

Homework 2 - Version 1.0

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

Submission: You must submit your solutions as a PDF file through MarkUs¹. 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 website² for detailed policies. You may ask questions about the assignment on Piazza³. *Note that 10% of the homework mark (worth 1 pt) may be removed for a lack of neatness.*

The teaching assistants for this assignment are Andrew Jung and Erfan Hosseini.

<mailto:csc413-2021-01-tas@cs.toronto.edu>

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, please 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 wish to find a linear model $f(x) = \hat{\mathbf{w}}^T \mathbf{x}$ 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:

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

1.1 Stochastic Gradient Descent (SGD)

SGD performs optimization by taking a stochastic estimate of the gradient from a single training example. This process is iterated until convergence is reached. Let $\mathbf{x}_i \in \mathbb{R}^d$, $1 \leq i \leq n$ be a single training datum taken from the data matrix X . Assume that X is full rank. Where \mathcal{L}_i denotes the loss with respect to \mathbf{x}_i , the update for a single step of SGD at time t with scalar learning rate η is:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t} \mathcal{L}_i(\mathbf{x}_i, \mathbf{w}_t) \quad (2)$$

SGD iterates by randomly drawing training samples and updating model weights using the above equation until convergence is reached.

1.1.1 Minimum Norm Solution [1pt]

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 SGD also converges to a solution $\hat{\mathbf{w}}$ such that $X\hat{\mathbf{w}} = \mathbf{t}$. Show that 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}_i contained in span of X ? Do the update steps of SGD ever leave the span of X ?

¹<https://markus.teach.cs.toronto.edu/csc413-2021-01>

²<https://csc413-uoft.github.io/2021/assets/misc/syllabus.pdf>

³<https://piazza.com/class/kjt32fc0f7y3kb>

Answer

Consider that \mathbf{x}_i is sampled as a row of X . By definition, \mathbf{x}_i is within the row space of X . The gradient calculation for squared loss is:

$$\nabla \mathcal{L}_i = \mathbf{x}_i^T \mathbf{x}_i \mathbf{w} - \mathbf{x}_i^T \mathbf{t}$$

Starting from zero weight initialization (contained in span of X), the gradient always lies within the span of X . Therefore, the update steps of SGD are linear combinations of vectors within X , and will never leave span of X . By the question premise, we can assume a solution $\hat{\mathbf{w}}$ is obtained. Since SGD only find linear combinations of X , we can represent $\hat{\mathbf{w}} = X^T a$ for some $a \in \mathbb{R}^n$ such that $\hat{\mathbf{w}} \in \text{span}\{X\}$. Let \mathbf{w}_1 be any other solution to the linear regression model. We have:

$$\begin{aligned} (\hat{\mathbf{w}} - \mathbf{w}_1)^T \hat{\mathbf{w}} &= (\hat{\mathbf{w}} - \mathbf{w}_1)^T X^T a \\ &= (X(\hat{\mathbf{w}} - \mathbf{w}_1))^T a \\ &= (\mathbf{t} - \mathbf{t})^T a = 0 \end{aligned}$$

Using this orthogonality condition and the generalized Pythagorean theorem, we can show that $\hat{\mathbf{w}}$ has the smallest norm of all solutions, also implying $\hat{\mathbf{w}} = \mathbf{w}^*$.

Geometric interpretation: the minimum norm solution lies in the span of X , and is orthogonal to all other solutions. This implies all other solutions involve an addition of a vector in the null space of X (due to the fundamental theorem of linear algebra). The updates in SGD do not involve any vectors in the null space of X . Assuming a solution is found, then it must be orthogonal to the null space of X .

1.1.2 SGD with Momentum [1pt]

As a corollary to Question 1.1.1, consider SGD with momentum. The update step at time t with scalars α and η is described as:

$$\delta_{t+1} = -\eta \nabla \mathcal{L}_i(\mathbf{x}_i) + \alpha \delta_t \quad (3)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \delta_{t+1} \quad (4)$$

Under the assumptions in Question 1.1.1, provide an intuitive argument regarding whether stochastic gradient descent with momentum obtains the minimum norm solution on convergence.

