Homework 2

Version: 1.1
Version Release Date: 2022-02-06
Changes by Version:

- **(v1.1)**
  (Q1.2) Added initial condition clarification.
  (Q2.1.2) Reweighted to be 0.5 points
  (Q2.3.1) Reweighted to be 0.5 points
  (Q2.3) Change the weight decay equation to use X instead of $\mathbf{x}_i$

Deadline: Friday, Feb.11, at 11:59pm.

Submission: You must submit your solutions as a PDF file through MarkUs[1]. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.

See the syllabus on the course website[2] for detailed policies. You may ask questions about the assignment on Piazza[3]. *Note that 10% of the homework mark (worth 1 pt) may be removed for a lack of neatness.*

You may notice that some questions are worth 0 pt, which means we will not mark them in this Homework. Feel free to skip them if you are busy. However, you are expected to see some of them in the midterm. So, we won't release the solution for those questions.

The teaching assistants for this assignment are Ian Shi and Phil Fradkin. Send your email with subject "*[CSC413] HW2 ...*" to `csc413-2022-01-tas@cs.toronto.edu` or post on Piazza with the tag `hw2`.

1 Optimization

This week, we will continue investigating the properties of optimization algorithms, focusing on stochastic gradient descent and adaptive gradient descent methods. For a refresher on optimization, refer to: `https://csc413-uoft.github.io/2021/assets/slides/lec03.pdf`.

We will continue using the linear regression model established in Homework 1. Given n pairs of input data with d features and scalar labels $(\mathbf{x}_i, t_i) \in \mathbb{R}^d \times \mathbb{R}$, we want to find a linear model $f(\mathbf{x}) = \hat{\mathbf{w}}^T\mathbf{x}$ with $\hat{\mathbf{w}} \in \mathbb{R}^d$ such that the squared error on training data is minimized. Given a data matrix $X \in \mathbb{R}^{n\times d}$ and corresponding labels $\mathbf{t} \in \mathbb{R}^n$, the objective function is defined as:

$$\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 \tag{1}$$

1.1 Mini-Batch Stochastic Gradient Descent (SGD)

Mini-batch SGD performs optimization by taking the average gradient over a mini-batch, denoted $\mathcal{B} \in \mathbb{R}^{b\times d}$, where $1 < b \ll n$. Each training example in the mini-batch, denoted $\mathbf{x}_j \in \mathcal{B}$, is

[1] `https://markus.teach.cs.toronto.edu/2022-01/courses/16`
[2] `https://uoft-csc413.github.io/2022/assets/misc/syllabus.pdf`
[3] `https://piazza.com/utoronto.ca/winter2022/csc4132516/`

randomly sampled without replacement from the data matrix X. Assume that X is full rank. Where $\mathcal{L}$ denotes the loss on $\mathbf{x}_j$, the update for a single step of mini-batch SGD at time t with scalar learning rate η is:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta}{b} \sum_{\mathbf{x}_j \in \mathcal{B}} \nabla_{\mathbf{w}_t} \mathcal{L}(\mathbf{x}_j, \mathbf{w}_t) \tag{2}$$

Mini-batch SGD iterates by randomly drawing mini-batches and updating model weights using the above equation until convergence is reached.

1.1.1 Minimum Norm Solution [2pt]

Recall Question 3.3 from Homework 1. For an overparameterized linear model, gradient descent starting from zero initialization finds the unique minimum norm solution $\mathbf{w}^*$ such that $X\mathbf{w}^* = \mathbf{t}$. Let $\mathbf{w}_0 = \mathbf{0}$, $d > n$. Assume mini-batch SGD also converges to a solution $\hat{\mathbf{w}}$ such that $X\hat{\mathbf{w}} = \mathbf{t}$. Show that mini-batch SGD solution is identical to the minimum norm solution $\mathbf{w}^*$ obtained by gradient descent, i.e., $\hat{\mathbf{w}} = \mathbf{w}^*$.

Hint: Is $\mathbf{x}_j$ or $\mathcal{B}$ contained in span of X? Do the update steps of mini-batch SGD ever leave the span of X?

1.2 Adaptive Methods

We now consider the behavior of adaptive gradient descent methods. In particular, we will investigate the RMSProp 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 RMSProp is shown by:

$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{v_{i,t}} + \epsilon} \nabla_{w_{i,t}} \mathcal{L}(w_{i,t}) \tag{3}$$

$$v_{i,t} = \beta(v_{i,t-1}) + (1-\beta)(\nabla_{w_{i,t}} \mathcal{L}(w_{i,t}))^2 \tag{4}$$

We begin the iteration at $t = 0$, and set $v_{i,-1} = 0$. The term ϵ is a fixed small scalar used for numerical stability. The momentum parameter β is typically set such that $\beta \geq 0.9$ Intuitively, RMSProp adapts a separate learning rate in each dimension to efficiently move through badly formed curvatures (see lecture slides/notes).

