Homework 2 - Version 1.0

Deadline: Thursday, 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.*

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:

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

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 $\mathbf{x}_i \in \mathbb{R}^d$, $1 \le i \le n$ 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:

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

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

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?

1.1.1 $\nabla_{w_t} L_i = \nabla_{w_t}\left(\frac{1}{n}\|x_i^T w_t - t_i\|^2\right) = x_i \frac{1}{n}\cdot 2\left(x_i^T w_t - t_i\right)$

$= \frac{2}{n} x_i \left(x_i^T w_t - t_i\right)$ (defined as row space)

As we know x_i is the i^{th} row of X, thus $x_i \in \text{span}\{X\}$.

Then $\nabla_{w_t} L_i = \frac{2}{n} x_i \underbrace{\left(x_i w_t - t_i\right)}_{\in R}$ is a constant multiple of x_i, thus also in span $\{X\}$.

Then, (1) $w_0 = 0 \in \text{span}\{X^T\}$

(2) if $w_k \in \text{span}\{X^T\}$, then $w_{k+1} = w_k - \nabla_{w_t} L_i \in \text{span}\{X^T\}$

so if there is a convergent solution $\hat{w}$, it must also be in the span of X^T.

Let $\hat{w} = X^T\hat{c}$, then $X\hat{w} = t$ gives: $XX^T\hat{c} = t$

$\Rightarrow \hat{c} = (XX^T)^{-1}t \Rightarrow \hat{w} = X^T\hat{c} = X^T(XX^T)^{-1}t$

we then need to prove that $\hat{w}$ is the the minimum norm solution.

Consider any feasible vector w s.t. $Xw = t$.

We have: $\hat{w}^T(\hat{w} - w) = (X^T c)^T(\hat{w} - w) = c^T(X\hat{w} - Xw) = c^T(t - t) = 0$

This means that $\hat{w}^T$ is orthogonal to $\hat{w} - w$.

Thus, $\|\hat{w}\|^2 + \|\hat{w} - w\|^2 = \|w\|^2$

so $\|\hat{w}\|^2 \le \|w\|^2$ meanin $\hat{w}$ is the min norm sol

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:

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \delta_{t+1} \tag{4}$$

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.

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:

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

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

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 $\mathbf{x}_1 = [1, 1]$, $w_0 = [0, 0]$, $t = [3]$.

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 desireable?

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

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:

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \delta_{t+1} \tag{4}$$

$\delta_0 = 0$

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.

(1) $\delta_0 = 0 \in \text{span}\{X^T\}$

(2) If $\delta_k \in \text{span}\{X^T\}$, since $\nabla L_i(x_i) \in \text{span}\{X^T\}$, then

$\delta_{k+1} = -\eta \nabla L_i(x_i) + \alpha \delta_k \in \text{span}\{X^T\}$.

Thus, by induction $\delta_t \in \text{span}\{X^T\}$ $\forall t$.

We then use the same argument as in 1.1.1 to show that $w_t \in \text{span}\{X^T\}$ $\forall t$ and so w converges to the minimum norm solution.

Intuitively, the momentum term doesn't affect linearity of gradient update.

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 $\mathbf{x}_1 = [1, 2]$, $w_0 = [0, 0]$, $t = [3]$.

1.2.1 Consider $x_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$, $w_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $t = (3)$. Then $n = 1$, $d = 2$. Let $\epsilon = 0.1$. $\eta = 0.01$.

$L_i x_i = \nabla_{w_t} \left(\frac{1}{n} \| x_i^T w_t - t_i \|^2 \right)$

$\nabla L_1(w_1) = \frac{2}{\eta} x_1 (x_1^T w_1 - t_1) = 2 \begin{pmatrix} 1 \\ 2 \end{pmatrix} (0 - 3) = \begin{pmatrix} -6 \\ -12 \end{pmatrix}$.

$G_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 36 \\ 144 \end{pmatrix} = \begin{pmatrix} 36 \\ 144 \end{pmatrix}$.

$w_1 = w_0 - 0.01 \times \begin{pmatrix} \frac{1}{6+0.1} \cdot (-6) \\ \frac{1}{12+0.1} \cdot (-12) \end{pmatrix} = \begin{pmatrix} 0.009836 \\ 0.009917 \end{pmatrix}$

However, this solution is not in the span of $\{ \begin{pmatrix} 1 \\ 2 \end{pmatrix} \}$, thus it's not a minimum norm sol.

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 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]

Condier 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.

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?

2.1.3 [0pt]

Memory issue / vanishing gradient

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

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

2.2.3 [1pt]

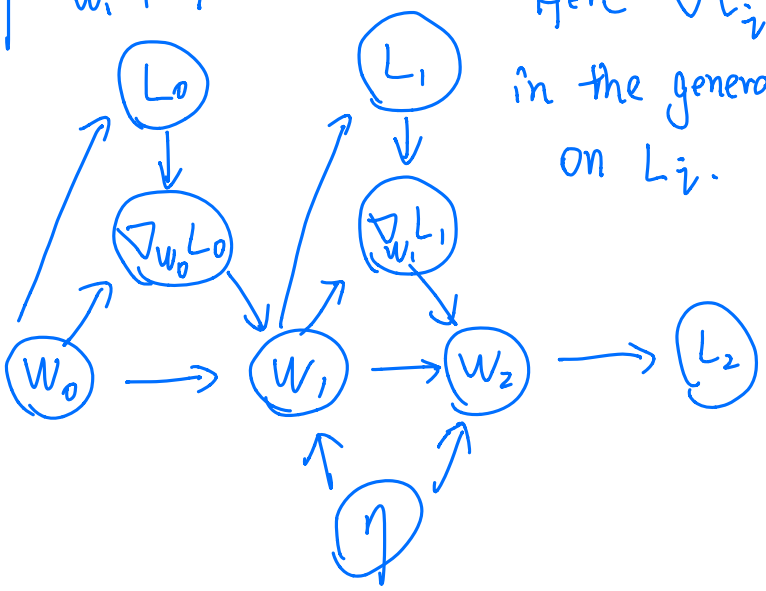
Write down the derivative of $\mathcal{L}_1$ w.r.t. η and use it to find the optimal learning rate η^* that minimizes the loss after one GD iteration. Show your work.

2.1.1 **[0.5pt]**

Condier 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.

$w_1 = w_0 - \eta \cdot \nabla_{w_0} L_0$
$w_2 = w_1 - \eta \cdot \nabla_{w_1} L_1$
$\Rightarrow L_2 = \frac{1}{n}\|Xw_2 - t\|^2$

Here ∇L_i depends only on w_i, but in the general case, it also depends on L_i.



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?

In forward propogation process, we only need to store one weight (w_i) at a time, so the complexity is $O(1)$. In back-propogation process, since $w_1, w_2, \cdots, w_t$ all depend on η, then we need to store $\nabla_\eta w_1, \nabla_\eta w_2, \cdots, \nabla_\eta w_t$ at the same time, thus $O(t)$ memory is required.

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

$$w_1 = w_0 - \eta \cdot \nabla_{w_0} L_0(w_0)$$

$$= w_0 - \underbrace{\eta \cdot \frac{2}{n}}_{\alpha} \cdot X^T \underbrace{(Xw_0 - t)}_{=\vec{a}} \qquad \alpha = \eta \cdot \frac{2}{n}$$

$$= \vec{w_0} - \alpha \cdot X^T \vec{a}$$

$$n \cdot L_1 = \|Xw_1 - t\|^2 = \|X(\vec{w_0} - \alpha X^T \vec{a}) - t\|^2$$

$$= \|X\vec{w}_0 - t - \alpha X X^T \vec{a}\|^2$$

$$= \|\vec{a} - \alpha \cdot X X^T \vec{a}\|^2$$

$$= \|(I - \alpha X X^T)\vec{a}\|^2$$

$$= \|(I - \eta \cdot \tfrac{2}{n} X X^T)\vec{a}\|^2$$

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

Let $M = -\frac{2}{n} X X^T$.