Hint: Consider the effects of the momentum term on the linearity of gradient update.

Answer

Yes. Solution essentially follows from solution in 1.1.1. Since momentum terms do not affect the linearity of the SGD update steps, the same arguments can be made to show that SGD recovers the minimum norm solution.

1.2 Adaptive Methods

We will next consider the behavior of adaptive gradient descent methods. In particular, we will investigate the Adagrad 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 shown by:

$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{G_{i,t}} + \epsilon} \nabla_{w_{i,t}} \mathcal{L}(w_{i,t}) \quad (5)$$

$$G_{i,t} = G_{i,t-1} + (\nabla_{w_{i,t}} \mathcal{L}(w_{i,t}))^2 \quad (6)$$

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 (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 Adagrad optimizer converges to a solution. Provide a proof or counterexample for whether Adagrad always obtains the minimum norm solution.

Hint: Compute the 2D case from HW1. Let $\mathbf{x}_1 = [1, 1]$, $w_0 = [0, 0]$, $t = [3]$.

Answer

It should be clear after a few updates that the updates take the weights outside of the span of X . The crux of the issue is that the adaptive gradient "corrections" break the linearity of the data, and it can be extrapolated that the updates will in most cases reach a solution which is not minimum norm. In a particularly degenerate case, there exists some learning rate which perfectly takes Adagrad to a non-minimum norm solution after just a 2 steps (that students can potentially find). Sufficient also to show that the update after a few steps does not lie within the span of X , meaning previous proof is violated.

1.2.2 [0pt]

Consider the result from the previous section. Does this result hold true for other adaptive methods (RMSprop, 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 linear regression problem, like in homework 1.

Similar to homework 1, a linear model will be used for this 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 $\hat{\mathbf{w}} \in \mathbb{R}^d$ that minimizes the squared error of prediction on the training samples. 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:

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

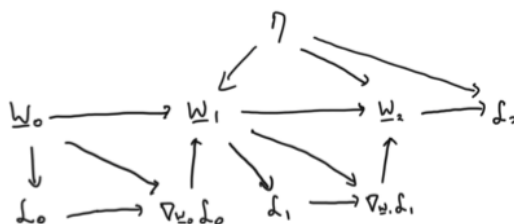
Starting with an initial weight parameters \mathbf{w}_0 , gradient descent (GD) updates \mathbf{w}_0 with a learning rate η for t number of iterations. Let's denote the weights after t iterations of GD as \mathbf{w}_t , the loss as \mathcal{L}_t , and its gradient as $\nabla_{\mathbf{w}_t}$. The goal is to 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 [1pt]

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

Answer



2.1.2 [1pt]

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, $\nabla_{\eta} \mathcal{L}_t$ in terms of t ?

Answer

$O(1)$ for forward-propagation, $O(t)$ for back-prop

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.

Answer

Potential memory issue, or vanishing gradient.

2.2 Optimal Learning Rates

In this section, we will take a closer look at the gradient w.r.t. the learning rate. 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 [0pt]

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 $\mathbf{a} = X\mathbf{w}_0 - \mathbf{t}$

Answer

$$\mathbf{w}_1 = \mathbf{w}_0 - \frac{2\eta}{n} X^T \mathbf{a}, \mathcal{L}_1 = \frac{1}{2} \mathbf{a}^T \left(-\frac{2\eta}{n} X X^T + I \right)^2 \mathbf{a}.$$

Or by introducing a constant factor to the loss to make the expression cleaner ($\mathcal{L} = \frac{1}{2} \|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2$)
 $\mathbf{w}_1 = \mathbf{w}_0 - \eta X^T \mathbf{a}, \mathcal{L}_1 = \frac{1}{2} \mathbf{a}^T (-\eta X X^T + I)^2 \mathbf{a}.$

2.2.2 [1pt]

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

Answer

\mathcal{L} is convex since positive order 2 polynomial w.r.t. η . Or show second order derivative, $\frac{8}{n^3} \|X X^T \mathbf{a}\|_2^2$, is positive (e.g. because dot product is non-negative and trivial case when equal to zero).

