

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

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

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

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

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

$$f(\theta) = (1 - [1, 0.1, 0.01]\theta)^2$$

(a) What specific vector  $\theta$  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?

I paste in the question context and part (a) directly from the pdf of the homework. I make no adjustments to weird formatting and potentially ambiguous notation. I also don't give any context, such as "this is a homework for CS 182 Deep Learning class at Berkeley", and I don't write including any interesting prompting diction that papers like Large Language Models as Optimizers (<https://arxiv.org/abs/2309.03409>) mention, for example "take a deep breath".

Thought for 77 seconds

- We are given a single training point with the equation:  $[1, 0.1, 0.01]\theta = 1$ .

The loss function is squared loss:  $f(\theta) = (1 - [1, 0.1, 0.01]\theta)^2$ .

We are to find the vector  $\theta$  that standard vanilla SGD (with  $M_t = I$  and constant  $\alpha > 0$  small enough) would converge to.

**Note: Since there is only one training point, we are effectively solving a linear system with one equation and three unknowns.**

The gradient of  $f(\theta)$  is:

$$\nabla f(\theta) = -2 * (1 - [1, 0.1, 0.01]\theta) * [1, 0.1, 0.01]^T$$

In vanilla SGD, the update rule is:

$$\theta_{t+1} = \theta_t - \alpha * \nabla f(\theta_t)$$

We start at  $\theta_0 = 0$ .

Since the problem is convex and the step size is small enough, SGD will converge to the minimum of the loss function.

The loss function is:

$$f(\theta) = (1 - x^T \theta)^2, \text{ where } x = [1, 0.1, 0.01]^T.$$

The gradient is zero when:

$$-2 * (1 - x^T \theta) * x = 0.$$

Since  $x$  is not the zero vector, we must have:

$$1 - x^T \theta = 0 \Rightarrow x^T \theta = 1.$$

This is a single equation in three variables. There are infinitely many solutions.

However, note that we are starting at  $\theta_0 = 0$  and using a constant step size. **The SGD update is:**

$$\theta_{t+1} = \theta_t + 2\alpha (1 - x^T \theta_t) x$$

**This is a linear recurrence. We can write it as:**

$$\theta_{t+1} = (I - 2\alpha x x^T) \theta_t + 2\alpha x$$

This is a linear dynamical system. The fixed point (if the step size is small enough so that the eigenvalues of  $(I - 2\alpha x x^T)$  are in the unit circle) will satisfy:

$$\theta = (I - 2\alpha x x^T) \theta + 2\alpha x$$

$$\Rightarrow 2\alpha x x^T \theta = 2\alpha x$$

$$\Rightarrow x x^T \theta = x$$

Since  $x x^T$  is a  $3 \times 3$  matrix and  $x$  is a vector, we note that  $x x^T$  is rank 1. The equation  $x x^T \theta = x$  means that  $\theta$  must be such that the projection of  $\theta$  on  $x$  is 1 (i.e.,  $x^T \theta = 1$ ).

However, note that the fixed point equation is:

$$x (x^T \theta) = x$$

This implies that if  $x$  is not zero, then we must have  $x^T \theta = 1$ . But note that the fixed point equation does not determine the component of  $\theta$  that is orthogonal to  $x$ .

In fact, the SGD update only changes the component of  $\theta$  in the direction of  $x$ . Let me write

$\theta$  as:

$$\theta = \theta_x + \theta_{\text{perp}}$$

where  $\theta_x$  is in the direction of  $x$  and  $\theta_{\text{perp}}$  is perpendicular to  $x$ .

Then the update for  $\theta_{\text{perp}}$  is zero because the gradient is in the direction of  $x$ . Therefore, if we start at  $\theta_0 = 0$ , then  $\theta_{\text{perp}}$  remains 0.

So the entire sequence of  $\theta_t$  remains in the direction of  $x$ . Therefore, we can write  $\theta_t = c_t * x$  for some scalar  $c_t$ .

