \sigma_\ell & 0 \\ 0 & \sigma_s \end{bmatrix} \\
\nabla_{\mathbf{w}} L(\mathbf{w}) &= 2(\Sigma^2\mathbf{w} - \Sigma\mathbf{y}) \\
\mathbf{w}_{t+1} &= (I - 2\eta\Sigma^2)\mathbf{w}_t + 2\eta\Sigma\mathbf{y}
\end{aligned} \tag{8}
$$

To ensure the convergence, we need

$$
\begin{cases} |1-2\eta\sigma_l^2| < 1 \\ |1-2\eta\sigma_s^2| < 1 \end{cases} \tag{9}
$$

$$
\eta < \min(\frac{1}{\sigma_l^2}, \frac{1}{\sigma_s^2}) = \frac{1}{\sigma_l^2} \tag{10}
$$

(d) **For the previous problem, depending on $\eta, \sigma_\ell, \sigma_s$, which of the two dimensions is converging faster and which is converging slower?**

Solution: We can rewrite the update rule w.r.t each dimension, this gives

$$
\begin{aligned}
w[1]_t &= \frac{y[1]}{\sigma_l} + (1-2\eta\sigma_l^2)^t(-\frac{y[1]}{\sigma_l}) \\
w[2]_t &= \frac{y[2]}{\sigma_s} + (1-2\eta\sigma_s^2)^t(-\frac{y[2]}{\sigma_s})
\end{aligned}
$$

This faster convergence dimension is $\max(|1-2\eta\sigma_l^2|, |1-2\eta\sigma_s^2|)$

(e) The speed of convergence overall will be dominated by the slower of the two. **For what value of η will we get the fastest overall convergence to the solution?**

Solution: $\eta = \frac{1}{\sigma_l^2+\sigma_s^2}$

(f) Comment on what would happen if we had more parallel problems with σ_i that all were in between σ_ℓ and σ_s? **Would they influence the choice of possible learning rates or the learning rate with the fastest convergence?**

Solution: The bounds on the learning rate should still be the same. For maximal convergence, the largest and smallest values of sigma are the two that cause gradient descent to take the longest if we poorly choose a learning rate.

(g) Using what you know about the SVD, **how is the simple scalar and parallel scalar problem analysis above relevant to solving general least-squares problems of the form $X\mathbf{w} \approx \mathbf{y}$ using gradient descent?**

Solution: We can think of SVD as a change of bases into and then back from a coordinate system where X instead is just diagonal, this will directly connect to the same problem with lots of parallel scalar problems.

2. Step size in Gradient Descent

In this problem, we will look at the convex function $f(\mathbf{x}) = ||\mathbf{x} - \mathbf{b}||_2$. Note that we are using "just" the regular Euclidean ℓ_2 norm, *not* the norm squared! This problem illustrates the importance of understanding how gradient descent works and choosing step sizes strategically. In fact, there is a lot of active research in variations on gradient descent. Throughout the question we will look at different kinds of step sizes. We will also look at the the rate at which the different step sizes decrease and draw some conclusions about the rate of convergence. You have been provided with a tool in `hw2/step_size_helper.py` which will help you visualize the problems below.

(a) Let $\mathbf{x}, \mathbf{b} \in \mathbb{R}^d$. **Prove that $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{b}\|_2$ is a convex function of $\mathbf{x}$.**

Solution:

Recall that a function is convex if for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$ we have

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

for all $0 \leq \lambda \leq 1$. For $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{b}\|_2$, the triangle inequality gives us:

$$\begin{aligned}\|\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 - \mathbf{b}\|_2 &= \|\lambda(\mathbf{x}_1 - \mathbf{b}) + (1-\lambda)(\mathbf{x}_2 - \mathbf{b})\|_2 \\ &\leq \|\lambda(\mathbf{x}_1 - \mathbf{b})\|_2 + \|(1-\lambda)(\mathbf{x}_2 - \mathbf{b})\|_2 \\ &= \lambda\|\mathbf{x}_1 - \mathbf{b}\|_2 + (1-\lambda)\|\mathbf{x}_2 - \mathbf{b}\|_2,\end{aligned}$$

which shows that f is convex.