Then $L_1 = \|(I + \eta M)\vec{a}\|^2 = \|\vec{a} + \eta M \vec{a}\|^2$

$$= (\|a\|^2 + \eta^2 \|Ma\|^2 + 2\eta\, a^T M a) \cdot \frac{1}{n}$$

$$\frac{\partial L_1}{\partial \eta} = (2\eta \|Ma\|^2 + 2a^T M a) \cdot \frac{1}{n}$$

$$\frac{\partial^2 L_1}{\partial \eta^2} = \frac{2}{n}\|Ma\|^2 > 0$$

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.

$$\frac{\partial \mathcal{L}_1}{\partial \eta} = \left(2\eta \|Ma\|^2 + 2a^T M a\right) \cdot \frac{1}{n} = 0$$

$$\Rightarrow \eta^* = -\frac{a^T M a}{\|Ma\|^2} \quad \text{where } M = -\frac{2}{n} X X^T,\ a = Xw_0 - t$$

This indeed is a minimum because $\mathcal{L}_1$ is convex.

$w_0 = \vec{0}$, $X = (1\ 1)$, $t = 3$, $a = Xw_0 - t = -3$, $M = -\frac{2}{n} XX^T = -4$,

$\eta = \frac{1}{4} \Rightarrow \nabla_{w_0} L_0(w_0) = \nabla_{w_0}\left(\frac{2}{n} X^T(Xw_0 - t)\right) = \begin{pmatrix} -6 \\ -6 \end{pmatrix} \Rightarrow w_1 = w_0 - \eta \cdot \nabla_{w_0} L_0(w_0) = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$

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)

projection onto the optimal plane

2.3 Multiple Inner-loop Iterations

2.3.1 [0.5pt]

✓

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

2.3.2 [1pt]

✓

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

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^*?

3 Convolutional Neural Networks

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

3.1 Convolutional Filters [0.5pt]

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?

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

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*.

2.3 Multiple Inner-loop Iterations

2.3.1 [0.5pt]

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

Let $a_i = Xw_i - t$,

$$a_{k+1} = Xw_{k+1} - t = X(w_k - \eta \cdot \nabla_{w_k} L) - t$$

$$= X\left(w_k - \frac{2\eta}{n} X^T(Xw_k - t)\right) - t$$

$$= Xw_k - t - \frac{2\eta}{n} XX^T(Xw_k - t)$$

$$= \left(I - \frac{2\eta}{n} XX^T\right)(Xw_k - t)$$

$$= \left(I - \frac{2\eta}{n} XX^T\right) a_k$$

Thus, $a_m = \left(I - \frac{2\eta}{n} XX^T\right)^m a_0 \quad \forall m \in \mathbb{N}$

So $L_m = \|Xw_m - t\|^2 = \|a_m\|^2$

$= \left\|\left(I - \frac{2\eta}{n} XX^T\right)^m a_0\right\|^2$ after m^{th} iteration

2.3.2 **[1pt]**

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

$$L_m = a_0^T\left[\left(I - \frac{2\eta}{n} XX^T\right)^m\right]^T \left(I - \frac{2\eta}{n} XX^T\right)^m a_0$$

$$= a_0^T \left(I - \frac{2\eta}{n} XX^T\right)^{2m} a_0 \quad \text{since } \left(I - \frac{2\eta}{n} XX^T\right) \text{ is symmetric}$$

Since XX^T is symmetric, then it has a spectral decomposition: $XX^T = Q^T \Lambda Q$, where Q is orthogonal and Λ is diagonal. And entries of Λ are positive since XX^T is positive definite. (entries of Λ denoted λ_i)

Then $I - \frac{2\eta}{n} XX^T = Q^T I Q - \frac{2\eta}{n} Q^T \Lambda Q$

$$= Q^T \left(I - \frac{2\eta}{n} \Lambda\right) Q$$

and $\left(I - \frac{2\eta}{n} XX^T\right)^{2m} = Q^T \left(I - \frac{2\eta}{n} \Lambda\right)^{2m} Q$

so $L_m = a_0^T Q^T \left(I - \frac{2\eta}{n} \Lambda\right)^{2m} Q a_0$

$$= b^T \left(I - \frac{2\eta}{n} \Lambda\right)^{2m} b, \text{ where } b = Q a_0$$

and $I - \frac{2\eta}{n} \Lambda$ is diagonal, so entries of $\left(I - \frac{2\eta}{n} \Lambda\right)^{2m}$ are given by $\left(1 - \frac{2\eta}{n} \lambda_i\right)^{2m}$, where $\lambda_i > 0$.

