CS 189 Introduction to Machine Learning
Fall 2018 HW7

1 Getting Started

Read through this page carefully. You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW7 Write-Up". If there are graphs, include those graphs in the correct sections. Do not simply reference your appendix.

(a) Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadverdently cheats.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

This homework is due **Monday, November 12 at 10pm.**

2 Linear Neural Networks

In this problem, we will consider neural networks with the the identity function as the activation. These are known as *linear neural networks*. Formally, consider a set of input data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, and outputs $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^p$. For a depth $L \geq 1$, and weight dimensions $d_0, \ldots, d_L$, where $d_0 = d$ and $d_L = p$, an L-layer neural network with weights $\mathbf{W}_{1:L} := (\mathbf{W}_1, \ldots, \mathbf{W}_L)$, where $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$, is defined as the predictor function

$$\mathbf{f}_{\mathbf{W}_{1:L}}(\mathbf{x}) := (\mathbf{W}_L \cdot \mathbf{W}_{L-1} \cdot \ldots \mathbf{W}_1)\mathbf{x}.$$

Note then that $\mathbf{f}_{\mathbf{W}_{1:L}}(\mathbf{x}) = \overline{\mathbf{W}}\mathbf{x}$, where $\overline{\mathbf{W}} := \mathbf{W}_L \cdot \mathbf{W}_{L-1} \cdot \ldots \mathbf{W}_1 \in \mathbb{R}^{p \times d}$. Thus, $\mathbf{f}_{\mathbf{W}_{1:L}}(\mathbf{x})$ is a *linear predictor*.

In what follows, we will examine the optimization landscape to understand the relationship between critical points and global minima.

(a) We begin by understanding the *capacity* of the linear network. Fix a matrix $\mathbf{W}_0 \in \mathbb{R}^{p \times d}$. **Show that there exists weights** $(\mathbf{W}_1, \mathbf{W}_2 \ldots \mathbf{W}_L)$ of associated dimensions $d_1, \ldots, d_L$ **such that** $\overline{\mathbf{W}} = \mathbf{W}_0$ **if and only if** $\operatorname{rank}(\mathbf{W}_0) \leq \min_{i=0}^{L} d_i$**.**

Hint: For the "if" direction, start with the case where

$$\mathbf{W}_0 = \begin{bmatrix} I_{d_*} & \mathbf{0}_{\mathbf{d}_* \times (\mathbf{d} - \mathbf{d}_*)} \\ \mathbf{0}_{(p-d_*) \times d_*} & \mathbf{0}_{(p-d_*) \times (d_* - d)}, \end{bmatrix}$$

(b) Next, we consider the convexity of the training loss. For a function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^p$, and data $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, where x_i in $\mathbb{R}^d$ and y_i in $\mathbb{R}^p$, we introduce the training loss

$$\widehat{\mathcal{R}}(\mathbf{f}) := \frac{1}{2} \sum_{i=1}^{n} \|\mathbf{f}(\mathbf{x}_i) - \mathbf{y}_i\|_2^2.$$

Show that for $\mathbf{f}_{\mathbf{W}_{1:L}}$ **defined above**,

$$\widehat{\mathcal{R}}(f_{\mathbf{W}_{1:L}}) = Q(\overline{\mathbf{W}}) := \frac{1}{2} \|\overline{\mathbf{W}}\mathbf{X}^\top - \mathbf{Y}^\top\|_F^2$$

where we define the data matrices $\mathbf{X}$ and $\mathbf{Y}$ to have rows given by $\mathbf{x}_i$ and $\mathbf{y}_i$ respectively.

Then, argue that $Q(\mathbf{W})$ **is a convex function**.

Hint: you may use the following fact for convexity. For a matrix $\mathbf{W} \in \mathbb{R}^{p \times d}$*, let* $\mathbf{W}[j] \in \mathbb{R}^d$ *denote the* j*-th row of* $\mathbf{W}$*. Then if* $F(\mathbf{W}) = \sum_{j=1}^{p} F_j(\mathbf{W}[j])$*, and each* $F_j(\cdot)$ *is convex, then* F *is convex.*

(c) Now, we will consider derivatives of the training loss. First, we will compute the derivative of the loss with respect to the *matrix product* $\overline{\mathbf{W}}$. To do so, we may want to make use of our knowledge of backpropagation. Consider the relation $\mathbf{F} = \mathbf{AB}$ and let $\mathbf{Z}$ denote the message backpropagated to the $\mathbf{F}$ node. The recall from lecture that $\mathbf{Z}_1 = \mathbf{ZB}^\top$ and $\mathbf{Z}_2 = \mathbf{A}^\top\mathbf{Z}$ are the messages that will be passed back from the $\mathbf{A}$ and $\mathbf{B}$ nodes respectively.

