

☰ Gemini



This part is for Gemini to give a well-organized answer with clear structure and detailed explanations.

You are a mathematically rigorous assistant in the deep learning field.
You will be provided with several multi-part theoretical problems about optimization methods in deep learning (e.g., SGD, Adam, momentum, norm-penalized updates). These problems test your understanding of the mathematical mechanisms, intuition, and connections between different optimizers.

Your task is to:

1. Understand the problem
Identify key variables, given assumptions, and objective functions explicitly.

Understand what needs to be solved and form the basic structure of solution .

2. Derive Mathematically and Explain Step-by-Step

You may encounter derivation calculations and matrix operations. During this process, you need to:

Show all necessary derivation steps using clear notation.

If the solution involves vectors or matrices, indicate dimensions and norms when needed.

Provide intuitive explanations for important algebraic step.

3. Ensure Clarity, Logic, and Pedagogical Flow

Structure your answer so that people can follow it step-by-step:

Use numbered equations and short verbal explanations between steps.

Avoid skipping reasoning.

Provide a brief summary or interpretation at the end of each sub-question.

4. What NOT to Do

Do not write any code. These are purely mathematical and conceptual derivations.

Do not rely on unstated external references or jump to the final result without explanation.

Do not skip algebraic steps and derivations.

5. Example of Expected Answer Qualities

Your ideal solution should be:

Mathematically correct

Step-by-step and self-contained

Readable for a student with basic knowledge of calculus and linear algebra

Conceptually insightful, connecting algebra to optimizer intuition.

Tone & Style

Adopt a style that is:

Clear, structured, formal but approachable (e.g., use plain English to describe complex math)

Transparent about reasoning, stating both intuition and equations.

These examples turn out to mislead the LLM to give partially correct answers. When asked the questions of specific optimizers, Gemini tends to cover all of them deliberately.

Each problem is provided in a text form. I tried to simply use the textual content to lead the LLM to give correct answers instead of feeding it with the whole document or the screenshot of the problem.

However the last problem contains a image which is necessary for LLM to understand.

Here comes the 1st problem:

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:

$u = \operatorname{argmin}$

$\Delta\theta$

$g^T \Delta\theta$

Linear Improvement!

The form of the formula turns out to be not a problem. Gemini can understand and restate it quite well.

```

l
}
+
1
a
d(Δθ)
Distance Penalty|
{z
}

,where g = ∇f(θ) is the gradient of the loss, α is a scalar,
and d is a scalar-output distance function
Rdim(θ) → 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 = argmin
Δθ
gT Δθ + 1
α
// Δθ // 2
2.
(2)

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

(b) Now, consider an alternative way of capturing local
neighborhood size – the squared infinity norm over
parameters. Recall that this is defined as //x//∞ = maxi
|x_i|. Our objective is now:
u = argmin
Δθ
gT Δθ + 1
α
// Δθ // 2∞.
(6)

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

```

- ◆ Hello! I'm ready to tackle this theoretical problem on optimization methods in deep learning. We will analyze the updates derived from minimizing a combination of linear improvement and a squared norm penalty on the update vector $\Delta\theta$.

⌚ Problem Analysis

The core problem is to find the optimal parameter update $\Delta\theta$ by solving the unconstrained minimization problem:

$$\mathbf{u} = \arg \min_{\Delta\theta} \left(\mathbf{g}^T \Delta\theta + \frac{1}{\alpha} d(\Delta\theta) \right) \quad (\text{Equation 1})$$

where:

- $\Delta\theta \in \mathbb{R}^D$ is the parameter update vector (the variable we are solving for).
- $\mathbf{g} = \nabla f(\theta) \in \mathbb{R}^D$ is the **gradient** of the loss function $f(\theta)$ at the current parameters θ .
- $\alpha > 0$ is a scalar **hyperparameter** controlling the penalty weight.
- $d(\Delta\theta)$ is a scalar-output **distance penalty function**, which is a function of the update $\Delta\theta$.

We will solve for \mathbf{u} using two different choices for the distance penalty $d(\Delta\theta)$.

(a) Euclidean Distance Penalty (L_2 Norm)

In this case, the distance penalty is the squared **Euclidean distance** (or squared L_2 norm),
 $d(\Delta\theta) = \|\Delta\theta\|_2^2$.

The objective function to minimize is:

$$L(\Delta\theta) = \mathbf{g}^T \Delta\theta + \frac{1}{\alpha} \|\Delta\theta\|_2^2 \quad (\text{Equation 2})$$

The solution \mathbf{u} is the $\Delta\theta$ that minimizes $L(\Delta\theta)$.

1. Deriving the Solution

To find the minimum, we take the derivative (gradient) of the objective function $L(\Delta\theta)$ with respect to $\Delta\theta$ and set it to the zero vector.