Then the update rule becomes:

$$c_{t+1} x = c_t x - \alpha * (-2 (1 - x^T (c_t x)) x)$$

$$\Rightarrow c_{t+1} x = c_t x + 2\alpha (1 - c_t x^T x) x$$

Since  $x = [1, 0.1, 0.01]^T$ , we compute  $x^T x = 1^2 + 0.1^2 + 0.01^2 = 1 + 0.01 + 0.0001 = 1.0101$ .

Then:

$$c_{t+1} = c_t + 2\alpha (1 - c_t * 1.0101)$$

This is a linear recurrence in  $c_t$ :

$$c_{t+1} = (1 - 2\alpha * 1.0101) c_t + 2\alpha$$

The fixed point  $c$  satisfies:

$$c = (1 - 2\alpha * 1.0101) c + 2\alpha$$

$$\Rightarrow 2\alpha * 1.0101 c = 2\alpha$$

$$\Rightarrow c = 1 / 1.0101$$

Therefore, the fixed point is:

$$\theta = (1 / 1.0101) * x = [1/1.0101, 0.1/1.0101, 0.01/1.0101]^T$$

Let's compute the numerical values:

$$1/1.0101 \approx 0.9900$$

$$0.1/1.0101 \approx 0.0990$$

$$0.01/1.0101 \approx 0.0099$$

So the vector is approximately  $[0.9900, 0.0990, 0.0099]^T$ .

However, note that the problem says "converge to" and we are using a constant step size.

The condition for convergence is that the step size  $\alpha$  is small enough so that the eigenvalue  $(1 - 2\alpha * 1.0101)$  is in  $(-1, 1)$ . This requires:

$$|1 - 2\alpha * 1.0101| < 1$$

$$\Rightarrow -1 < 1 - 2\alpha * 1.0101 < 1$$

$$\Rightarrow -2 < -2\alpha * 1.0101 < 0$$

$$\Rightarrow 0 < 2\alpha * 1.0101 < 2$$

$$\Rightarrow 0 < \alpha < 1/1.0101 \approx 0.9900$$

So for  $\alpha$  in  $(0, 0.9900)$ , the iteration converges.

Therefore, the vector that vanilla SGD converges to is:

$$\theta = (1 / (1^2 + 0.1^2 + 0.01^2)) * [1, 0.1, 0.01]^T = [1/1.0101, 0.1/1.0101, 0.01/1.0101]^T.$$

But note: This is the minimum norm solution? **Actually, note that we started at 0 and the update only moves in the direction of  $x$ .** So we get the solution with the smallest norm (the one that lies in the span of  $x$  and has no component perpendicular to  $x$ ) that solves  $x^T \theta = 1$ .

In fact, this is the solution given by the pseudoinverse for the underdetermined system. For the equation  $x^T \theta = 1$ , **the minimum norm solution is indeed  $(x^T x)^+ x / (x^T x) = [1, 0.1, 0.01]^T / 1.0101$ .**

So the answer is:

$$\theta = [1/1.0101, 0.1/1.0101, 0.01/1.0101]^T.$$

However, the problem says "What specific vector  $\theta$  would standard vanilla SGD ... converge to". Since we start at 0 and the updates are in the direction of the gradient (which is in the span of  $x$ ), we get the minimum norm solution.

Let me write it exactly:

$$\theta = [1, 0.1, 0.01] / (1^2 + 0.1^2 + 0.01^2) = [1, 0.1, 0.01] / 1.0101.$$

So the answer is:

$$\theta = [1/1.0101, 0.1/1.0101, 0.01/1.0101]^T.$$

But note: the problem says "simplified Adam-style optimizer" but part (a) is about vanilla SGD. So we are ignoring  $M_t$  and just using identity.

Takes some convoluted steps but arrives at correct answer at the end.

The model seldom mentions any "wait but", "oh no", "ah I see", meaning that Deepseek is pretty comfortable and confident with this type of reasoning.