Derive the expression for $\frac{\partial Q}{\partial \overline{\mathbf{W}}}$.

Hint: write $Q(\overline{\mathbf{W}})$ *as a computation graph:* $\mathbf{E} = \overline{\mathbf{W}}\mathbf{X}^\top - \mathbf{Y}^\top$, $Q = \frac{1}{2}\sum_{i,j}\mathbf{E}_{ij}^2$

(d) Now, we can reason about the global minimizers of the training loss. **Prove that if** $\min_{i=0}^{L} d_i = \min\{p, d\}$**, then** $\mathbf{W}_1, \ldots, \mathbf{W}_L$ **is a global minimizer of** $(\mathbf{W}_1, \ldots, \mathbf{W}_L) \mapsto \widehat{\mathcal{R}}(f_{\mathbf{W}_{1:L}})$ **if and only if** $\overline{\mathbf{W}}\mathbf{X}^\top\mathbf{X} - \mathbf{Y}^\top\mathbf{X} = 0$**.**

Hint: for this problem, you will need to show an equivalence (i.e. "if and only if") between the global minimizer of $Q(\cdot)$ *and the global minimizers of the training loss* $\widehat{\mathcal{R}}(f_{\mathbf{W}_{1:L}})$.

(e) Now we consider each layer's weight matrix independently. **Write down** $\frac{\partial \widehat{\mathcal{R}}(f_{\mathbf{W}_{1:L}})}{\partial \mathbf{W}_i}$. You may want to write $\widehat{\mathcal{R}}(f_{\mathbf{W}_{1:L}})$ as an extension of the computation graph in part (c). For this, you can define $\overline{\mathbf{W}}_{(i-1:1)} := \mathbf{W}_{i-1} \cdot \cdots \cdot \mathbf{W}_1$ and $\overline{\mathbf{W}}_{(L:i+1)} := \mathbf{W}_L \cdot \cdots \cdot \mathbf{W}_{i+1}$ (or identity if $i = 1, L$). Then $\mathbf{P} = \mathbf{W}_i\overline{\mathbf{W}}_{(i-1:1)}$ and $\overline{\mathbf{W}} = \overline{\mathbf{W}}_{(L:i+1)}\mathbf{P}$.

Use this answer to explain why a critical point of $\widehat{\mathcal{R}}$ **is not necessarily a global minimum**.

(f) Suppose that $p = d$ and $d = d_0 = d_1 = \cdots = d_L$, that is, all the the weight matrices are square. Moreover, suppose that $\mathbf{W}_{1:L}^* = (\mathbf{W}_1^*, \ldots, \mathbf{W}_L^*)$ is a critical point of the objective $\mathbf{W}_{1:L} \mapsto \widehat{\mathcal{R}}(f_{\mathbf{W}_{1:L}})$. **Give a sufficient condition on** $\mathbf{W}_1^*, \ldots, \mathbf{W}_L^*$ **which ensures that** $\mathbf{W}_{1:L}^*$ **is in fact a global minimum.**

3 Promises and Pitfalls of Neural Networks

In this problem, we will consider the ability of neural networks to represent and learn functions. Consider the following setting: features $\mathbf{x}$ are be drawn uniformly at random from $\mathcal{X} = \{0, 1\}^d$. The labels depend on the parameter $\mathbf{v}_*$, drawn uniformly at random (and independently from $\mathbf{x}$) from $\mathcal{V} = \{0, 1\}^d$. Then the labels are given as $y = (-1)^{\mathbf{v}_*^\top \mathbf{x}}$.

In this setting, we can consider the parameter $\mathbf{v}_*$ as a mask, so $\mathbf{v}_*^\top\mathbf{x}$ counts the nonzero elements of $\mathbf{x}$ at the positions determined by the mask $\mathbf{v}_*$. Then the output y classifies whether this quantity is even or odd.

