Instructor: Vishnu Boddeti
CSE 891-001: Deep Learning
Homework 2
September, 29

Instructions:

- **Filename:** Submit solutions in PDF format titled `written-assignment-2-{msunetid}.pdf`. Submissions in other formats or with other filenames will not be graded. You can produce your file however you like, LaTeX, Word or scan. Handwritten scans that are not legible will not be graded.
- **Submission:** Only homeworks uploaded to Google Classroom will be graded. Make sure to show all the steps of your derivations in order to receive full credit.
- **Integrity and Collaboration:** You are expected to work on the homeworks by yourself. You are not permitted to discuss them with anyone except the instructor. The homework that you hand in should be entirely your own work. You may be asked to demonstrate how you got any results that you report.
- **Clarifications:** If you have any question, please look at Google Classroom first. Other students may have encountered the same problem, and is solved already. If not, post your question there. We will respond as soon as possible.

1 Backpropagation (2pts)

Consider a MLP consisting of N input units, K hidden units and N output units. The activations are computed as follows:

$$\begin{aligned}
\boldsymbol{z} &= \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)} \\
\boldsymbol{h} &= \sigma(\boldsymbol{z}) \\
\boldsymbol{y} &= \boldsymbol{x} + \boldsymbol{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}
\end{aligned}$$

where $\sigma(\cdot)$ is the sigmoid function, applied element wise. The loss $\mathcal{L}$ will involve both $\boldsymbol{h}$ and $\boldsymbol{y}$:

$$\begin{aligned}
\mathcal{L} &= \mathcal{S} + \mathcal{R} \\
\mathcal{S} &= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{s}\|_2^2 \\
\mathcal{R} &= \boldsymbol{r}^T\boldsymbol{h}
\end{aligned}$$

where $\boldsymbol{r}$ and $\boldsymbol{s}$ are given vectors.

1. (**1pt**) Draw the computation graph relating $\boldsymbol{x}$, $\boldsymbol{z}$, $\boldsymbol{h}$, $\boldsymbol{y}$, $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{L}$
2. (**1pt**) Derive the backpropagation equations for computing $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}$. You may use σ' to denote the derivation of the sigmoid function, so you do not have to write it out explicitly.

2 Vector-Jacobian Products (2pts)

Consider the function $f : \mathbb{R}^n \to \mathbb{R}^n$ where $f(\boldsymbol{x}) = \boldsymbol{v}\boldsymbol{v}^T\boldsymbol{x}$, and $\boldsymbol{v} \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$. Here, we will explore the relative costs of evaluating Jacobians and vector-Jacobian products. We denote the Jacobian of f with respect to $\boldsymbol{x}$ as $\boldsymbol{J} \in \mathbb{R}^{n\times n}$.

1. (**0.5 pt**) Compute $\boldsymbol{J}$ for $n = 3$ and $\boldsymbol{v}^T = [1, 2, 3]$ and write down all its values.
2. (**0.5 pt**) What is the time and memory cost of evaluating the Jacobian of the function f as a function of n?
3. (**1.0 pt**) Describe how to evaluate $\boldsymbol{J}^T\boldsymbol{y}$ where $\boldsymbol{y} \in \mathbb{R}^n$ with a time and memory cost that is linear in n. Then, compute $\boldsymbol{z} = \boldsymbol{J}^T\boldsymbol{y}$ where $\boldsymbol{v}^T = [1, 2, 3]$ and $\boldsymbol{y} = [1, 1, 1]$.

3 Linear Regression (3pts)

In this problem we will explore some curious properties of gradient descent and its variants for learning overparameterized models. We will use linear regression for this purpose but some of these properties also hold for deep neural networks.

Given n pairs of input data with d features and scalar label $(\boldsymbol{x}_i, t_i) \in \mathbb{R}^d \times \mathbb{R}$, we wish to find a linear model $f(\boldsymbol{x}) = \hat{\boldsymbol{w}}^T \boldsymbol{x}$ with $\hat{\boldsymbol{w}} \in \mathbb{R}^d$ that minimizes the squared error of prediction on the training samples defined below. For concise notation, denote the data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and the corresponding label vector $\mathbf{t} \in \mathbb{R}^n$. The training objective is to minimize the following loss:

$$\min_{\hat{\boldsymbol{w}}} \frac{1}{n} \sum_{i=1}^{n} (\hat{\boldsymbol{w}}^T \boldsymbol{x}_i - t_i)^2 = \min_{\hat{\boldsymbol{w}}} \frac{1}{n} \|\boldsymbol{X}\hat{\boldsymbol{w}} - \boldsymbol{t}\|_2^2 \tag{1}$$

Assume that $\boldsymbol{X}$ is full rank: $\boldsymbol{X}^T\boldsymbol{X}$ is invertible when $n > d$, and $\boldsymbol{X}\boldsymbol{X}^T$ is invertible otherwise. Note that when $d > n$, the problem is *underdetermined*, i.e., there are less training samples than parameters to be learned. This is analogous to learning an *overparameterized* model, which is common when training of deep neural networks.

1. **Deriving the Gradient (0.5pt)** Write down the gradient of the loss, w.r.t. the learned parameter vector $\hat{\boldsymbol{w}}$.

2. **Underparameterized Model (0.5pt)**

   (a) First consider the underparameterized $d < n$ case. Write down the solution obtained by gradient descent assuming training converges. Show your work. Is the solution unique?