Amid Deepseek's double-checking journey, it makes a valid connection to pure linear algebra, where the solution to this is the (i.e., gradient descent converges to) the min-norm LS solution.

Therefore, the answer for part (a) is the minimum norm solution.

Let me box the answer:

$$\theta = [1/1.0101, 0.1/1.0101, 0.01/1.0101]^T.$$

However, if we are to write it in a box, we can write it as:

$$\begin{bmatrix} \frac{1}{1.0101} & \frac{0.1}{1.0101} & \frac{0.01}{1.0101} \end{bmatrix}$$

But note: the problem might expect an exact expression. Since  $1.0101 = 1 + 0.1^2 + 0.01^2$ , we can write:

$$\theta = \frac{1}{1 + 0.1^2 + 0.01^2} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$$

But let me check:  $0.1^2 = 0.01$ ,  $0.01^2 = 0.0001$ , so  $1 + 0.01 + 0.0001 = 1.0101$ , so yes.

Alternatively, we can write it as:

$$\theta = \frac{1}{1.0101} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$$

But note: the problem says "specific vector". There are infinitely many solutions to the equation, but because we start at 0 and use SGD, we get this one.

So I'll write:

$$\begin{bmatrix} \frac{1}{1.0101} & \frac{0.1}{1.0101} & \frac{0.01}{1.0101} \end{bmatrix}$$

## 🧠 Step 1: Understand the Problem

We have a single training point defined by the equation:

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

The loss function is the squared error:

$$f(\theta) = (1 - [1, 0.1, 0.01]\theta)^2$$

We use **vanilla SGD** with update rule:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$$

where  $\alpha > 0$  is a small constant step size, and we start from  $\theta_0 = 0$ .

As with most cases, after thinking Deepseek presents the solution in a well formatted and logically organized manner.

## 🔍 Step 2: Compute the Gradient

Let  $x = [1, 0.1, 0.01]^T$ . Then the loss becomes:

$$f(\theta) = (1 - x^T \theta)^2$$

The gradient is:

$$\nabla f(\theta) = -2(1 - x^T \theta)x$$

## 🔄 Step 3: Write the SGD Update

The SGD update rule becomes:

$$\theta_{t+1} = \theta_t + 2\alpha(1 - x^T \theta_t)x$$

## 🎯 Step 4: Find the Fixed Point

At convergence,  $\theta_{t+1} = \theta_t = \theta^*$ . Substituting into the update:

$$\theta^* = \theta^* + 2\alpha(1 - x^T \theta^*)x$$

Subtracting  $\theta^*$  from both sides:

$$0 = 2\alpha(1 - x^T \theta^*)x$$

Since  $x \neq 0$ , we must have:

$$1 - x^T \theta^* = 0 \quad \Rightarrow \quad x^T \theta^* = 1$$

## Step 5: Determine the Specific Solution

The equation  $x^\top \theta = 1$  has infinitely many solutions. However, since we start at  $\theta_0 = 0$  and all updates are in the direction of  $x$ , the solution will be the **minimum norm solution**:

$$\theta^* = \frac{x}{\|x\|^2}$$

Compute  $\|x\|^2$ :

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

Thus,

$$\theta^* = \frac{1}{1.0101} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$$

## Final Answer

$$\begin{bmatrix} \frac{1}{1.0101} \\ \frac{0.1}{1.0101} \\ \frac{0.01}{1.0101} \end{bmatrix}$$

(b) What specific vector  $\theta$  would the simplified version of Adam Overconverge to assuming appropriate step-sizes at  $>0$  to give convergence?

Again, I paste in the part directly without addition formatting fixes or any filler words.

Thought for 195 seconds

- We are given: "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  $\alpha_t$  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."

We have exactly  $n=1$  training point:  $[1, 0.1, 0.01]\theta = 1$ . So loss:  $f(\theta) = (1 - [1, 0.1, 0.01]\theta)^2$ .