In this problem we will show that neural networks of modest size can approximate such a function exactly. This is a positive result on the *capacity* of neural networks. However, we will then explore the failure of a gradient-based training scheme to arrive at these correct parameters, showing a negative result on the ability to *learn*.

We will consider the two layer fully connected neural network with k ReLu activation units in the

hidden layer and a linear activation (with a bias term) on the single output node:

$$f_{\mathbf{w},\mathbf{b},\boldsymbol{\alpha},\beta}(\mathbf{x}) = \sum_{i=1}^{k} \alpha_i \max\{0, \mathbf{w}_i^\top \mathbf{x} + b_i\} + \beta \,.$$

For brevity, we will define a vector of parameters as

$$\mathbf{p} = \begin{bmatrix} \mathbf{w}_1^\top & \dots & \mathbf{w}_k^\top & b_1 & \dots & b_k & \alpha_1 & \dots & \alpha_k & \beta \end{bmatrix} \in \mathbb{R}^{dk+2k+1} \,.$$

In this problem, we will make use of pytorch, a python library which supports tensor computation and auto-differentiation. You should install pytorch and take a look at the basic tutorial here: `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`.

(a) We claim that if $k \geq \frac{3d}{2}$, then the network can exactly represent the function $y = (-1)^{\mathbf{v}_*^\top \mathbf{x}}$ for a fixed $\mathbf{v}_*$. Specifically, we claim that the following parameter settings achieve this:

- $\mathbf{w}_i = \mathbf{v}_*$ for $1 \leq i \leq \frac{3d}{2}$, and zero otherwise,
- for $0 \leq i \leq \frac{d}{2}$, set
  - $b_{3i+1} = -(2i - \frac{1}{2})$, $b_{3i+2} = -2i$, $b_{3i+3} = -(2i + \frac{1}{2})$,
  - $\alpha_{3i+1} = 4$, $\alpha_{3i+2} = -8$, $\alpha_{3i+3} = 4$,
- $\beta = -1$.

Your task is to verify this statement numerically. To do so, first **implement the network structure using the starter code in `pitfalls.py` with** $k = 10d$. Please read the starter code comments and examples carefully. Then, **manually set the weights of the network as described above** (make note of the fact that python indexing starts at 0 rather than 1!). Running the script will verify that the network does represent the function. Please include screenshots of your code.

(b) Instead of specifying weights by hand, we prefer to train a deep classifier based on data $\{\mathbf{x}_i, y_i\}_{i=1}^n$. To do so, we will run gradient descent on the hinge loss:

$$\ell(f_{\mathbf{p}}(\mathbf{x}), y) = \max\{0, 1 - f_{\mathbf{p}}(\mathbf{x}) \cdot y\} \,.$$

Implement the batch loss in `myLoss()`. Be sure that you use only functions supplied in the `torch` library so that the auto-differentiation functionality will work. Then use the provided code to train the network for 5×10^4 iterations with input dimensions $d = 5, 10, 30$. **Plot the loss over iterations, and comment on what you see**. Please include plots and screenshots of your code.

(c) Despite the capacity result in (a), it appears that in some cases, our neural network is not learning. We will now show that this failure comes from a lack of informative gradients, i.e. gradients do not give useful information about the key parameter $\mathbf{v}_*$. A network is trained with gradient descent updates on the *risk*, which for this problem is given by[1]

$$R(\mathbf{p}) = \mathbb{E}_{\mathbf{x},y}\left[\ell(f_{\mathbf{p}}(\mathbf{x}), y)\right] = \mathbb{E}_{\mathbf{x}}\left[\max\{0, 1 - f_{\mathbf{p}}(\mathbf{x}) \cdot (-1)^{\mathbf{v}_*^\top \mathbf{x}}\}\right] \,.$$

[1] In practice, networks are trained on the *empirical risk*, which is $\frac{1}{n}\sum_{i=1}^n \ell(f_{\mathbf{p}}(\mathbf{x}_i), y_i)$. In this problem, we analyze the full *population risk* to show that even in the best case, the gradients are uninformative.