(b) **For $\nabla_\mathbf{x} f(\mathbf{x})$, determine where it is well-defined and compute its value.**

Solution:

$$\begin{aligned}\frac{\partial}{\partial x_i} f(\mathbf{x}) &= \frac{\partial}{\partial x_i}\sqrt{\sum_{i=1}^{d}(x_i - b_i)^2} \\ &= (x_i - b_i)\left(\sum_{i=1}^{d}(x_i - b_i)^2\right)^{-1/2} \\ &= \frac{x_i - b_i}{\|\mathbf{x} - \mathbf{b}\|_2}.\end{aligned}$$

Note that this is only well-defined when $\mathbf{x} \neq \mathbf{b}$. Thus $\nabla_{\mathbf{x}} f(\mathbf{x}) = \frac{\mathbf{x}-\mathbf{b}}{\|\mathbf{x}-\mathbf{b}\|_2}$ when $\mathbf{x} \neq \mathbf{b}$.

(c) We are minimizing $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{b}\|_2$, where $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{b} = [6, 4.5] \in \mathbb{R}^2$, with gradient descent. We use a constant step size of $t_i = 1$. That is,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - t_i \nabla f(\mathbf{x}_i) = \mathbf{x}_i - \nabla f(\mathbf{x}_i).$$

We start at $\mathbf{x}_0 = [0, 0]$. **Will gradient descent find the optimal solution? If so, how many steps will it take to get within** 0.01 **of the optimal solution (for this and subsequent problems, this is in terms of** $\|\mathbf{x_k} - \mathbf{x}_*\|_2$**)? If not, why not?** Prove your answer. (Hint: use the tool to compute the first ten steps.) **What about general** $\mathbf{b} \neq \mathbf{0}$**?**

Solution: Using the tool provided (or computing gradients by hand), we get

$$\mathbf{x}_6 = \mathbf{x}_8 = [5.6, 4.2]$$

and

$$\mathbf{x}_7 = \mathbf{x}_9 = [6.4, 4.8].$$

Examine the formula

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \nabla f(\mathbf{x}_i)$$

and notice that $\mathbf{x}_{i+1}$ only depends on $\mathbf{x}_i$, regardless of what step we are on (this is only the case because we are using a constant step size). It means that if $\mathbf{x}_i = \mathbf{x}_j$, then $\mathbf{x}_{i+1} = \mathbf{x}_{j+1}$. Thus, this pattern will repeat indefinitely and we do not reach the optimal solution. It is also helpful to notice that

$$\|\mathbf{x}_6\|, \|\mathbf{x}_7\| \in \mathbb{Z}.$$

In general, notice that

$$-\nabla f(\mathbf{x}_i) = \frac{\mathbf{b} - \mathbf{x}_i}{\|\mathbf{x}_i - \mathbf{b}\|}$$

will always have unit length. In fact, we can prove by induction that $\mathbf{x}_k$ is always an integer multiple of the unit vector $\frac{\mathbf{b}}{\|\mathbf{b}\|}$. Thus,

$$\|\mathbf{x}_k\| \in \mathbb{Z}.$$

If we are lucky and $\|\mathbf{b}\|$ happens to be an integer (or very close to integer), then we will reach or get near the optimal solution. Otherwise, our step size is too coarse to hit it.

In fact, for the function $\|\mathbf{x} - \mathbf{b}\|$, a constant step size will rarely work. We are too prone to "jumping over" our solution. A decreasing step size is necessary and we will investigate decreasing stepsizes next.

(d) We are minimizing $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{b}\|_2$, where $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{b} = [6, 4.5] \in \mathbb{R}^2$, now with a decreasing step size of $t_i = \frac{1}{i+1}$ at step i. That is,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - t_i \nabla f(\mathbf{x}_i) = \mathbf{x}_i - \frac{1}{i+1} \nabla f(\mathbf{x}_i).$$

We start at $\mathbf{x}_0 = [0, 0]$. **Will gradient descent find the optimal solution? If so, how many steps will it take to get within** 0.01 **of the optimal solution? If not, why not?** Prove your answer. (Hint: examine $\|\mathbf{x}_i\|_2$, and use $\sum_{i=1}^{n} \frac{1}{i}$ is of the order $\log n$.) **What about general** $\mathbf{b} \neq \mathbf{0}$**? State (roughly—up to the correct order) the amount of steps required to reach the optimum in the general case, for whichever cases are possible.**