We start with  $\theta_0 = 0$ .

We are using the optimizer:  $\theta_{t+1} = \theta_t - \alpha_t M_t \nabla f_t(\theta_t)$ . But here  $f_t$  is the same loss?

Actually, since there is only one training point, then each epoch? But careful: " $M_t$  is recomputed over each epoch of training". And it is a diagonal matrix with entries: inverse square roots of the mean squared value for the gradients during the epoch for that coordinate.

Since there is only one training point, then the gradient computed at any iteration is the same? Actually, careful: The optimizer uses the gradient  $\nabla f_t(\theta_t)$ . But  $f_t$  is the loss at iteration  $t$ . But since there is only one data point, then  $f_t(\theta) = (1 - x^\top \theta)^2$ , so  $\nabla f_t(\theta) = -2(1 - x^\top \theta)x$ . So it's always in the direction of  $x$ . So then the gradients along each coordinate are proportional to the components of  $x$ . So if we compute the mean squared value over an epoch, since there is only one data point? But wait: "recomputed over each epoch" might mean that we compute the average of the squared gradients over the iterations in that epoch. But since the gradient is the same function? Actually, it might be that we are doing full-batch? But then it's just one gradient per epoch? But then the mean squared value for each coordinate would be the square of the gradient component? But then  $M_t$  would be diagonal with entries:  $1/\sqrt{(1/\text{gradient}_i^2)^2}$ ? But then that would be  $1/|\text{gradient}_i|$ . But then the update becomes:  $\theta_{t+1} = \theta_t - \alpha_t (1/|\nabla f(\theta_t)_i|) \nabla f(\theta_t)_i$ . That would essentially normalize each coordinate's step by the magnitude of that coordinate's gradient. But then the update in coordinate  $i$  becomes:  $\theta_i \leftarrow \theta_i - \alpha_t \text{sign}(\nabla f(\theta_t)_i)$ . That is like sign descent. But wait, careful: "mean squared value for the gradients during the epoch" might be computed as the average over the iterations in that epoch. But if we have only one data point, then the gradient is the same at every iteration? Actually, no, because  $\theta$  changes, so the gradient changes. But then the mean squared value over an epoch is the average of  $(\nabla f(\theta_t)_i)^2$  over  $t$  in that epoch. But then as we converge, the gradient goes to zero. But then  $M_t$  becomes large? That seems problematic.

Maybe we interpret: The optimizer O is a simplified Adam-style. Adam uses running averages of squared gradients. But here it says: "Mt 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." So that means that at the beginning of each epoch, we reset some accumulator? And then during the epoch, we compute the gradients at each step, and then at the end of the epoch, we compute the mean of the squares of these gradients along each coordinate. Then Mt for the next epoch is set to be  $\text{diag}(1/\sqrt{\text{mean\_square\_i}})$ . And then we use that Mt for all steps in that epoch? That is a simplified version? Possibly.

But since we have only one data point, the training is essentially on the same loss function repeatedly. And we start from  $\theta_0=0$ . And we use step sizes  $\alpha_t$  that are appropriate to give convergence.

We want: "What specific vector  $\theta$  would the simplified version of Adam O converge to?"

Let's denote  $x = [1, 0.1, 0.01]^T$ . Then  $f(\theta) = (1 - x^T \theta)^2$ . So  $\nabla f(\theta) = -2(1 - x^T \theta) x$ .

Now, consider the update with Mt. Mt is diagonal. Let's denote  $M_t = \text{diag}(m_1, m_2, m_3)$ . And these  $m_i$  are computed from the previous epoch? But then the dynamics become complicated because Mt changes per epoch.