   (b) Assume that the ground truth labels are generated by a linear target: $t_i = \boldsymbol{w}^{*T}\boldsymbol{x}_i$. Show that the solution in part (a) achieves perfect generalization when $d < n$, i.e., $\forall \boldsymbol{x} \in \mathbb{R}^d$, $(\boldsymbol{w}^{*T}\boldsymbol{x} - \hat{\boldsymbol{w}}^T\boldsymbol{x})^2 = 0$.

3. **Overparameterized Model: 2D Example (1pt)**

   (a) Now consider the overparameterized $d > n$ case. We will first illustrate that there exists multiple empirical risk minimizers. For simplicity let $n = 1$ and $d = 2$. Choose $x_1 = [2, 1]$ and $t_1 = 2$, i.e., the one data point and all possible $\hat{\boldsymbol{w}}$ lie on a 2D plane. Show that there exists infinitely many $\hat{\boldsymbol{w}}$ satisfying $\hat{\boldsymbol{w}}^T\boldsymbol{x}_1 = y_1$ on a real line. Write down the equation of the line.

   (b) We know that multiple empirical risk minimizers exist in overparameterized linear regression and there is only one true solution. Thus, it seems unlikely that gradient descent will generalize if it returns an arbitrary solution. However, we will show that gradient descent tends to find a certain solution with desirable properties. This phenomenon, known as *implicit regularization*, helps explain the success of gradient-based methods to train overparameterized models, like deep neural networks. First consider the 2-dimensional example in the previous part: starting from zero initialization i.e., $\hat{\boldsymbol{w}}(0) = \mathbf{0}$, what is the direction of the gradient? You should write down a unit-norm vector. Does the direction change along the trajectory? Based on this geometric intuition, which solution - along the line of solutions - does gradient descent find? Provide a pictorial sketch or a short description of your reasoning.

   (c) Give a geometric argument that among all the solutions on the line, the gradient descent solution from the previous part has the smallest Euclidean norm. *Hint:* Use the Pythogream Theorem.

4. **Overparameterized Model: General Case (1pt)**

   (a) Now we will generalize the previous geometric insight developed to general $d > n$. Show that gradient descent from zero initialization i.e., $\hat{\boldsymbol{w}}(0) = \mathbf{0}$ finds a unique minimizer if it converges. Write down the solution and show your work. *Hint:* Recall the result on the direction of the gradient in the previous 2D example, which suggests that the gradient vector is always spanned by the rows of $\boldsymbol{X}$. What does this tell you about the gradient descent solution?

   (b) Given the gradient descent solution from the previous part $\hat{\boldsymbol{w}}$ and another zero-loss solution $\hat{\boldsymbol{w}}_1$, evaluate $(\hat{\boldsymbol{w}} - \hat{\boldsymbol{w}}_1)^T\hat{\boldsymbol{w}}$. Use this quantity to show that among all the empirical risk minimizers for $d > n$, the gradient descent solution has the smallest Euclidean norm.

5. **Extra Credit (1pt)** Benefit of Overparameterization: Visualize and compare underparameterized with overparameterized polynomial regression (see attached jupyter notebook). Include your code snippets for the `fit_poly` function in the write-up. Does overparameterization (higher degree polynomial) always lead to overfitting, i.e., larger test error?

3 Linear Regression and Optimization (3pts)

In this question we will study the solutions to linear regression from the variants of gradient descent that we saw in class.

1. **Stochastic Gradient Descent (1pt)** Recall that SGD estimates the gradient from a single randomly sampled training example and updates the parameters until convergence. In the overparameterized $d > n$ case, assume that SGD also converges to a solution $\hat{\boldsymbol{w}}$ such that $\boldsymbol{X}\hat{\boldsymbol{w}} = \boldsymbol{t}$. Show that the SGD solution is identical to the minimum norm solution $\boldsymbol{w}^*$ obtained by gradient descent, i.e., $\hat{\boldsymbol{w}} = \boldsymbol{w}^*$. *Hint:* Is $\boldsymbol{x}_i$ contained in the span of $\boldsymbol{X}$? Do the update steps SGD ever leave the span of $\boldsymbol{X}$?

2. **Mini-Batch SGD (1pt)** Recall that SGD estimates the gradient from mini-batches of data $\mathbf{B} \in \mathbb{R}^{b\times d}$, where $1 < b << n$ and $\boldsymbol{B}$ is taken from the rows of $\boldsymbol{X}$. In this case, does mini-batch SGD obtain the minimum norm solution on convergence?

3. **Adaptive Methods (1pt)** We will now consider the behavior of adaptive gradient descent methods, in particular the AdaGrad[a] method. Let w_i denote the $i-$th parameter. A scalar learning rate η is used. At time t for parameter i, the update step for AdaGrad is as follows:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,i}} + \epsilon} \nabla_{w_{t,i}} \mathcal{L}(\boldsymbol{w})$$
$$G_{t,i} = G_{t-1,i} + (\nabla_{w_{t,i}} \mathcal{L}(\boldsymbol{w}))^2$$

   The term ϵ is a fixed small scalar used for numerical stability. Intuitively, Adagrad can be thought of as adapting the learning rate in each dimension to efficiently move through badly formed curvatures.

   Consider the overparameterized linear regression model $(d > n)$. Assume that AdaGrad converges to a solution. Provide a proof or a counterexample for whether AdaGrad always obtains the minimum norm solution. *Hint:* Compute the 2D case i.e., $\boldsymbol{x}_1 = [2, 1]$, $\boldsymbol{w}_0 = [0, 0]$, and $t = [2]$.

   Does this result hold true for other adaptive methods (RMSProp, Adam) in general?

[a] https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf