

# CS282-specialParticipationA-HW2



## 1. Optimizers as Penalized Linear Improvement with different norm penalties

In lecture, you saw the locally linear perspective of a neural network and the loss by Taylor expanding the loss around the current value of the parameters. This approximation is only very good in a near neighborhood of those values. One way to proceed with optimization is to consider the size of the neighborhood as a hyperparameter and to bound our update to stay within that neighborhood while minimizing our linear approximation to the loss. You saw in lecture that the choice of norm in defining that neighborhood also matters.

In this problem, you will work out for yourself a slightly different perspective. Instead of treating the norm as a constraint (with the size of the acceptable norm as a hyperparameter), we can do an unconstrained optimization with a weighted penalty that corresponds to the squared norm — where that weight is a hyperparameter.

At each iteration, we wish to maximize linear improvement of the objective (as defined by the dot-product between the gradient and the update) locally regularized by a penalty on the size of the update. This can be expressed (in traditional minimization form) as:

$$u = \operatorname{argmin}_{\Delta\theta} \underbrace{g^T \Delta\theta}_{\text{Linear Improvement}} + \frac{1}{\alpha} \underbrace{d(\Delta\theta)}_{\text{Distance Penalty}}, \quad (1)$$

where  $g = \nabla f(\theta)$  is the gradient of the loss,  $\alpha$  is a scalar, and  $d$  is a scalar-output distance function  $\mathbb{R}^{\dim(\theta)} \rightarrow \mathbb{R}^+$ .

- (a) Let's assume *Euclidean distance* is the norm that captures our sense of relevant neighborhoods in parameter space. Then our objective can be:

$$u = \operatorname{argmin}_{\Delta\theta} g^T \Delta\theta + \frac{1}{\alpha} \|\Delta\theta\|_2^2. \quad (2)$$

**What is the analytical solution for  $u$  in the above problem? What standard optimizer does this recover?**

Please solve the problem attached

10/13/2025, 12:47:31 PM

*Thought: Unraveled gradient descent optimizer through analytical solution.*

Looking at this optimization problem, I need to find the analytical solution for equation (2).

## Finding the Analytical Solution

The problem is to minimize:  $J(\Delta\theta) = g^T \Delta\theta + \frac{1}{2\alpha} \|\Delta\theta\|_2^2$

where  $\|\Delta\theta\|_2^2 = \Delta\theta^T \Delta\theta$ .

Expanding this:  $J(\Delta\theta) = g^T \Delta\theta + \frac{1}{2\alpha} \Delta\theta^T \Delta\theta$

To find the minimum, I take the derivative with respect to  $\Delta\theta$  and set it equal to zero:

$$\frac{\partial J}{\partial \Delta\theta} = g + \frac{1}{\alpha} \Delta\theta = 0$$

Solving for  $\Delta\theta$ :  $\Delta\theta = -\frac{\alpha}{2} g$

Therefore, the optimal update is:

$$u = -\frac{\alpha}{2} g$$

## What Optimizer Does This Recover?

This recovers **Gradient Descent** (also known as Steepest Descent).

The update rule is:  $\theta_{\text{new}} = \theta_{\text{old}} + u = \theta_{\text{old}} - \frac{\alpha}{2} g$

If we define the learning rate as  $\eta = \frac{\alpha}{2}$ , this becomes:  $\theta_{\text{new}} = \theta_{\text{old}} - \eta \nabla f(\theta)$

which is exactly the standard gradient descent update rule.

**Key Insight:** The Euclidean ( $L_2$ ) penalty on the update size leads naturally to gradient descent. The hyperparameter  $\alpha$  controls the trade-off between maximizing linear improvement (following the gradient direction) and keeping the update small, which is equivalent to choosing a learning rate.





(b) Now, consider an alternative way of capturing local neighborhood size – the squared infinity norm over parameters. Recall that this is defined as  $\|x\|_\infty = \max_i |x_i|$ . Our objective is now:

$$u = \operatorname{argmin}_{\Delta\theta} g^T \Delta\theta + \frac{1}{\alpha} \|\Delta\theta\|_\infty^2. \quad (6)$$

**What is the analytical solution for  $u$  in this case? Which optimizer does this correspond to?**

10/13/2025, 12:56:40 PM

*Thought: Unraveled gradient descent optimization with infinity norm penalty.*

Looking at problem (b) with the infinity norm penalty.

## Finding the Analytical Solution

The problem is:  $u = \operatorname{argmin}_{\Delta\theta} g^T \Delta\theta + \frac{1}{\alpha} \|\Delta\theta\|_\infty^2$

Let me denote  $m = \|\Delta\theta\|_\infty = \max_i |\Delta\theta_i|$ .

For a fixed value of  $m$ , I want to minimize the linear term  $g^T \Delta\theta$  subject to the constraint  $|\Delta\theta_i| \leq m$  for all  $i$ .

To maximize the decrease (minimize  $g^T \Delta\theta$ ), I should set each component to move maximally against its gradient:  $\Delta\theta_i = -m \cdot \text{sign}(g_i)$

This gives:  $g^T \Delta\theta = \sum_i g_i (-m \cdot \text{sign}(g_i)) = -m \sum_i |g_i| = -m |g|_1$  [Here it provides more detailed expansion compared to the HW solution.](#)

Substituting back into the objective:  $J(m) = -m |g|_1 + \frac{m^2}{\alpha}$

To find the optimal  $m$ , I take the derivative and set it to zero:  $\frac{dJ}{dm} = -|g|_1 + \frac{2m}{\alpha} = 0$

Solving for  $m$ :  $m = \frac{\alpha \|g\|_1}{2}$

Therefore, the optimal update is:

$$u_i = -\frac{\alpha \|g\|_1}{2} \cdot \text{sign}(g_i)$$

Or in vector form:  $\boxed{u = -\frac{\alpha \|g\|_1}{2} \cdot \text{sign}(g)}$

## What Optimizer Does This Correspond To?

This corresponds to **Sign Descent** (also known as **SignSGD** or **signed gradient descent**).

The key characteristic is that the update uses only the **sign** of each gradient component, not its magnitude:  $\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \text{sign}(\nabla f(\theta))$

In our case, the effective learning rate is  $\eta = \frac{\alpha \|g\|_1}{2}$ , which adapts based on the L1 norm of the gradient.

Although it pointed out the learning rate would involve scaling by  $\|g\|_1$ , but it does not explicitly mention how this is different from Sign GD. And it's unable to generalize the update patterns at first.

### Key Insights:

- The  $L_\infty$  penalty leads to all components of the update having the same absolute value
- Only the sign of each gradient component matters for the direction
- The magnitude is determined globally by the L1 norm of the entire gradient
- This method is robust to the scale of individual gradient components and can be more stable in practice than standard gradient descent

10/13/2025, 12:57:38 PM

👤 Are you sure this is the same as sign Descent? Also, can you derive a more general pattern from the above two solutions?

10/13/2025, 1:09:07 PM

But after I provided a bit more guidance, it was able to recover both cases and further provided a more in-depth explanation about the dual norm patterns compared to HW solution.

*Thought: Uncovered nuanced pattern in gradient descent optimization techniques.*

You're absolutely right to question that! Let me reconsider more carefully.

## Is This Really SignSGD?

Not quite. In standard **SignSGD**, the update is:  $\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \text{sign}(g)$

where  $\eta$  is a fixed learning rate.

However, in my solution:  $u = -\frac{\alpha \|g\|_1}{2} \cdot \text{sign}(g)$

the effective learning rate is  $\frac{\alpha \|g\|_1}{2}$ , which **adapts** based on the L1 norm of the gradient. So this is more like an **adaptive version of sign descent** where the step size scales with the total magnitude of the gradient.

## The General Pattern: Dual Norms

Looking at both solutions, there's a beautiful pattern related to **dual norms**!

Recall that norms come in dual pairs where  $\frac{1}{p} + \frac{1}{q} = 1$ :

- **L2 is self-dual:**  $(L_2)^* = L_2$  (since  $p=q=2$ )
- **$L_\infty$  and L1 are dual:**  $(L_\infty)^* = L_1$  and  $(L_1)^* = L_\infty$

## The Pattern:

When we use an  $\|\Delta\theta\|_p$  penalty, the optimal update direction is determined by the **dual norm** structure:

Penalty Norm	Dual Norm	Update Direction	Optimizer
$ \cdot _2$	$ \cdot _2$	$u \propto -g$	Gradient Descent
$ \cdot _\infty$	$ \cdot _1$	$u \propto -\text{sign}(g)$ , magnitude $\propto  g _1$	Adaptive Sign Descent
$ \cdot _1$	$ \cdot _\infty$	$u_i = -\alpha \cdot g_i /  g _\infty$ (normalized by max)	Would give another variant

## Why This Happens:

By Hölder's inequality:  $|g^T \Delta\theta| \leq \|g\|_q \|\Delta\theta\|_p$

The maximum decrease in the linear term (subject to bounded  $\|\Delta\theta\|_p$ ) is achieved when  $\Delta\theta$  aligns with the dual norm structure of  $g$ .