1.2.1 Minimum Norm Solution [1pt]

Consider the overparameterized linear model ($d > n$) for the loss function defined in Section 1. Assume the RMSProp optimizer converges to a solution. Provide a proof or counterexample for whether RMSProp always obtains the minimum norm solution.

Hint: Compute a simple 2D case. Let $\mathbf{x}_1 = [2, 1]$, $w_0 = [0, 0]$, $t = [2]$.

1.2.2 [0pt]

Consider the result from the previous section. Does this result hold true for other adaptive methods (Adagrad, Adam) in general? Why might making learning rates independent per dimension be desirable?

2 Gradient-based Hyper-parameter Optimization

In this problem, we will implement a simple toy example of *gradient-based hyper-parameter optimization*, introduced in Lecture 3 (slides 14).

Often in practice, hyper-parameters are chosen by trial-and-error based on a model evaluation criterion. Instead, *gradient-based hyper-parameter optimization* computes gradient of the evaluation criterion w.r.t. the hyper-parameters and uses this gradient to directly optimize for the best set of hyper-parameters. For this problem, we will optimize for the learning rate of gradient descent in a regularized linear regression problem.

Specifically, given n pairs of input data with d features and scalar label $(\mathbf{x}_i, t_i) \in \mathbb{R}^d \times \mathbb{R}$, we wish to find a linear model $f(\mathbf{x}) = \hat{\mathbf{w}}^\top \mathbf{x}$ with $\hat{\mathbf{w}} \in \mathbb{R}^d$ and a L2 penalty, $\tilde{\lambda}\|\hat{\mathbf{w}}_2^2\|$, that minimizes the squared error of prediction on the training samples. $\tilde{\lambda}$ is a hyperparameter that modulates the impact of the L2 regularization on the loss function. Using the concise notation for the data matrix $X \in \mathbb{R}^{n\times d}$ and the corresponding label vector $\mathbf{t} \in \mathbb{R}^n$, the squared error loss can be written as:

$$\tilde{\mathcal{L}} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 + \tilde{\lambda}\|\hat{\mathbf{w}}\|_2^2.$$

Starting with an initial weight parameters $\mathbf{w}_0$, gradient descent (GD) updates $\mathbf{w}_0$ with a learning rate η for t number of iterations. Let's denote the weights after t iterations of GD as $\mathbf{w}_t$, the loss as $\mathcal{L}_t$, and its gradient as $\nabla_{\mathbf{w}_t}$. The goal is the find the optimal learning rate by following the gradient of $\mathcal{L}_t$ w.r.t. the learning rate η.

2.1 Computation Graph

2.1.1 [0.5pt]

Consider a case of 2 GD iterations. Draw the computation graph to obtain the final loss $\tilde{\mathcal{L}}_2$ in terms of $\mathbf{w}_0, \nabla_{\mathbf{w}_0}\tilde{\mathcal{L}}_0, \tilde{\mathcal{L}}_0, \mathbf{w}_1, \tilde{\mathcal{L}}_1, \nabla_{\mathbf{w}_1}\tilde{\mathcal{L}}_1, \mathbf{w}_2, \tilde{\lambda}$ and η.

2.1.2 [0.5pt]

Then, consider a case of t iterations of GD. What is the memory complexity for the forward-propagation in terms of t? What is the memory complexity for using the standard back-propagation to compute the gradient w.r.t. the learning rate, $\nabla_\eta \tilde{\mathcal{L}}_t$ in terms of t?
Hint: Express your answer in the form of $\mathcal{O}$ in terms of t.

2.1.3 [0pt]

Explain one potential problem for applying gradient-based hyper-parameter optimization in more realistic examples where models often take many iterations to converge.

2.2 Optimal Learning Rates

In this section, we will take a closer look at the gradient w.r.t. the learning rate. To simplify the computation for this section, consider an unregularized loss function of the form $\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2$. Let's start with the case with only one GD iteration, where GD updates the model weights from $\mathbf{w}_0$ to $\mathbf{w}_1$.

2.2.1 [1pt]

Write down the expression of $\mathbf{w}_1$ in terms of $\mathbf{w}_0$, η, $\mathbf{t}$ and X. Then use the expression to derive the loss $\mathcal{L}_1$ in terms of η.
Hint: If the expression gets too messy, introduce a constant vector $\mathbf{a} = X\mathbf{w}_0 - \mathbf{t}$

2.2.2 [0pt]

Determine if $\mathcal{L}_1$ is convex w.r.t. the learning rate η.
Hint: A function is *convex* if its second order derivative is positive

2.2.3 [1pt]

Write down the derivative of $\mathcal{L}_1$ w.r.t. η and use it to find the optimal learning rate η^* that minimizes the loss after one GD iteration. Show your work.

2.3 Weight decay and L2 regularization

Although well studied in statistics, L2 regularization is usually replaced with explicit weight decay in modern neural network architectures:

$$\mathbf{w}_{i+1} = (1-\lambda)\mathbf{w}_i - \eta\nabla\mathcal{L}_i(X) \tag{5}$$

In this question you will compare regularized regression of the form $\tilde{\mathcal{L}} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 + \tilde{\lambda}\|\hat{\mathbf{w}}\|_2^2$ with unregularized loss, $\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2$, accompanied by weight decay (equation 5).

2.3.1 [0.5pt]

Write down two expressions for $\mathbf{w}_1$ in terms of $\mathbf{w}_0$, η, $\mathbf{t}$, λ, $\tilde{\lambda}$, and X. The first one using $\tilde{\mathcal{L}}$, the second with $\mathcal{L}$ and weight decay.

2.3.2 [0.5pt]

How can you express $\tilde{\lambda}$ (corresponding to L2 loss) so that it is equivalent to λ (corresponding to weight decay)?
Hint: Think about how you can express $\tilde{\lambda}$ in terms of λ and another hyperparameter.

2.3.3 [0pt]

Adaptive gradient update methods like RMSprop (equation 4) modulate the learning rate for each weight individually. Can you describe how L2 regularization is different from weight decay when adaptive gradient methods are used? In practice it has been shown that for adaptive gradients methods weight decay is more successful than l2 regularization.

3 Convolutional Neural Networks

The last set of questions aims to build basic familiarity with convolutional neural networks (CNNs).

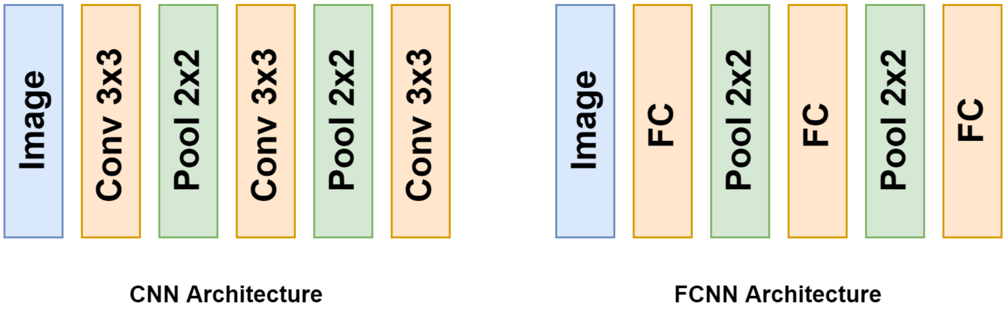
3.1 Convolutional Filters [0.5pt]

Given the input matrix $\mathbf{I}$ and filter $\mathbf{J}$ shown below, compute $\mathbf{I} * \mathbf{J}$, the output of the convolution operation (as defined in lecture 4). Assume zero padding is used such that the input and output are of the same dimension. What feature does this convolutional filter detect?

$$\mathbf{I} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \mathbf{J} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \mathbf{I} * \mathbf{J} = \begin{bmatrix} ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \end{bmatrix}$$

3.2 Size of Conv Nets [1pt]

CNNs provides several advantages over fully connected neural networks (FCNNs) when applied to image data. In particular, FCNNs do not scale well to high dimensional image data, which you will demonstrate below. Consider the following CNN architecture on the left:



The input image has dimension 32×32 and is grey-scale (one channel). For ease of computation, assume all convolutional layers only have 1 output channel, and use 3×3 kernels. Assume zero padding is used in convolutional layers such that the output dimension is equal to the input dimension. Each max pooling layer has a filter size of 2×2 and a stride of 2.

We consider an alternative architecture, shown on the right, which replaces convolutional layers with fully connected (FC) layers in an otherwise identical architecture. For both the CNN architecture and the FCNN architecture, compute the total number of neurons in the network, and the total number of trainable parameters. You should report four numbers in total. Finally, name one disadvantage of having more trainable parameters.

3.3 Receptive Fields [0.5pt]

The receptive field of a neuron in a CNN is the area of the image input that can affect the neuron (i.e. the area a neuron can 'see'). For example, in the first layer of the previous architecture, one neuron is computed using an input of the size 3×3 (the filter size), so its receptive field is of size 3×3. However, as we go deeper into the CNN, the receptive field increases. The receptive field of a neuron after the first max pooling is 4×4, and the receptive field of a neuron after the second convolutional layer is 8×8. Some helpful resources to visualize receptive fields can be found at: https://distill.pub/2019/computing-receptive-fields/ and https://github.com/vdumoulin/conv_arithmetic[4]

[4] See: "A guide to convolution arithmetic for deep learning", https://arxiv.org/abs/1603.07285

In the previous architecture, suppose we replaced the 3×3 sized filter with a 5×5 sized filter in the convolution layers, while keeping max-pooling, stride, padding, etc. constant. What is the receptive field of a neuron after the second convolutional layer? List 2 other things that can affect the size of the receptive field of a neuron.