Maybe we can analyze the fixed point condition. At convergence, we expect that the update goes to zero. That is,  $\alpha_t M_t \nabla f(\theta) = 0$ . Since  $\alpha_t > 0$  and  $M_t$  is positive definite (assuming gradients are not zero? But at convergence, gradient might be zero? But then that would be the same as SGD? But then  $\theta$  would satisfy  $x^T \theta = 1$ . But then there are many solutions. But with adaptive scaling, the algorithm might converge to a different solution. This is similar to the phenomenon in Adam and other adaptive methods that they converge to solutions that minimize a certain norm? There is known that Adam-like methods converge to solutions that are scale-invariant? Actually, there is known that for linear regression, gradient descent converges to the minimum norm solution. But adaptive methods like Adam might converge to a different solution. There is some analysis: "Adaptive methods do not converge to the minimum norm solution" etc.

Maybe we can simulate the behavior? But we need to derive analytically.

Let's consider the update rule per coordinate. Write  $\theta = (\theta_1, \theta_2, \theta_3)$ . Then  $f(\theta) = (1 - \theta_1 - 0.1\theta_2 - 0.01\theta_3)^2$ .

Then  $\nabla f(\theta) = -2(1 - \theta_1 - 0.1\theta_2 - 0.01\theta_3) \cdot (1, 0.1, 0.01)$ .

So the gradients are proportional to  $(1, 0.1, 0.01)$ . So at any point, the gradient components are in fixed ratios:  $g_1 : g_2 : g_3 = 1 : 0.1 : 0.01$ .

Now, the adaptive matrix Mt is diagonal. It is computed over an epoch. Suppose an epoch consists of T steps? But then "recomputed over each epoch" means that we compute the average of the squared gradients along each coordinate over the steps in that epoch. But since the gradients are always in the same proportion relative to each other? Actually, careful: The gradient depends on  $(1 - x^T \theta)$ . That factor is common to all coordinates. So if we denote  $A = 2(1 - x^T \theta)$  (with a sign? Actually,  $\nabla f = -2(1 - x^T \theta)x$ , so the magnitude is  $2|1 - x^T \theta|$  times  $x$ ). So then the squared gradient for coordinate i is  $[2(1 - x^T \theta)]^2 \cdot (x_i)^2$ . So the mean squared value over an epoch for coordinate i would be:  $(1/T) \sum_{t \text{ in epoch}} [2(1 - x^T \theta_t)]^2 \cdot (x_i)^2 = (x_i)^2 \cdot (1/T) \sum_{t \text{ in epoch}} [2(1 - x^T \theta_t)]^2$ . So then the diagonal entry  $M_{(ii)} = 1 / \sqrt{\text{mean squared gradient}_i} = 1 / (|x_i| \cdot \sqrt{(1/T) \sum_{t \text{ in epoch}} [2(1 - x^T \theta_t)]^2}) = 1 / (|x_i| \cdot \text{constant})$ , where constant  $= 1 / \sqrt{(1/T) \sum_{t \text{ in epoch}} [2(1 - x^T \theta_t)]^2}$ . But note that constant is the same for all coordinates. So then Mt is essentially:  $M_{(ii)} = c / |x_i|$ , where c is some positive number that might change per epoch. But wait, is that exactly? Check:  $x_1=1, x_2=0.1, x_3=0.01$ . So then  $M_{(11)} = c/1, M_{(22)} = c/0.1 = 10c, M_{(33)} = c/0.01 = 100c$ . So indeed, the adaptive matrix scales each coordinate inversely by the magnitude of the corresponding feature? That is, it gives larger steps for coordinates with smaller feature values? Actually, careful: The update is  $\theta \leftarrow \theta - \alpha \nabla f$ . And  $\nabla f_i = -2(1 - x^T \theta) x_i$ . So then the update for coordinate i is:  $\theta_i \leftarrow \theta_i - \alpha \cdot (c/|x_i|) \cdot (-2(1 - x^T \theta) x_i) = \theta_i + 2\alpha c (1 - x^T \theta) (x_i/|x_i|)$  but  $x_i$  are positive? Actually,  $x_i$  are positive: 1, 0.1, 0.01. So  $|x_i| = x_i$ . So then update:  $\theta_i \leftarrow \theta_i + 2\alpha c (1 - x^T \theta)$ . Notice that it becomes independent of  $i$ ! That is, all coordinates get the same additive update:  $\Delta \theta_i = 2\alpha c (1 - x^T \theta)$ . So then the update vector is:  $\Delta \theta = (2\alpha c (1 - x^T \theta)) \cdot (1, 1, 1)$ ? But wait, check: For  $i=1$ :  $M_{(11)}=c/1$ , so update:  $\theta_1 \leftarrow \theta_1 - \alpha (c/1) [-2(1 - x^T \theta) \cdot 1] = \theta_1 + 2\alpha c (1 - x^T \theta)$ . For  $i=2$ :  $\theta_2 \leftarrow \theta_2 - \alpha (c/0.1) [-2(1 - x^T \theta) \cdot 0.1] = \theta_2 + 2\alpha c (1 - x^T \theta)$ . So yes, indeed, all coordinates get the same increment.