To show that gradients are uninformative, we will show that there is little variation in their value for different values of $\mathbf{v}_*$. Specifically, we will show that

$$\mathbb{E}_{\mathbf{v}_*}\left[\|\nabla R(\mathbf{p}) - \mathbf{a}\|_2^2\right] \leq \frac{C_{\mathbf{p}}}{2^d}\,,$$

where $\mathbf{a}$ is a quantity independent of $\mathbf{v}_*$ and $C_{\mathbf{p}}$ is a constant depending on network parameters. This statement means the amount of information that the gradient contains about the underlying parameter $\mathbf{v}_*$ exponentially decreases in d.

(i) The first step is to compute the gradient of the network with respect to its parameters. **Compute** $\nabla_{\mathbf{p}} f_{\mathbf{p}}(\mathbf{x})$, which is defined as

$$\nabla_{\mathbf{p}} f_{\mathbf{p}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_{\mathbf{p}}(\mathbf{x})}{\partial \mathbf{w}_1} & \dots & \frac{\partial f_{\mathbf{p}}(\mathbf{x})}{\partial b_1} & \dots & \frac{\partial f_{\mathbf{p}}(\mathbf{x})}{\partial \alpha_1} & \dots & \frac{\partial f_{\mathbf{p}}(\mathbf{x})}{\partial \beta} \end{bmatrix}^\top .$$

Then, **find a constant $c_{\mathbf{p}}$ depending only on** $\mathbf{w}_i, b_i, \alpha_i, \beta$ such that every element of the gradient is bounded, $|\nabla_{\mathbf{p}} f_{\mathbf{p}}(\mathbf{x})_j| \leq c_{\mathbf{p}}$ for all j.

(ii) The next step is to write an expression for $\nabla R(\mathbf{p})$ that is amenable to analysis. **Show that** $\nabla R(\mathbf{p}) = \mathbf{a} + \mathbb{E}_{\mathbf{x}}\left[(-1)^{\mathbf{v}_*^\top \mathbf{x}} \mathbf{g}(\mathbf{x})\right]$ for some vector valued function $\mathbf{g}(\mathbf{x})$ and $\mathbf{a} = -\mathbb{E}_{\mathbf{x}}\left[\frac{\mathbf{1}_1(f_{\mathbf{p}}(\mathbf{x})) - \mathbf{1}_1(-f_{\mathbf{p}}(\mathbf{x}))}{2} \nabla_{\mathbf{p}} f_{\mathbf{p}}(\mathbf{x})\right]$, where we define an indicator function

$$\mathbf{1}_1(u) = \begin{cases} 0 & u \geq 1 \\ 1 & u \leq 1 \end{cases}.$$

Hint: It may be useful to note that $y \cdot \mathbf{1}_1(f_{\mathbf{p}}(\mathbf{x}) \cdot y) = \begin{cases} \mathbf{1}_1(f_{\mathbf{p}}(\mathbf{x})) & y = 1 \\ -\mathbf{1}_1(-f_{\mathbf{p}}(\mathbf{x})) & y = -1 \end{cases}$

(iii) Next, we verify an important property that will help in later simplifications. A collection of functions $f_i : \mathcal{X} \to \mathbb{R}$ for $i = 1, \dots, m$ is said to be *sub-orthonormal* if

$$\mathbb{E}_{\mathbf{x}}\left[f_i(\mathbf{x}) f_\ell(\mathbf{x})\right] = 0 \text{ for } i \neq \ell \text{ and } \mathbb{E}_{\mathbf{x}}[f_i(\mathbf{x})^2] \leq 1\,.$$

Show that for distinct $v_1, ..., v_{|\mathcal{V}|}$**, the functions given by** $(-1)^{\mathbf{v}_i^\top \mathbf{x}}$ for $i = 1, \dots, |\mathcal{V}|$ are sub-orthonormal.