- **For L2:** All components matter proportionally  $\rightarrow$  update is  $-g$
- **For L $\infty$ :** Only the direction matters, all components move equally  $\rightarrow$  update is  $-\text{sign}(g)$

This is a fundamental principle in optimization: **the geometry of the constraint (penalty) determines the structure of the optimal solution through duality.**

10/13/2025, 1:09:46 PM



## 2. Optimizers and their convergence

Consider  $\mathcal{O}$ : a simplified Adam-style optimizer without weight decay that has iterates

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha_t M_t \nabla f_t(\boldsymbol{\theta}_t) \quad (14)$$

where  $f_t$  is the loss at iteration  $t$  and  $\alpha_t$  is the step size (learning rate).

Further suppose that the adaptive scaling matrix  $M_t$  is recomputed over each epoch of training and just consists of a diagonal populated by the inverses of the square roots of the mean squared value for the gradients during the epoch for that specific coordinate.

For this part, we have exactly  $n = 1$  training point corresponding to the single equation

$$[1, 0.1, 0.01]\boldsymbol{\theta} = 1 \quad (15)$$

with a 3-dimensional learnable parameters  $\boldsymbol{\theta}$ . Suppose that we start with  $\boldsymbol{\theta}_0 = \mathbf{0}$  and use squared loss  $f_t(\boldsymbol{\theta}) = (1 - [1, 0.1, 0.01]\boldsymbol{\theta})^2$ .

- (a) **What specific vector  $\boldsymbol{\theta}$  would standard vanilla SGD (i.e. (14) with  $M_t = I$  and  $\alpha_t = \alpha$ ) converge to assuming  $\alpha > 0$  was small enough to give convergence?**

Please solve the problem attached

10/13/2025, 1:21:08 PM

*Thought: Unraveled SGD convergence for linear equation optimization.*

Looking at this problem, I need to analyze where vanilla SGD converges for this simple linear regression problem.

## Setting up the problem

Given:

- Equation:  $\mathbf{a}^T \boldsymbol{\theta} = 1$  where  $\mathbf{a} = [1, 0.1, 0.01]^T$
- Loss:  $f(\boldsymbol{\theta}) = (1 - \mathbf{a}^T \boldsymbol{\theta})^2$
- Initial:  $\boldsymbol{\theta}_0 = \mathbf{0}$
- Update:  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla f(\boldsymbol{\theta}_t)$

## Computing the gradient



$$\nabla f(\boldsymbol{\theta}) = 2(1 - \mathbf{a}^T \boldsymbol{\theta})(-\mathbf{a}) = -2(1 - \mathbf{a}^T \boldsymbol{\theta})\mathbf{a}$$

## The SGD update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + 2\alpha(1 - \mathbf{a}^T \boldsymbol{\theta}_t)\mathbf{a}$$

Let  $r_t = 1 - \mathbf{a}^T \boldsymbol{\theta}_t$  denote the residual. Then:  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + 2\alpha r_t \mathbf{a}$

## Analyzing convergence

Taking the inner product with  $\mathbf{a}$ :  $\mathbf{a}^T \boldsymbol{\theta}_{t+1} = \mathbf{a}^T \boldsymbol{\theta}_t + 2\alpha r_t \mathbf{a}^T \mathbf{a}$

Therefore:  $r_{t+1} = 1 - \mathbf{a}^T \boldsymbol{\theta}_{t+1} = r_t - 2\alpha r_t \mathbf{a}^T \mathbf{a} = r_t(1 - 2\alpha \|\mathbf{a}\|^2)$

Computing  $\|\mathbf{a}\|^2$ :  $\|\mathbf{a}\|^2 = 1^2 + 0.1^2 + 0.01^2 = 1 + 0.01 + 0.0001 = 1.0101$

For convergence, we need  $|1 - 2\alpha \cdot 1.0101| < 1$ , which gives  $\alpha < \frac{1}{1.0101}$ .