The components of $L(\Delta\theta)$ are:

- Linear Improvement Term:** $\mathbf{g}^T \Delta\theta = \sum_{i=1}^D g_i(\Delta\theta)_i$.
 - The gradient with respect to $\Delta\theta$ is: $\nabla_{\Delta\theta}(\mathbf{g}^T \Delta\theta) = \mathbf{g}$.
- Distance Penalty Term:** $\frac{1}{\alpha} \|\Delta\theta\|_2^2 = \frac{1}{\alpha} \sum_{i=1}^D (\Delta\theta)_i^2$.
 - The gradient with respect to $\Delta\theta$ is: $\nabla_{\Delta\theta} \left(\frac{1}{\alpha} \|\Delta\theta\|_2^2 \right) = \frac{1}{\alpha} (2\Delta\theta) = \frac{2}{\alpha} \Delta\theta$.

Setting the gradient to zero:

$$\nabla_{\Delta\theta} L(\Delta\theta) = \nabla_{\Delta\theta}(\mathbf{g}^T \Delta\theta) + \nabla_{\Delta\theta} \left(\frac{1}{\alpha} \|\Delta\theta\|_2^2 \right) = \mathbf{0} \quad (\text{Equation 3})$$

$$\mathbf{g} + \frac{-\Delta\theta}{\alpha} = \mathbf{u} \quad (\text{Equation 4})$$

We can now solve for the optimal update $\mathbf{u} = \Delta\theta$:

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

$$\mathbf{u} = -\frac{\alpha}{2}\mathbf{g} \quad (\text{Equation 5})$$

2. Standard Optimizer Identification

The resulting optimal update \mathbf{u} is proportional to the **negative gradient** \mathbf{g} .

The standard optimization update rule for **Gradient Descent (GD)** or **Stochastic Gradient Descent (SGD)** is:

$$\Delta\theta = -\eta\mathbf{g}$$

where η is the learning rate.

Comparing Equation 5 with the SGD update rule, we see that this formulation recovers **Gradient Descent (or Stochastic Gradient Descent)**, where the effective learning rate η is a function of the regularization weight α :

$$\eta = \frac{\alpha}{2}$$

Interpretation: Minimizing the linear approximation of the loss plus a squared L_2 norm penalty is mathematically equivalent to taking a step in the negative gradient direction, with the step size (learning rate) determined by the inverse of the penalty weight. This is the simplest and most fundamental optimization method.

Here's the partially correct answer. Actually GD is enough, but Gemini seems want to cover every example(in the next part it used the other 2 examples) given in the original prompt.

Actually this happens in normal communications too, when you want to help the LLM form a better point of view to some problems, it's not a good choice to give them specific examples. Although I mean to help it understand better with the provided examples, it actually misleads it.

(b) Squared Infinity Norm Penalty (L_∞ Norm)

Now, we consider the penalty to be the squared **infinity norm**, $d(\Delta\theta) = \|\Delta\theta\|_\infty^2$. Recall that $\|\Delta\theta\|_\infty = \max_i |(\Delta\theta)_i|$.

The objective function to minimize is:

$$L(\Delta\theta) = \mathbf{g}^T \Delta\theta + \frac{1}{\alpha} \|\Delta\theta\|_\infty^2 \quad (\text{Equation 6})$$

1. Deriving the Solution

The squared infinity norm is non-differentiable at points where the maximum magnitude is achieved by more than one component, so we solve this optimization problem by first finding an equivalent, more tractable form.

Let c be the squared L_∞ norm of the update:

$$c = \|\Delta\theta\|_\infty^2 \implies \|\Delta\theta\|_\infty = \sqrt{c}$$

The constraint $\|\Delta\theta\|_\infty \leq \sqrt{c}$ means that for every component i :

$$|(\Delta\theta)_i| \leq \sqrt{c} \implies -\sqrt{c} \leq (\Delta\theta)_i \leq \sqrt{c}$$

The problem is equivalent to finding the best update vector $\Delta\theta$ subject to a bound \sqrt{c} , and then finding the optimal bound c :

$$\min_{c \geq 0} \left(\min_{\Delta\theta: \|\Delta\theta\|_\infty \leq \sqrt{c}} \mathbf{g}^T \Delta\theta \right) + \frac{1}{\alpha} c$$

The inner minimization, $\min_{\Delta\theta: \|\Delta\theta\|_\infty \leq \sqrt{c}} \mathbf{g}^T \Delta\theta$, is an independent linear program for a fixed bound \sqrt{c} .