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.

Answer

$$\frac{\partial \mathcal{L}_1}{\partial \eta} = \mathbf{a}^T (-\eta X X^T + I) (-X X^T) \mathbf{a} = 0. \text{ Thus, } \eta = \frac{(X^T \mathbf{a})^2}{(X X^T \mathbf{a})^2}.$$

2.2.4 [0pt]

Using the 2D over-parameterized case from homework 1 (i.e. $X = [1; 1]$ and $\mathbf{t} = [3]$), calculate the optimal η^* ? Use η^* to calculate \mathbf{w}_1 and compare this with the result you obtained from HW1. How does the optimal η^* help reach this solution? (i.e. describe the trajectory)

Answer

$\eta = \frac{1}{2}$. Get the same \mathbf{w}_1 from homework 1. Learning rate such that in a single gradient descent, reach optimal \mathbf{w} .

2.3 Multiple Inner-loop Iterations**2.3.1 [1pt]**

Derive the expression of the loss \mathcal{L}_t after t iterations of GD. Show your work.

Hint: proof by induction and binomial coefficients can be useful

Answer

$$\mathcal{L}_t = \frac{1}{2} \mathbf{a}^T (-\eta X X^T + I)^{2t} \mathbf{a}.$$

This can be shown by using proof by induction or simply showing sufficient number of L_t terms and stating a pattern. Should show at least 3 terms (not sure if it is valid to claim a pattern with just one or two...).

2.3.2 [1pt]

Determine if in general \mathcal{L}_t is convex w.r.t. the learning rate η ?

Answer

Multiple possible ways to show convexity.

After decomposition, $XX^T = Q\Lambda Q^T$.

Then, $\mathcal{L}_t = \frac{1}{2} \mathbf{a}^T Q (-\eta\Lambda + I)^{2t} Q^T \mathbf{a} = \frac{1}{2} \sum_{i=1}^n b_i^2 (-\eta\lambda_i + 1)^{2t}$ where b_i, λ_i are i'th element of $Q^T \mathbf{a}, \Lambda$.

$$\frac{\partial^2 \mathcal{L}_t}{\partial \eta^2} = t(2t-1) \sum_{i=1}^n b_i^2 \lambda_i^2 (-\eta\lambda_i + 1)^{2t-2} > 0$$

Therefore, convex.

For those familiar with convex analysis, can show using composition rules.

Or can directly take the second derivative using matrix calculus (should only need few lines).

Partial marks taken off for randomly arranging terms without justification. Matrix multiplications are not commutative in general, so it is incorrect to arbitrarily arrange terms (e.g. a or XX^T), even though it happens to reach the same conclusion. Need some justification (e.g. stating it is valid for symmetric matrices, etc...).

Partial marks taken off for concluding non-negativity without some detailed justification (e.g. no justification, simply stating even power, or stating > 0 for terms that is not scalar unless it is very clear from the context this was meant as \succ, PD). For most cases, should have shown PD of the middle terms surround by a^T and a .

2.3.3 [0pt]

Using the 2D over-parameterized case from HW1 (i.e. $X = [1; 1]$ and $\mathbf{t} = [3]$), calculate the optimal η^* ?

Answer

In this case, the loss is convex and get the same $\eta = \frac{1}{2}$.

3 Convolutional Neural Networks

The last set of questions aims to build basic familiarity with Convolutional Neural Networks.