## Finding the limit

Since  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + 2\alpha r_t \mathbf{a}$  and all updates are in the direction of  $\mathbf{a}$ :

$$\boldsymbol{\theta}_t = 2\alpha \left( \sum_{i=0}^{t-1} r_i \right) \mathbf{a}$$

As  $t \rightarrow \infty$ , the geometric series sums to:  $\sum_{i=0}^{\infty} r_i = \frac{r_0}{1 - 2\alpha \|\mathbf{a}\|^2} = \frac{1}{1 - 2\alpha \cdot 1.0101}$

Therefore:  $\boxed{\boldsymbol{\theta}_{\infty} = \frac{\mathbf{a}}{\|\mathbf{a}\|^2} = \frac{[1, 0.1, 0.01]}{1.0101} = \left[ \frac{10000}{10101}, \frac{1000}{10101}, \frac{100}{10101} \right]}$

Or numerically:  $\theta_\infty \approx [0.9901, 0.09901, 0.009901]^T$

**Key insight:** Vanilla SGD converges to the **minimum  $L_2$  norm solution** that satisfies the equation  $\mathbf{a}^T \theta = 1$ . This is because SGD starting from  $\mathbf{0}$  always updates in the direction of  $\mathbf{a}$  (the gradient direction), so the solution lies in  $\text{span}(\mathbf{a})$ .

10/13/2025, 1:21:49 PM

⊙ As you mentioned, SGD starting from 0 always updates in the direction of  $\mathbf{a}$  (the gradient direction), so the solution lies in  $\text{span}(\mathbf{a})$ . Can you think of a simpler solution?

10/13/2025, 1:27:29 PM

It's able to one-shot the problem in an over-complicated way. After I provided a bit more guidance, it's able to come up with the same approach (much cleaner) as provided in the HW solution.

*Thought: Simplified solution by leveraging  $\text{span}(\mathbf{a})$  and convergence constraint.*

Ah yes, excellent point! There's a much simpler way to solve this.

## Simpler Solution

Since SGD starting from  $\theta_0 = \mathbf{0}$  always updates in the direction of  $\mathbf{a}$ , we know:

$$\theta_t = c_t \mathbf{a}$$

for some scalar  $c_t$ .

At convergence, the loss must be zero (assuming  $\alpha$  is small enough), which means:  $\mathbf{a}^T \theta_\infty = 1$

Substituting  $\theta_\infty = c_\infty \mathbf{a}$ :  $\mathbf{a}^T (c_\infty \mathbf{a}) = 1$   $c_\infty |\mathbf{a}|^2 = 1$   $c_\infty = \frac{1}{|\mathbf{a}|^2}$

Therefore:  $\boxed{\theta_\infty = \frac{\mathbf{a}}{|\mathbf{a}|^2}}$

**That's it!** No need for the geometric series or residual analysis. Just:

1. Recognize the solution is in  $\text{span}(\mathbf{a})$
2. Write  $\theta_\infty = c\mathbf{a}$
3. Solve for  $c$  using the constraint that the loss is zero

Much cleaner!

10/13/2025, 1:27:42 PM



Great. Then what specific vector  $\theta$  would the simplified version of Adam O converge to assuming appropriate step-sizes  $\alpha_t > 0$  to give convergence?

10/13/2025, 1:30:22 PM

It's able to one-shot the problem, with additional step-by-step explanations of how the observation of  $M_t$  is obtained.

*Thought: Unraveled Adam optimizer's convergence to unique parameter vector.*

Great question! Let me work through what happens with the Adam optimizer  $\mathcal{O}$ .

## Analyzing the Adam Update

The gradient at iteration  $t$  is:  $\nabla f(\boldsymbol{\theta}_t) = -2(1 - \mathbf{a}^T \boldsymbol{\theta}_t) \mathbf{a} = -2r_t \mathbf{a}$

where  $r_t = 1 - \mathbf{a}^T \boldsymbol{\theta}_t$  is the residual.