So the dynamics become:  $\theta_{(t+1)} = \theta_t + \eta_t (1, 1, 1)$  where  $\eta_t = 2\alpha c_t (1 - x^T \theta_t)$ . And  $c_t$  is determined from the previous epoch's gradients. But careful:  $c_t$  is not constant over an epoch? Actually, the description: "Mt is recomputed over each epoch". That means that within one epoch, Mt is fixed. And it is computed from the mean squared gradients from the previous epoch. So if we denote epoch k, then during epoch k, we use a constant matrix  $M^{(k)}$ . And  $M^{(k)}$  is computed from the gradients in epoch k-1. But then as we converge, the gradients become small, so c becomes large? Actually,  $c = 1 / \sqrt{(1/T) \sum_{t \text{ in epoch}} [2(1 - x^T \theta_t)]^2} = 1 / (2 \cdot \sqrt{(1/T) \sum_{t \text{ in epoch}} (1 - x^T \theta_t)^2})$ . So c is inversely proportional to

Thinks through possible interpretations of the prompt to come to the correct one. This shows its robustness to small input syntax shifts, which I likely introduced with my purposefully careless copy pasting.

Reaches the correct observations regarding the direction theta is moved towards, i.e.,  $[1, 1, 1]^T$

the RMS of  $(1 - x^T \theta)$  over the epoch. So then  $\eta_t$  becomes:  $\eta_t = 2\alpha_t c_t (1 - x^T \theta_t)$ . But  $\alpha_t$  is the step size. They said "assuming appropriate step-sizes  $\alpha_t > 0$  to give convergence". So we can choose  $\alpha_t$  appropriately. Perhaps in the limit, the algorithm will converge to a point where the update goes to zero. That requires that  $\eta_t \rightarrow 0$ . That means either  $(1 - x^T \theta) \rightarrow 0$  or  $\alpha_t c_t \rightarrow 0$ . But if  $(1 - x^T \theta)$  is not zero, then eventually, the gradients are constant? Actually, if we are not at the solution, then  $(1 - x^T \theta)$  is some constant? But then  $c_t$  from the previous epoch would be computed from that constant?