Solution: The key realization here is that all our gradient steps have $\|\nabla f(\mathbf{x}_i)\| = 1$ and move along the same vector towards $\mathbf{b}$ (to be precise, we could prove by induction that our $\mathbf{x}_i$ and $\nabla f(\mathbf{x}_i)$ will always be in the span of $\mathbf{b}$). We can write

$$\|\mathbf{x}_{i+1}\| = \|\mathbf{x}_0 - \sum_{j=0}^{i} (\frac{1}{j+1})\nabla f(\mathbf{x}_i)\| = \sum_{j=0}^{i} \|(\frac{1}{j+1})\nabla f(\mathbf{x}_i)\| = \sum_{j=1}^{i} \frac{1}{j+1} \approx \ln i.$$

as long as $\|\mathbf{x}_j\|_2 \leq \|b\|_2$ for all $j \leq i$. As soon as this condition is violated, the iterates $\mathbf{x}_i$ oscillate around b, getting closer after each iteration because the sum of the step sizes diverges and the norm of the gradients is always one.

This sum diverges, so we will reach the optimal solution $\mathbf{b}$ eventually, but it could take quite a while. For $\mathbf{b} = [6, 4.5]$ it takes about 1000 steps (1004 to get within $\|\cdot\| \leq 0.01$ of the optimal $\mathbf{x}$—7.49046, to be exact). In generally, the number of steps it takes will be exponential in $\|\mathbf{b}\|$, since we have

$$\ln i \approx \|\mathbf{b}\| \implies i \approx e^{\|\mathbf{b}\|}.$$

We may not hit $\mathbf{b}$ exactly. Once we exceed $\mathbf{b}$, we start oscillating around it. Let i_0 be the first step where we "cross over" $\mathbf{b}$. After that, we know we are always moving towards $\mathbf{b}$ (whether we are stepping backwards or forwards), so after step i (assuming $i > i_0$), we can be at most $\frac{1}{i+1}$ from the optimal solution (if we are more than $\frac{1}{i+1}$ away, that means our previous step went in the wrong direction).

3. **Visualizing features from local linearization of neural nets**

This problem expects you to modify the Jupyter Notebook you were given in the first discussion section for the course to allow the visualization of the effective "features" that correspond to the local linearization of the network in the neighborhood of the parameters.

(a) **Augment the Jupyter notebook to allow for the visualization of the features corresponding to $\frac{\partial}{\partial w_i^{(1)}} y(x)$ and $\frac{\partial}{\partial b_i^{(1)}} y(x)$ where $w_i^{(1)}$ are the first hidden layer's weights and the $b_i^{(1)}$ are the first hidden layer's biases.** These derivatives should be evaluated at at least both the random initialization and the final trained network. When visualizing these features, plot them as a function of the scalar input x, the same way that the notebook plots the constituent "elbow" features that are the outputs of the penultimate layer.

(HINT: You are going to want to leverage PyTorch's autograd functionality here.)

(b) During training, we can imagine that we have a generalized linear model with a feature matrix corresponding to the linearized features corresponding to each learnable parameter. We know from our analysis of gradient descent, that the singular values and singular vectors corresponding to this feature matrix are important.

Use the SVD of this feature matrix to plot both the singular values and visualize the "principle features" that correspond to the d-dimensional singular vectors multiplied by all the features corresponding to the parameters.

(HINT: Remember that the feature matrix whose SVD you are taking has n rows where each row corresponds to one training point and d columns where each column corresponds to each of the learnable features. Meanwhile, you are going to be plotting/visualizing the "principle features" as functions of x even at places where you don't have training points.)

(c) With a different random initialization for the network, all the individual features will change quite significantly. Visualize how the principle features change with different random initializations as well as when the network has converged. **When the network has a significant number of hidden units, do the principle features change more or less than any individual specific feature?'**

(d) Augment the jupyter notebook to add a second hidden layer of the same size as the first hidden layer, fully connected to the first hidden layer using He initialzation for the initial condition of the weights and Xavier intialization for the initial condition of the biases. **Enhance the code you have already written to allow the visualization of the features corresponding to the parameters in both hidden layers, as well as the "principle features" and the singular values.**

(e) (Optional) **Make the code work where the number of hidden layers is a parameter. What do you see with deeper networks of fully-connected layers?**

4. Coding Question: Initialization and Optimizers

In this last question, you'll implement He Initialization and Different Optimizers. Look at the Jupyter notebook Optimizer_Initialization.ipynb in the hw2 folder to see what you have to do!

For this question, please submit a .zip file your completed work to the Gradescope assignment titled "HW 2 (Code)". No written portion.

5. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

Contributors:

- Anant Sahai.
- Sheng Shen.
- Vasileios Oikonomou.
- Hao Liu.
- Andrew Ng.