For coordinate  $i$ , the gradient component is:  $g_t^{(i)} = -2r_t a_i$

## Computing the Adaptive Scaling Matrix $M_t$

The mean squared gradient for coordinate  $i$  during the epoch is:  $(g_t^{(i)})^2 = 4r_t^2 a_i^2$

Therefore, the diagonal entry of  $M_t$  is:  $M_t^{(i)} = \frac{1}{\sqrt{4r_t^2 a_i^2}} = \frac{1}{2|r_t| |a_i|}$

Since  $r_t > 0$  (assuming convergence from below) and all  $a_i > 0$ :  $M_t^{(i)} = \frac{1}{2r_t a_i}$

## The Update Rule

$$\theta_{t+1}^{(i)} = \theta_t^{(i)} - \alpha_t M_t^{(i)} g_t^{(i)} = \theta_t^{(i)} - \alpha_t \cdot \frac{1}{2r_t a_i} \cdot (-2r_t a_i)$$

$$= \theta_t^{(i)} + \alpha_t$$

**Key observation:** All coordinates increase by exactly the same amount  $\alpha_t$  at each iteration, regardless of the values of  $a_i$ !

## Finding the Limit

Therefore:  $\boldsymbol{\theta}_t = \left( \sum_{s=0}^{t-1} \alpha_s \right) [1, 1, 1]^T$

At convergence, we need  $\mathbf{a}^T \boldsymbol{\theta}_\infty = 1$ :  $[1, 0.1, 0.01] \cdot [1, 1, 1]^T = 1$   
 $c(1.11) = 1 \implies c = \frac{1}{1.11} = \frac{100}{111}$

Therefore:  $\boxed{\boldsymbol{\theta}_\infty} = \frac{1}{1.11} [1, 1, 1]^T = \left[ \frac{100}{111}, \frac{100}{111}, \frac{100}{111} \right] \approx [0.9009, 0.9009, 0.9009]^T$

## Key Insight

**Adam converges to the minimum  $L_\infty$  norm solution!** The adaptive scaling completely cancels out the different magnitudes in  $\mathbf{a}$ , making all coordinates equal. This is fundamentally different from vanilla SGD, which converged to the minimum  $L_2$  norm solution  $\boldsymbol{\theta} \propto \mathbf{a}$ .

10/13/2025, 1:30:51 PM

- ⊙ Consider a learning approach that first did training input feature rescaling (so that each feature had unit second-moment), then ran SGD to convergence, and then converted the solution for the rescaled problem back to the original units. What specific vector  $\theta$  would it give as its final solution (for use in original coordinates)?

10/13/2025, 1:44:35 PM

Successfully one-shot the problem with more detailed step-by-step solutions.

*Thought: Decoded feature scaling's impact on SGD solution vector.*

Excellent question! Let me work through this step by step.

## Step 1: Feature Rescaling

We have one training point with features  $\mathbf{a} = [1, 0.1, 0.01]^T$ .

To rescale each feature to have unit second-moment (since we have one data point, the second moment of feature  $i$  is  $a_i^2$ ), we divide each feature by its absolute value:

$$\tilde{a}_i = \frac{a_i}{|a_i|} = 1 \text{ for all } i$$

So  $\tilde{\mathbf{a}} = [1, 1, 1]^T$ .

For the parameters, to maintain equivalence  $\mathbf{a}^T \boldsymbol{\theta} = \tilde{\mathbf{a}}^T \tilde{\boldsymbol{\theta}}$ , we need:

$$\tilde{\theta}_i = |a_i| \theta_i$$

$$\text{or inversely: } \theta_i = \frac{1}{|a_i|} \tilde{\theta}_i$$

## Step 2: Run SGD on Rescaled Problem

In the rescaled space, we have  $\tilde{\mathbf{a}} = [1, 1, 1]^T$  and need to satisfy  $\tilde{\mathbf{a}}^T \tilde{\boldsymbol{\theta}} = 1$ .

From our previous analysis, SGD from  $\tilde{\boldsymbol{\theta}}_0 = \mathbf{0}$  converges to:

$$\begin{aligned} \tilde{\boldsymbol{\theta}}_{\infty} &= \frac{\tilde{\mathbf{a}}}{\|\tilde{\mathbf{a}}\|^2} \\ \|\tilde{\mathbf{a}}\|^2 &= \frac{[1, 1, 1]}{3} = \left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right] \end{aligned}$$