**Actually, careful: The dynamics are:  $\theta$  updates by adding a vector that is always  $(1,1,1)$  times some scalar.** So that means that throughout the optimization, the parameter vector  $\theta$  always lies on a line? Initially,  $\theta_0=0$ . Then after one update,  $\theta$  becomes  $(\eta, \eta, \eta)$  for some  $\eta$ . So then  $\theta$  is always of the form  $(a, a, a)$  for some  $a$ . Let's check: If  $\theta = (a, a, a)$ , then  $x^T \theta = a^* (1+0.1+0.01) = a * 1.11$ . So then the update is:  $\Delta \theta = (\eta, \eta, \eta)$  with  $\eta = 2\alpha c (1 - 1.11a)$ . So indeed, if we start at  $(0,0,0)$ , then after one step,  $\theta$  becomes  $(\eta, \eta, \eta)$ . So then it remains of that form. So we can reduce the problem to a single scalar: Let  $\theta = (a, a, a)$ . Then the loss is  $f(a) = (1 - 1.11a)^2$ . And the gradient in each coordinate? Actually, careful: The update for each coordinate is the same. So then the dynamics are:  $a_{t+1} = a_t + \eta_t$ , where  $\eta_t = 2\alpha_t c_t (1 - 1.11a_t)$ . And  $c_t$  is determined from the previous epoch? But then  $c_t$  depends on the gradients from the previous epoch. But if we are in a steady state where  $a$  is constant? Actually, convergence means that  $a$  becomes such that the update  $\eta_t$  becomes 0. That requires  $1 - 1.11a = 0$ , so  $a = 1/1.11 = 100/111 \approx 0.9009$ . **So then the candidate is  $\theta = (a, a, a)$  with  $a = 1/1.11$ ? But wait, does that satisfy  $x^T \theta = 1$ ?  $x^T \theta = a^* (1+0.1+0.01) = 1.11a$ , so indeed  $a = 1/1.11$  gives  $x^T \theta = 1$ . So that is a solution.** But is it the one that this adaptive method converges to? But wait, there are many solutions to  $x^T \theta = 1$ . **The vanilla SGD converges to the minimum norm solution, which is  $\theta = (1/1.0101, 0.1/1.0101, 0.01/1.0101)$  as computed in part (a).** That is not of the form  $(a, a, a)$  generally? Check: For that, we have  $\theta_1 = 1/1.0101 \approx 0.990$ ,  $\theta_2 = 0.1/1.0101 \approx 0.099$ ,  $\theta_3 = 0.01/1.0101 \approx 0.0099$ . So that is not all equal. So the adaptive method seems to drive all coordinates to be equal? But wait, is that necessarily the case? Let's check the update carefully: We have  $\theta_{t+1} = \theta_t - \alpha_t M_t \nabla f(\theta_t)$ . And we computed that  $M_t$  is diagonal with entries:  $M_{t(ii)} = c / x_i$ , where  $c = 1/(2 * (\text{RMS of } (1 - x^T \theta) \text{ over epoch}))$ . But careful: That was if the mean squared gradient for coordinate  $i$  is  $(x_i)^2 * (\text{mean of } [2(1 - x^T \theta)]^2)$ . So then  $M_{t(ii)} = 1 / \sqrt{2 * (x_i)^2 * E} = 1 / (|x_i| \sqrt{2E}) = 1 / (x_i \sqrt{2E})$  because  $x_i > 0$ . So then  $M_t \nabla f(\theta) = \text{diag}(1/(x_i \sqrt{2E})) * (-2(1 - x^T \theta) x) = -2(1 - x^T \theta) / \sqrt{2E} * (1, 1, 1)$  because for each coordinate  $i$ :  $(1/(x_i)) * x_i = 1$ . So indeed, the update is:  $\theta_{t+1} = \theta_t + (2\alpha_t / \sqrt{2E}) (1 - x^T \theta_t) * (1, 1, 1)$ . So the update always adds a vector that is constant across coordinates. So if we start at  $\theta_0=0$ , then after one update,  $\theta_1 = (\delta, \delta, \delta)$  for some  $\delta$ . And then subsequently, if  $\theta_t = (a, a, a)$ , then  $x^T \theta_t = a(1+0.1+0.01) = 1.11a$ , so the update is again  $(\delta', \delta', \delta')$ . So indeed, the entire trajectory remains in the subspace where  $\theta_1 = \theta_2 = \theta_3$ . So the adaptive method will converge to a point on that line that satisfies  $x^T \theta = 1$ . And on that line,  $\theta = (a, a, a)$ , so  $x^T \theta = 1.11a = 1$ , so  $a = 1/1.11$ . **So then the convergent point is  $\theta = (1/1.11, 1/1.11, 1/1.11)$ . But wait,  $1/1.11 = 100/111 \approx 0.9009$ . So that is the answer.**