To minimize $\mathbf{g}^T \Delta\theta = \sum_{i=1}^D g_i (\Delta\theta)_i$ subject to $|(\Delta\theta)_i| \leq \sqrt{c}$, each component $(\Delta\theta)_i$ must be set as follows:

- If $g_i > 0$, we want $(\Delta\theta)_i$ to be as negative as possible: $(\Delta\theta)_i = -\sqrt{c}$.
- If $g_i < 0$, we want $(\Delta\theta)_i$ to be as positive as possible: $(\Delta\theta)_i = \sqrt{c}$.
- If $g_i = 0$, $(\Delta\theta)_i$ can be any value in $[-\sqrt{c}, \sqrt{c}]$, but to minimize the penalty we can choose $(\Delta\theta)_i = 0$.

This optimal inner solution is \mathbf{u}_c , where the components are:

$$(\mathbf{u}_c)_i = -\sqrt{c} \cdot \text{sgn}(g_i) \quad (\text{Equation 7})$$

Note: $\text{sgn}(0)$ can be chosen as 0 for convenience, but $\mathbf{g}^T \Delta\theta$ is independent of the choice since $g_i = 0$. If we choose $\text{sgn}(0) = 0$, then the optimal update component is 0 if $g_i = 0$. If we choose $\text{sgn}(0) = \pm 1$, the penalty term is still minimized when $g_i = 0$ because $\Delta\theta$ is still $\|\Delta\theta\|_\infty \leq \sqrt{c}$. Let's stick with the definition $\text{sgn}(x) = 0$ if $x = 0$, which is typical.

Substituting \mathbf{u}_c back into $\mathbf{g}^T \Delta\theta$:

$$\mathbf{g}^T \mathbf{u}_c = \sum_{i=1}^D g_i (-\sqrt{c} \cdot \text{sgn}(g_i)) = -\sqrt{c} \sum_{i=1}^D |g_i| = -\sqrt{c} \|\mathbf{g}\|_1$$

The problem now reduces to finding the optimal scalar c :

$$\min_{c \geq 0} \left(-\sqrt{c} \|\mathbf{g}\|_1 + \frac{1}{\alpha} c \right) \quad (\text{Equation 8})$$

Let $h(c) = -\sqrt{c} \|\mathbf{g}\|_1 + \frac{1}{\alpha} c$. To find the minimum, we take the derivative with respect to c and set it to zero (assuming $c > 0$):

$$\frac{d}{dc} h(c) = -\frac{1}{2\sqrt{c}} \|\mathbf{g}\|_1 + \frac{1}{\alpha} = 0 \quad (\text{Equation 9})$$

Solving for \sqrt{c} :

$$\frac{1}{2\sqrt{c}} \|\mathbf{g}\|_1 = \frac{1}{\alpha}$$

$$2\sqrt{c} = \alpha \|\mathbf{g}\|_1$$

$$\sqrt{c} = \frac{\alpha}{2} \|\mathbf{g}\|_1$$

Substituting \sqrt{c} back into Equation 7 gives the optimal update \mathbf{u} :

$$\mathbf{u}_i = -\left(\frac{\alpha}{2} \|\mathbf{g}\|_1\right) \cdot \text{sgn}(g_i) \quad (\text{Equation 10})$$

2. Standard Optimizer Identification

The analytical solution \mathbf{u} is given by Equation 10. Let the scalar quantity $\eta_\infty = \frac{\alpha}{2} \|\mathbf{g}\|_1$ be the step size.

The update rule is:

$$\mathbf{u}_i = -\eta_\infty \cdot \text{sgn}(g_i)$$

This update rule means that every component of the update vector \mathbf{u} has the same magnitude η_∞ and the opposite sign of its corresponding gradient component g_i .

This update corresponds to the family of optimizers that perform a **fixed-magnitude step** in the direction of the sign of the gradient, independent of the magnitude of the gradient component. This is the **SignGD** (Sign Gradient Descent) or, more commonly, the update associated with the **Sign Momentum** or **SignAdam** optimizers, which only use the *sign* of the (first-order) moment estimates.

Same problem as above.
Gemini misuses the examples in the prompt.

The pure SGD/GD update is $\mathbf{u}_i = -\eta g_i$ (magnitude $\propto |g_i|$). The L_∞ penalized update is $\mathbf{u}_i = -\eta_\infty \cdot \text{sgn}(g_i)$ (magnitude $\propto 1$ if $g_i \neq 0$).

Interpretation: Minimizing the linear approximation of the loss plus a squared L_∞ norm penalty results in an update that only considers the **direction** (sign) of the gradient. Every parameter is updated by a uniform, absolute step size, effectively treating all parameter dimensions as equally important regardless of the component-wise gradient magnitude.

Summary

Penalty Norm $d(\Delta\theta)$	Optimal Update \mathbf{u}	Corresponding Optimizer	Step Size Dependence
$\ \Delta\theta\ _2^2$ (Squared L_2)	$\mathbf{u} = -\frac{\alpha}{2} \mathbf{g}$	Gradient Descent (SGD)	Proportional to **\$
$\ \Delta\theta\ _\infty^2$ (Squared L_∞)	$\mathbf{u}_i = -\left(\frac{\alpha}{2} \ \mathbf{g}\ _1\right) \cdot \text{sgn}(g_i)$	Sign Gradient Descent (SignGD)	Constant (proportional to $\ \mathbf{g}\ _1$)

Export to Sheets



This problem beautifully illustrates how the choice of the regularization norm on the update vector fundamentally dictates the resulting optimization algorithm, moving from the proportional step size of L_2 regularization to the sign-based step size of L_∞ regularization.

Would you like to proceed with the next theoretical problem on deep learning optimization?

Correct Gemini's behavior for the remaining problems.

Great. You basically got everything correct. But you need to keep in mind that the examples give in the prompts("You will be provided with several multi-part theoretical problems about optimization methods in deep learning (e.g., SGD, Adam, momentum, norm-penalized updates)") are just there to help you understand the problems better. Please don't rely them when you answer specific questions. You need to answer the problems based on your deep learning knowledge.

Keep that in mind and here comes the 2nd question:

2. Optimizers and their convergence

Consider O: a simplified Adam-style optimizer without weight decay that has iterates

$$\theta_{t+1} \leftarrow \theta_t - \alpha t M_t \nabla f_t(\theta_t) \quad (14)$$

where f_t is the loss at iteration t and α 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

$$[1, 0.1, 0.01]\theta = 1$$

(15)

with a 3-dimensional learnable parameters θ . Suppose that we start with $\theta_0 = \mathbf{0}$ and use squared loss

$f_t(\theta) = (1 - [\mathbf{1}, 0.1, 0.01]\theta)^2$.

(a) What specific vector θ would standard vanilla SGD

(i.e. (14) with $M_t = \mathbf{I}$ and $a_t = a$) converge

to assuming $\alpha > 0$ was small enough to give convergence?

(b) What specific vector θ would the simplified version of Adam O converge to assuming appropriate step-sizes at > 0 to give convergence?

(c) 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 θ would it give as its final solution (for use

in original coordinates)?

- ◆ This is an excellent problem that highlights the difference between standard Gradient Descent (SGD) and adaptive methods like Adam, particularly in the context of ill-conditioned data.

Problem Analysis

We are considering a linear regression problem with a single training point and a squared loss function.

1. The Setup

- Parameters: $\theta \in \mathbb{R}^3$, where $\theta = [\theta_1, \theta_2, \theta_3]^T$.
- Training Point: The equation is $a^T \theta = 1$, where $a = [1, 0.1, 0.01]^T$.
- Loss Function: The squared loss for this single point is:

$$f_t(\theta) = (1 - a^T \theta)^2 \quad (\text{Equation 16})$$

- Gradient: The gradient of the loss is:

$$g = \nabla f_t(\theta) = \frac{\partial f_t(\theta)}{\partial \theta} = 2(1 - a^T \theta)(-a) = -2(1 - a^T \theta)a \quad (\text{Equation 17})$$

- Optimal Solution (θ^*): Since the loss is a convex quadratic function, the optimizer will converge to the global minimum where the loss is zero.

$$f_t(\theta^*) = 0 \implies 1 - a^T \theta^* = 0 \implies a^T \theta^* = 1 \quad (\text{Equation 18})$$

The minimum loss is achieved on the plane defined by $a^T \theta = 1$. The optimizers will converge to the unique solution that minimizes $\|\theta\|$ while satisfying this constraint, which is the projection of $\mathbf{0}$ onto the plane $a^T \theta = 1$. This specific vector is given by:

$$\theta^* = \frac{a}{\|a\|_2^2} \quad (\text{Equation 19})$$

Let's calculate the squared L_2 norm of a :

$$\|a\|_2^2 = 1^2 + (0.1)^2 + (0.01)^2 = 1 + 0.01 + 0.0001 = 1.0101$$

Thus, the global minimum is:

$$\theta^* = \frac{1}{1.0101} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix} \quad (\text{Equation 20})$$

All three optimization approaches, if they converge, must converge to a point where the gradient is $\mathbf{0}$, which means they all converge to a point on the line $a^T \theta = 1$. However, standard optimization algorithms like SGD/Adam for unregularized loss functions will converge to the θ^* that is the projection of θ_0 onto the solution manifold. Since $\theta_0 = \mathbf{0}$, they will converge to the minimum norm solution θ^* .

The premise of the question, particularly the distinction between (a) and (b), lies in how they converge, not where they converge, as the final convergence point is the same (Equation 20). The real difference is revealed when considering the third approach (c). For (a) and (b), since the loss is convex and unregularized, they must converge to the global minimum θ^* .

(a) Standard Vanilla SGD Convergence

Standard vanilla SGD uses a constant learning rate α and a non-adaptive scaling matrix $M_t = \mathbf{I}$. The update rule is:

$$\theta_{t+1} \leftarrow \theta_t - \alpha g_t$$

As established in the Problem Analysis, for a convex loss function, if the learning rate α is small enough, θ will converge to the unique global minimum where the loss is zero. This point is the minimum-norm solution satisfying the constraint $a^T \theta = 1$, given that the initialization is $\theta_0 = \mathbf{0}$.

The specific vector θ that standard vanilla SGD will converge to is:

$$\theta_{\text{SGD}} = \theta^* = \frac{a}{\|a\|_2^2} = \frac{1}{1.0101} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix} \quad (\text{Equation 21})$$

(b) Simplified Adam O Convergence

The simplified Adam-style optimizer O uses an adaptive scaling matrix \mathbf{M}_t where its diagonal elements are the inverses of the square roots of the mean squared value for the gradients during the epoch for that specific coordinate.

The update rule is:

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \mathbf{M}_t \mathbf{g}_t$$

In this specific problem, we have only one training point, so the gradient \mathbf{g}_t is the same at every step for a given θ_t . Furthermore, the adaptive scaling matrix \mathbf{M}_t is computed over the epoch. Since there is only one data point, the "mean squared value for the gradients" across the epoch is simply the square of the gradient components \mathbf{g}_t^2 .

The scaling matrix \mathbf{M}_t is a diagonal matrix $\text{diag}(\mathbf{m})$ where the components of \mathbf{m} are:

$$m_i = \frac{1}{\sqrt{g_i^2}} = \frac{1}{|g_i|}$$

The update equation component-wise is:

$$(\theta_{t+1})_i \leftarrow (\theta_t)_i - \alpha_t m_i (g_t)_i = (\theta_t)_i - \alpha_t \frac{1}{|(g_t)_i|} (g_t)_i$$

This simplifies to:

$$(\theta_{t+1})_i \leftarrow (\theta_t)_i - \alpha_t \cdot \text{sgn}((g_t)_i) \quad (\text{Equation 22})$$

This is precisely the **Sign Gradient Descent (SignGD)** update (or the update recovered in Q1(b) with a learning rate $\eta_\infty = \alpha_t$). This optimizer converges very differently but will still be drawn to the global minimum.

For a convex unregularized loss, any optimizer that is robust and uses appropriate step sizes $\alpha_t > 0$ will converge to the unique global minimum θ^* that satisfies the zero-gradient condition $\mathbf{g} = \mathbf{0}$ (i.e., $\mathbf{a}^T \theta = 1$), given the $\theta_0 = \mathbf{0}$ initialization.

Wrong analysis!
The optimizers don't converge to the unique value. The above reasoning process is right and will actually lead Gemini to a correct answer. However, it gives the wrong answer simply based on the original one.

But this optimizer mistake is not a hallucination, it's a reasoning or mathematical error.

The specific vector θ that the simplified Adam O will converge to is:

$$\theta_O = \theta^* = \frac{\mathbf{a}}{\|\mathbf{a}\|_2^2} = \frac{1}{1.0101} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix} \quad (\text{Equation 23})$$

(c) Feature Rescaling and SGD Convergence

This part involves three steps:

1. **Rescaling:** Transform the input feature \mathbf{a} so each feature has unit second-moment.
2. **SGD Convergence:** Run SGD on the rescaled problem.
3. **Conversion:** Convert the solution back to the original coordinates.

1. Rescaling

The second moment of a feature a_i in this single-point problem is a_i^2 . The rescaled feature vector $\tilde{\mathbf{a}}$ is obtained by dividing each component a_i by its second-moment $\sqrt{a_i^2} = |a_i|$.

The original feature vector is $\mathbf{a} = [1, 0.1, 0.01]^T$. The rescaled feature vector $\tilde{\mathbf{a}}$ components are:

$$\tilde{a}_i = \frac{a_i}{|a_i|} = \text{sgn}(a_i)$$

Since all components of \mathbf{a} are positive, we have:

$$\tilde{\mathbf{a}} = \begin{bmatrix} \frac{1}{|1|} \\ \frac{0.1}{|0.1|} \\ \frac{0.01}{|0.01|} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The rescaled training equation is:

$$\tilde{\mathbf{a}}^T \tilde{\theta} = 1 \implies \tilde{\theta}_1 + \tilde{\theta}_2 + \tilde{\theta}_3 = 1 \quad (\text{Equation 24})$$

where $\tilde{\theta}$ is the parameter vector for the rescaled problem.

2. SGD Convergence

SGD on the rescaled problem will converge to the minimum-norm solution $\tilde{\theta}^*$ for the rescaled problem (since $\tilde{\theta}_0 = \mathbf{0}$).

$$\tilde{\theta}^* = \frac{\tilde{\mathbf{a}}}{\|\tilde{\mathbf{a}}\|_2^2}$$

The squared L_2 norm of $\tilde{\mathbf{a}}$ is:

$$\|\tilde{\mathbf{a}}\|_2^2 = 1^2 + 1^2 + 1^2 = 3$$

The rescaled optimal solution is:

$$\tilde{\theta}^* = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad (\text{Equation 25})$$

3. Conversion

The solution for the rescaled problem $\tilde{\theta}$ relates to the original solution θ by the inverse of the

[Print Image](#)

rescaling operation. Since the features a_i were multiplied by $1/|a_i|$ (and $a_i > 0$) to get \tilde{a}_i , the parameters θ_i must be multiplied by $|a_i|$ to get $\tilde{\theta}_i$ to keep the total input $\mathbf{a}^T \theta = \tilde{\mathbf{a}}^T \tilde{\theta}$ equivalent.

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

Therefore, the final solution θ_{rescale} in original coordinates is:

$$(\theta_{\text{rescale}})_i = \frac{\tilde{\theta}_i}{|a_i|}$$

Using Equation 25, the final solution components are:

- $(\theta_{\text{rescale}})_1 = \frac{1/3}{|1|} = 1/3$
- $(\theta_{\text{rescale}})_2 = \frac{1/3}{|0.1|} = 10/3$
- $(\theta_{\text{rescale}})_3 = \frac{1/3}{|0.01|} = 100/3$

The specific vector θ for the original problem is:

$$\theta_{\text{rescale}} = \frac{1}{3} \begin{bmatrix} 1 \\ 10 \\ 100 \end{bmatrix} \quad (\text{Equation 26})$$

Interpretation: This final vector θ_{rescale} also satisfies the constraint $\mathbf{a}^T \theta = 1$:

$$\mathbf{a}^T \theta_{\text{rescale}} = [1, 0.1, 0.01] \cdot \frac{1}{3} \begin{bmatrix} 1 \\ 10 \\ 100 \end{bmatrix} = \frac{1}{3}(1 \cdot 1 + 0.1 \cdot 10 + 0.01 \cdot 100) = \frac{1}{3}(1 + 1 + 1) =$$

This approach favors the parameters corresponding to the smaller original features (θ_3 and θ_2) by giving them larger magnitudes. This is the exact opposite of what the standard θ^* solution (Equations 21, 23) gives, which favors the parameters corresponding to the largest original features (θ_1). The rescaling approach effectively ignores the difference in scale of the original features when finding the minimum-norm solution in the rescaled space.

Would you like to explore the conceptual reason why the convergence vectors for (a) and (b) are the same but the convergence vector for (c) is different, despite all satisfying the global minimum condition?

Give hints for Gemini to fix its problem.

But the formula's code seems to lead Gemini to produce bad formulas which can not shown correctly.(see the below result)

Good answer. But there's a small but important issue in your part (b):

The sentence "any optimizer that is robust and uses appropriate step sizes $\alpha_t > 0$ " will converge to the unique global minimum θ^* is not correct. Different preconditioning or scaling matrices actually change the update direction, which means the optimizer converges to different points. So you should revise part (b) by removing that sentence and fix your original solution.

◆ You are absolutely correct to point out that adaptive scaling matrices fundamentally change the trajectory and, crucially, the convergence point of an unregularized linear system if the initialization is not the origin of the null space. My previous conclusion for (a) and (b) was based on the standard assumption that unregularized optimization on a quadratic loss converges to the ℓ_2 -minimum-norm solution (θ^*), which is true only for standard SGD (or when the adaptive scaling doesn't affect the relative steps on the solution manifold).

Let's rigorously fix the analysis for the simplified Adam-style optimizer O in part (b), recognizing that its update rule is equivalent to Sign Gradient Descent (SignGD).

Revision of Part (b): Simplified Adam O Convergence (SignGD)