## Step 3: Convert Back to Original Coordinates

Using  $\theta_i = \frac{1}{|a_i|} \tilde{\theta}_i$ :

$$\begin{aligned} \theta_1 &= \frac{1}{1} \cdot \frac{1}{3} = \frac{1}{3} \\ \theta_2 &= \frac{1}{0.1} \cdot \frac{1}{3} = \frac{10}{3} \\ \theta_3 &= \frac{1}{0.01} \cdot \frac{1}{3} = \frac{100}{3} \end{aligned}$$

Therefore:  $\boxed{\boldsymbol{\theta}_{\infty} = \left[\frac{1}{3}, \frac{10}{3}, \frac{100}{3}\right] \approx [0.333, 3.333, 33.333]^T}$

We can verify:  $1 \cdot \frac{1}{3} + 0.1 \cdot \frac{10}{3} + 0.01 \cdot \frac{100}{3} = \frac{1+1+1}{3} = 1 \checkmark$

## Key Insight

This solution has  $\theta_i \propto \frac{1}{a_i}$ , giving much larger weights to features with smaller coefficients! This minimizes a **weighted  $L_2$  norm** where weights are inversely proportional to feature magnitudes. Feature rescaling makes SGD treat all features "democratically" in the rescaled space, but when converted back, features that were originally smaller receive proportionally larger parameter values.

10/13/2025, 1:45:20 PM





## 5. Analyzing Distributed Training

For real-world models trained on lots of data, the training of neural networks is parallelized and accelerated by running workers on distributed resources, such as clusters of GPUs. In this question, we will explore three popular distributed training paradigms:

**All-to-All Communication:** Each worker maintains a copy of the model parameters (weights) and processes a subset of the training data. After each iteration, each worker communicates with every other worker and updates its local weights by averaging the gradients from all workers.

**Parameter Server:** A dedicated server, called the parameter server, stores the global model parameters. The workers compute gradients for a subset of the training data and send these gradients to the parameter server. The server then updates the global model parameters and sends the updated weights back to the workers.

**Ring All-Reduce:** Arranges  $n$  workers in a logical ring and updates the model parameters by passing messages in a circular fashion. Each worker computes gradients for a subset of the training data, splits the gradients into  $n$  equally sized chunks and sends a chunk of the gradients to their neighbors in the ring. Each worker receives the gradient chunks from its neighbors, updates its local parameters, and passes the updated gradient chunks along the ring. After  $n - 1$  passes, all gradient chunks have been aggregated across workers, and the aggregated chunks are passed along to all workers in the next  $n - 1$  steps. This is illustrated in Fig. 1.

**For each of the distributed training paradigms, fill in the total number of messages sent and the size of each message.** Assume that there are  $n$  workers and the model has  $p$  parameters, with  $p$  divisible by  $n$ .

	Number of Messages Sent	Size of each message
All-to-All		$p$
Parameter Server	$2n$	
Ring All-Reduce	$n(2(n - 1))$	

Please fill in the table attached according to the problem set up

10/13/2025, 1:54:23 PM

One-shot the problem perfectly.

*Thought: Analyzed distributed training paradigms and communication strategies.*

Looking at each distributed training paradigm, I'll analyze the communication patterns:



## All-to-All Communication

- Each worker sends its gradients to all other  $(n-1)$  workers
- Total messages from all  $n$  workers:  **$n(n-1)$**
- Each message contains the full gradient vector:  **$p$**

## Parameter Server

- Each worker sends gradients to the parameter server:  $n$  messages
- Parameter server sends updated weights back to each worker:  $n$  messages
- Total messages:  **$2n$**
- Each message contains full gradients/weights:  **$p$**

## Ring All-Reduce

- The algorithm has two phases, each with  $(n-1)$  passes:
  - **Reduce-Scatter phase:**  $(n-1)$  passes where each worker sends gradient chunks
  - **All-Gather phase:**  $(n-1)$  passes where aggregated chunks are distributed
- In each pass, all  $n$  workers send one message
- Total messages:  $n \times 2(n-1) = \mathbf{2n(n-1)}$  [which matches the given format  **$n(2(n-1))$** ]
- Each message is one chunk:  **$p/n$**

## Completed Table:

	Number of Messages Sent	Size of each message
All-to-All	<b><math>n(n-1)</math></b>	$p$
Parameter Server	$2n$	<b><math>p</math></b>

	Number of Messages Sent	Size of each message
Ring All-Reduce	$n(2(n-1))$	$p/n$

10/13/2025, 1:54:47 PM