(iv) **Finally, show that** $\mathbb{E}_{\mathbf{v}_*}\left[\|\nabla R(\mathbf{p}) - \mathbf{a}\|_2^2\right] \leq \frac{C_{\mathbf{p}}}{2^d}$, where $C_{\mathbf{p}}$ is a constant you will define. You may use the following property without proof:[2] If functions $f_i : \mathcal{X} \to \mathbb{R}$ for $i = 1, \dots, m$ are *sub-orthonormal*, then for any scalar valued function $h : \mathcal{X} \to \mathbb{R}$,

$$\sum_{i=1}^{m} (\mathbb{E}_{\mathbf{x}}\left[f_i(\mathbf{x}) h(\mathbf{x})\right])^2 \leq \mathbb{E}_{\mathbf{x}}[h(\mathbf{x})^2]\,.$$

[2] You are equipped with the right skill set to prove this property. If you are interested, try to prove this simpler statement first: For a collection of vectors $\{\mathbf{z}_i\}_{i=1}^n$ for which $\mathbf{z}_i \perp \mathbf{z}_j$ and $\|\mathbf{z}_i\|_2 \leq 1$,

$$\sum_{i=1}^{n} (\mathbf{z}_i^\top \mathbf{x})^2 \leq \|\mathbf{x}\|_2^2\,.$$

Then think about the connection between this statement and the problem.

Hint: Expand $\mathbb{E}_{\mathbf{v}_*}\left[\|\nabla R(\mathbf{p}) - \mathbf{a}\|_2^2\right]$ *as a double summation, over elements* $\mathbf{v}_i \in \mathcal{V}$ *and elements of vector valued function* $\mathbf{g}_j(\mathbf{x})$.

(d) Your ask your friend, who did not take 189, about the training problems that you observed in part (b). Uninterested in your thoughtful analysis in part (c), your friend suggests that you make the network deeper, and switch to sigmoid activation functions. **Why might this be a bad idea, even for small** d**?**

4 Image Classification

In this problem, we will consider the task of image classification. We will use the CIFAR-10 dataset, which contains sixty thousand 32×32 color images in ten different classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into $50,000$ training images and $10,000$ validation images.

First, we approach this problem using linear methods studied earlier in the semester. We will visualize the data using principle component analysis (PCA) and canonical correlation analysis (CCA), and then we will try a simple linear regression approach to classification. Finally, we will use a state-of-the-art convolution neural network (CNN) model. There are many images in the CIFAR-10 dataset, and the CNN model we use will be fairly computationally intensive. For this reason, make sure to give yourself plenty of time to complete this problem.

(a) How hard is this image classification problem? We will investigate by visualizing the dataset in two dimensions. Dataset loading code is provided for you in `cifar10.py`, along with methods to compute the PCA and CCA embeddings of the dataset. **Fill in code to visualize the two dimensional PCA and CCA embeddings.** Make sure to use different colors for each class, and label your plots with titles and legends. **Include your plots, and explain the difference between these two visualization methods.**

(b) Overlapping clusters in a two dimensional visualization does not necessarily mean overlap in a higher dimensional space. As a baseline, we will try linear regression. **Set `LINEAR_FLAG=True` and report the training and validation accuracy**.

(c) Now, we will use state-of-the-art neural net architectures. Specifically, we will use a ResNet18 model: a residual CNN with 18 layers. Inspect the code in the class `ResNet18` to understand the architecture. You may also want to run the script with `PRINT_NN_FLAG=True`. **Include a diagram summarizing the network architecture.** Your diagram should indicate the number of dimensions for each layer.

(d) For state-of-the-art performance, a common strategy is data augmentation, in which random crops and shifts are added to the dataset. **Modify `transform_train` to include a random crop of size** 32 **and a random horizontal flip**. You should use methods in `torchvision.transform`. Include a screenshot of your code.

(e) We are almost ready to train the network. In this step, we will compare three different learning rate schemes: constant, annealed, and a specially defined scheme. The constant learning rate

should always return a constant value. The annealed learning rate should return the previous learning rate multiplied by a decay factor. The special scheme is defined for you and does not need to be modified. **Implement the constant and annealed learning rate functions**. Include a screenshot of you code in your writeup.

Now, train the network with each of these learning rate schemes for one epoch, using the existing code and setting `TRAIN_NN_FLAG=True`. You may want to use the subset flags to test your code (e.g. `python cifar10_sol.py --subset-train 128 --subset-val 512`), since it will take a nontrivial amount of time to run on a CPU. **Report your results here**.

(f) (**Not required**) Continue running with the best learning rate scheme for several more epochs to increase the accuracy. By running for several epochs, it is possible to get to over 90% accuracy. This will take several hours on a CPU. (If you have access to a GPU, you can speed up the training significantly.)