Thus, $L_m = \sum_{i=1}^{n} \left(1 - \frac{2\eta}{n}\lambda_i\right)^{2m} \cdot b_i^2$

$$\frac{\partial^2 L_m}{\partial \eta^2} = \sum_{i=1}^{n} b_i^2 \cdot 2m \cdot (2m-1)\left(1 - \frac{2\eta}{n}\lambda_i\right)^{2m-2} \cdot \left(-\frac{2\lambda_i}{n}\right)^2 > 0$$

Thus L_m is convex w.r.t η.

3.1 Convolutional Filters [0.5pt]

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?

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

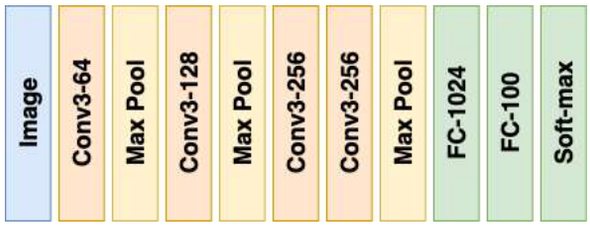
1) $\begin{pmatrix} -1 & 2 & 2 & -2 & 0 \\ -2 & 1 & 0 & 2 & -1 \\ 3 & 0 & 0 & 1 & -1 \\ -2 & 2 & 0 & 2 & -1 \\ 0 & -2 & 3 & -2 & 0 \end{pmatrix}$

2) Edge detection.

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*.

112×112×3 3×3×3×64



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.

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.

3.2

The size of each output layer are as follows:

Image 112×112×3 → (3×3) Conv 3-64 112×112×64 → (2×2) Max Pool 56×56×64

→ (3×3) Conv 3-128 56×56×128 → (2×2) Max Pool 28×28×128

→ (3×3) Conv 3-256 28×28×256 → (3×3) Conv 3-256 28×28×256

→ (2×2) Max Pool 14×14×256 → FC-1024 1024×1

→ FC-100 100×1 → Softmax 100×1

1) # of parameters = Conv 3-64 $(3\times3\times3\times64+64)$

+ Conv 3-128 $(3\times3\times64\times128+128)$ + Conv 3-256 $(3\times3\times128\times256+256)$

+ Conv 3-256 $(3\times3\times256\times256+256)$ + FC-1024 $(14\times14\times256\times1024+1024)$

+ FC-100 $(1024\times100+100)$ = 52444644

2) # of neurons = Conv 3-64 $112\times112\times64$ + Max Pool $56\times56\times64$ + Conv 3-128 $56\times56\times128$

+ Max Pool $28\times28\times128$ + Conv 3-256 $28\times28\times256$ + Conv 3-256 $28\times28\times256$

+ Max Pool $14\times14\times256$ + FC-1024 1024 + FC-100 100 + softmax 100

= 1958088

3) # of input connections for each layer:

Cov 3-64: $3\times3\times3\times64\times112^2 = 21676032$

Max Pool: $4\times56\times56\times64 = 802816$

Cov 3-128: $64\times3\times3\times56^2\times128 = 231211008$

Max Pool: $4\times28\times28\times128 = 401408$

Cov 3-256: $3\times3\times128\times28^2\times256 = 231211008$

Cov 3-256: $3\times3\times256\times28^2\times256 = 462422016$

Max Pool: $4\times14\times14\times256 = 200704$

FC-1024: $14\times14\times256\times1024 = 51380224$

FC-100: $1024\times100 = 102400$

Softmax: $100\times100 = 10000$

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.

1) Number of layers. Each output neuron contains information from all input units inside the $K \times K$ kernel. As a result, higher the # of layers, wider the range of units that can affect an output neuron.

2) The kernel size. This directly reflects how many input values can affect the output neuron in the next layer.

3) The stride value: A larger stride value can make far apart input units "close" in the output layer, thereby increasing the receptive field.