3.1 Convolutional Filters [1pt]

Given the input matrix \mathbf{I} and filter \mathbf{J} shown below, 1) Write down the values of the resulting matrix ($\mathbf{I} * \mathbf{J}$) (the convolution operation as defined in the Lec 4 slides). Assume we have use zero padding around the input. 2) What feature does this convolutional filter detect?

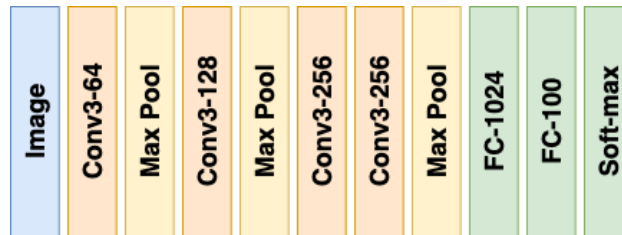
$$\mathbf{I} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \mathbf{J} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \mathbf{I} * \mathbf{J} = \begin{bmatrix} ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \end{bmatrix}$$

Answer

$$\mathbf{I} * \mathbf{J} = \begin{bmatrix} -1 & 2 & 2 & -2 & 0 \\ -2 & 1 & 0 & 2 & -1 \\ 3 & 0 & 0 & 1 & -1 \\ -2 & 2 & 0 & 2 & -1 \\ 0 & -2 & 3 & -2 & 0 \end{bmatrix}$$

3.2 Size of Conv Nets [1pt]

Consider a Conv Net with 4 conv layers like in the diagram below. All 4 conv layers have kernel size of 3×3 . The number after the hyphen specifies the number of output channels or units of a layer (e.g. *Conv3-64* layer has 64 output channels and *FC-1024* has 1024 output units). All the *Max Pool* in the diagram has size of 2×2 . Assume zero padding for conv layers and stride 2 for *Max Pool*.



Size of the RGB input image is 112×112 (3 channels).

Calculate the following: 1) the number of parameters for this Conv Net including the bias units, 2) the total number of neurons and 3) number of input connections for neurons in each layer.

Answer

Number of Neurons:

input: $112 \times 112 \times 3$, conv3-64: $112 \times 112 \times 64$, Max Pool1: $56 \times 56 \times 64$, Conv3-128: $56 \times 56 \times 128$, Max Pool2: $28 \times 28 \times 128$, Conv3-256-1: $28 \times 28 \times 256$, Conv3-256-2: $28 \times 28 \times 256$, FC-1024: 1024, FC-100: 100, Softmax: 100

Parameters:

Conv layers: $(9 \cdot 3 \cdot 64 + 64) + (9 \cdot 64 \cdot 128 + 128) + (9 \cdot 128 \cdot 256 + 256) + (9 \cdot 256 \cdot 256 + 256) + (9 \cdot 128 \cdot 256 + 256) = 1792 + 73856 + 295168 + 590080 = 960896$

Dense layers: $(14 \cdot 14 \cdot 256 \cdot 1024 + 1024) + (1024 \cdot 100 + 100) = 51381248 + 102500 = 51483748$

Total: 52444644

Number of connections:

conv3-64: $3 \times 3 \times 3$, Max Pool1: 2×2 , Conv3-128: $3 \times 3 \times 64$, Max Pool2: 2×2 , Conv3-256-1: $3 \times 3 \times 128$, Conv3-256-2: $3 \times 3 \times 256$, FC-1024: $28 \times 28 \times 256$, FC-100: 1024, Softmax: 100

3.3 Receptive field [0.5pt]

Receptive field of a neuron in a Conv Net is the area of the input that can affect the neuron (i.e. the area a neuron can ‘see’). For example, a neuron in a 3×3 conv layer is computed from an input area of 3×3 of the input, so its receptive field is 3×3 . A neuron in the next 3×3 conv layer is computed from an input area of 5×5 , so its receptive field is 5×5 . List 3 things that can affect the size of the receptive field of a neuron and briefly explain your answers.

Answer

Some of the possible answers:

- Size of the kernel
- Pooling
- Stride
- Depth of the layer