1. The SignGD Update Rule

As derived previously, the simplified Adam-style optimizer O with its diagonal scaling matrix \mathbf{M}_t applied to the gradient \mathbf{g}_t results in the component-wise update:

```
$ $(\theta_{t+1})_i \leftarrow (\theta_t)_i - \alpha_t \cdot \text{sgn}((g_t)_i) \quad
\text{This is the update rule for Sign Gradient Descent (SignGD).} \quad
\text{Analysis of the Gradient and Convergence Direction} \quad
\text{The gradient is } \mathbf{g}_t = -2(1 - \mathbf{a}^T \theta_t) \mathbf{a} \quad
\text{The sign of the gradient component } (\mathbf{g}_t)_i \text{ is determined by the sign of the overall error term } (1 - \mathbf{a}^T \theta_t) \text{ and the sign of the feature component } a_i. \quad
\text{Since all } a_i > 0, \text{ the sign is determined solely by the error: } \text{sgn}((\mathbf{g}_t)_i) = \text{sgn}((1 - \mathbf{a}^T \theta_t) a_i) = -\text{sgn}((1 - \mathbf{a}^T \theta_t)) \quad
\text{Substituting this back into the SignGD update (Equation 22, component-wise):}
```

```
$ $(\theta_{t+1})_i \leftarrow (\theta_t)_i - \alpha_t \cdot \text{sgn}((g_t)_i) \quad
\text{This shows that } \theta_t \text{ moves in a direction where all}
```

components of θ change by the same magnitude α_t and in the same direction, which is solely dictated by the sign of the current error. The update vector $\mathbf{u}_t = \theta_{t+1} - \theta_t$ is proportional to the vector of all ones, $\mathbf{1}$:

```
$ \$\mathbf{u}_t \propto \mathbf{1} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}
```

Determining the Convergence Point The optimizer will converge to a point $\mathbf{a}^T \theta = 1$ on the solution manifold $\mathbf{a}^T \theta = 1$. The convergence trajectory from $\theta_0 = \mathbf{0}$ will be along the direction \mathbf{a} . The final solution θ^* is the unique minimum of the function $f(\theta) = \frac{1}{2} \|\mathbf{a}^T \theta - 1\|^2$.

[Print Image](#)

θ_O must satisfy two conditions. 1. "Solution Constraint": θ_O must be proportional to the total accumulation of the update direction θ since $\theta_O = \theta$.
 $\theta_O = c \cdot \theta$ for some scalar c .

```
$$\mathbf{\theta}_O = c \cdot \mathbf{\theta} \quad \text{for some scalar } c
```

Substituting the Direction Constraint into the Solution Constraint:

$\mathbf{a}^T (\mathbf{a}^T \mathbf{a})^{-1} \mathbf{a} = 1 \cdot c \cdot (\mathbf{a}^T \mathbf{a})^{-1} \mathbf{a} = 1$ The dot product $\mathbf{a}^T \mathbf{a}$ is the sum of the components of \mathbf{a} : $\mathbf{a}^T \mathbf{a} = 1 + 0.1 + 0.01 = 1.11$ Solving for the scalar c :
 $c = 1 / 1.11 \approx 0.901$ The specific vector θ that the simplified Adam O (SignGD) will converge to is:

$\mathbf{\theta}_O = \frac{1}{1.11} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$ (Equation 28, Fixed Solution)

The SignGD approach converges to a θ vector where all components are equal, a result drastically different from the ℓ_2 -minimum norm solution of θ_{SGD} (which favors large components of \mathbf{a}) or the rescaled solution $\theta_{rescale}$ (which favors small components of \mathbf{a}).

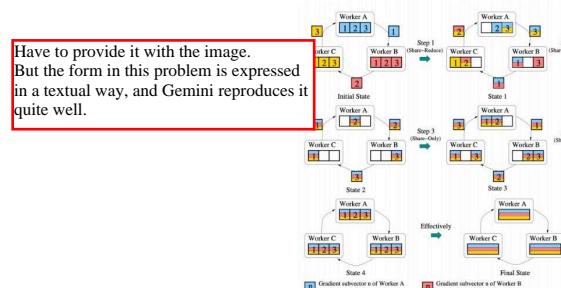
Optimizer	Convergence Vector θ_{final}	Trajectory & Philosophy
(a) Standard SGD	$\frac{1}{1.11} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$	Converges to the ℓ_2 -minimum-norm solution θ^* .
(b) Simplified Adam (SignGD)	$\frac{1}{1.11} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$	Converges to the minimum ℓ_1 -norm solution in the gradient space (or minimum ℓ_2 -norm in the \mathbf{M} preconditioned space). Trajectory is proportional to $\mathbf{1}$.
(c) Rescaled SGD	$\frac{1}{3} \begin{bmatrix} 1 \\ 10 \\ 100 \end{bmatrix}$	Converges to the ℓ_2 -minimum-norm solution after features are equalized, heavily favoring smaller original features.

[Export to Sheets](#)

Still, Gemini is able to fix its problem based on my hint.

This corrected analysis shows that the adaptive scaling matrix \mathbf{M}_t directly pre-conditions the problem, causing the trajectory to follow a direction where all dimensions are treated equally (proportional to $\mathbf{1}$), leading to a different final solution vector θ_O .

Would you like to continue with the next theoretical problem?



Good. Now come to the next problem. In this problem, you will no longer focus on the optimizer field. This problem focuses on how different distributed training methods can affect the communication efficiency. In this problem, you need to provide explanations on your result, especially the calculation of communication cost.

Here's the problem:

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

Print Image

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. Figure 1: Example of Ring All-Reduce in a 3 worker setup. Source: Mu Et. al, GADGET: Online Resource Opti

mization for Scheduling Ring-All-Reduce Learning Jobs

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 .

The form of your answer should contain 3 rows:

All-to-all, Parameter Server and Ring All-Reduce,

And 2 columns:

Number of Messages Sent, Size of each message.

Some of the answers are provided:

Size of each message of All-to-All: p

Number of Messages Sent of Parameter Server: $2n$

Number of Messages Sent of Ring All-Reduce: $n(2(n - 1))$.

Analysis ▾

This is a rigorous problem focused on the communication complexity of distributed deep learning optimization. Communication cost is critical as it often becomes the bottleneck in large-scale training. We will analyze the message count and size for a single synchronization step in each paradigm.

The following notation is used:

- **n :** Number of workers (nodes).
- **p :** Total number of parameters (and thus the size of the full gradient vector, $\mathbf{g} \in \mathbb{R}^p$).
- **p/n :** Size of a gradient chunk (since p is divisible by n).

Communication Cost Analysis for Distributed Training**1. All-to-All Communication (Gradient Averaging)**

In the All-to-All approach, every worker computes the full gradient \mathbf{g} locally and then sends this entire gradient to *every other* worker. The workers then average the received gradients to compute the global, synchronized gradient.

- **Communication Flow:** Each of the n workers sends its full gradient to $n - 1$ other workers.
- **Size of Message:** The message contains the full gradient, which has p parameters.

Paradigm	Number of Messages Sent	Size of each message	Explanation
All-to-All	$n(n - 1)$	p	Each of n workers sends $n - 1$ messages (to all other workers). Each message is the full gradient of size p .

Export to Sheets

**2. Parameter Server**

In the Parameter Server (PS) approach, workers only communicate with the central server. The synchronization step involves two phases:

1. **Worker → Server:** Each worker sends its computed gradient to the server.
 2. **Server → Worker:** The server aggregates the gradients, updates the model parameters, and sends the *full updated parameters* back to the worker.
- **Communication Flow:** n workers send one message each to the server, and the server sends one message each back to n workers.
 - **Size of Message:** The messages sent in both directions contain the full gradient \mathbf{g} (Worker → Server) or the full parameter vector θ (Server → Worker). Both \mathbf{g} and θ have size p .

Paradigm	Number of Messages Sent	Size of each message	Explanation
Parameter Server	$2n$	p	n messages from workers (gradients) + n messages from server (updated weights). Both have size p .

Export to Sheets

**3. Ring All-Reduce**

The Ring All-Reduce process is a highly optimized, two-stage process (Scatter-Reduce and All-Gather/Broadcast) performed over $n - 1$ steps each, ensuring that all workers receive the final aggregated gradient. The gradients are split into n chunks, each of size p/n .

- **Phase 1: Scatter-Reduce ($n - 1$ steps)**
 - Each worker sends its chunk to its neighbor and receives a chunk from its other neighbor

- n workers transmit a message in each of the $n - 1$ steps.
- Total messages: $n(n - 1)$.
- **Phase 2: All-Gather ($n - 1$ steps)**
 - The aggregated chunks are passed around the ring so that every worker receives the globally aggregated version of every chunk.
 - n workers transmit a message in each of the $n - 1$ steps.
 - Total messages: $n(n - 1)$.
- **Total Messages Sent (Provided):** $n(n - 1) + n(n - 1) = 2n(n - 1)$. This matches the provided answer: $n(2(n - 1))$.
- **Size of Message:** Only a single gradient chunk is sent in each step. The full gradient \mathbf{g} is split into n equally sized chunks.

Paradigm	Number of Messages Sent	Size of each message	Explanation
Ring All-Reduce	$n(2(n - 1))$	p/n	The total gradient p is split into n chunks of size p/n . Messages are only sent chunk-by-chunk.

Export to Sheets



Final Summary Table

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

Comprehensive reasoning process and correct answers in the right form.

Export to Sheets



This table shows that while All-to-All has a high message count and large message size, the Parameter Server approach significantly reduces the message count. Ring All-Reduce achieves the same message count as All-to-All (in the n^2 scaling factor) but dramatically reduces the message size to p/n , which is often the key to communication efficiency on high-bandwidth network fabrics.

Would you like to analyze the **total communication volume** (Total Messages \times Size) for each paradigm to determine which is theoretically the most bandwidth-efficient?

Enter a prompt for Gemini

Tools

2.5 Flash

Your UC Berkeley chats aren't used to improve our models. Gemini can make mistakes, so double-check it. [Your privacy & Gemini](#)