update is again  $(\delta', \delta', \delta')$ . So indeed, the entire trajectory remains in the subspace where  $\theta_1 = \theta_2 = \theta_3$ . So the adaptive method will converge to a point on that line that satisfies  $x^T \theta = 1$ . And on that line,  $\theta = (a, a, a)$ , so  $x^T \theta = 1.11a = 1$ , so  $a = 1/1.11$ . **So then the convergent point is  $\theta = (1/1.11, 1/1.11, 1/1.11)$ . But wait,  $1/1.11 = 100/111 \approx 0.9009$ . So that is the answer.**

But let's double-check: The adaptive method effectively normalizes the gradients by the feature scales? Actually, it gives each coordinate an update that is proportional to the sign of the gradient? But here since all gradients have the same sign? Actually, the gradient components are all positive? Because  $x$  is positive and  $(1 - x^T \theta)$  might be positive or negative. But initially, at  $\theta=0$ ,  $1 - x^T \theta = 1$ , so gradients are negative? Actually,  $\nabla f = -2(1 - x^T \theta)x$ , so at  $\theta=0$ ,  $\nabla f = -2(1)x = -2*(1, 0.1, 0.01)$ . So the gradients are negative. Then the update with  $M_t$ :  $M \nabla f = -2(1 - x^T \theta) / \sqrt{2E} * (1, 1, 1)$  so that is also negative? Actually, careful:  $\theta_{t+1} = \theta_t - \alpha_t M_t \nabla f(\theta_t)$ . But we computed  $M_t \nabla f(\theta_t) = ?$  Let's recompute carefully:

$$f(\theta) = (1 - x^T \theta)^2.$$

$$\nabla f(\theta) = -2(1 - x^T \theta) x.$$

$M_t$  is diagonal with entries:  $M_{t(ii)} = 1 / \sqrt{v_i}$  where  $v_i$  is the mean squared gradient for coordinate  $i$  over the epoch.

But as argued,  $v_i = (x_i)^2 * E$ , where  $E = (1/T) \sum [2(1 - x^T \theta)]^2$ . Actually, careful: The gradient at time  $t$  is:  $g_t = -2(1 - x^T \theta_t) x$ . So its square per coordinate:  $(g_{t(i)})^2 = 4(1 - x^T \theta_t)^2 (x_i)^2$ .

So the mean over an epoch:  $v_i = (1/T) \sum_t 4(1 - x^T \theta_t)^2 (x_i)^2 = 4 (x_i)^2 * (1/T) \sum_t (1 - x^T \theta_t)^2$ .

So then  $\sqrt{v_i} = 2 |x_i| * \sqrt{(1/T) \sum_t (1 - x^T \theta_t)^2} = 2 x_i * \sqrt{(1/T) \sum_t (1 - x^T \theta_t)^2}$  because  $x_i > 0$ .

So then  $M_{t(ii)} = 1/(2 x_i * \sqrt{(1/T) \sum_t (1 - x^T \theta_t)^2})$ .

So then  $M_t \nabla f(\theta)$  has components:  $(M_t \nabla f(\theta))_i = M_{t(ii)} * (-2(1 - x^T \theta) x_i) = -2(1 - x^T \theta) x_i / (2 x_i * \sqrt{(1/T) \sum_t (1 - x^T \theta_t)^2}) = -(1 - x^T \theta) / \sqrt{(1/T) \sum_t (1 - x^T \theta_t)^2}$ .

So indeed,  $M_t \nabla f(\theta) = -(1 - x^T \theta) / D * (1, 1, 1)$  where  $D = \sqrt{(1/T) \sum_t (1 - x^T \theta_t)^2}$ .

Then the update is:  $\theta_{t+1} = \theta_t - \alpha_t M_t \nabla f(\theta_t) = \theta_t + \alpha_t (1 - x^T \theta_t) / D * (1, 1, 1)$ .

So it's exactly:  $\theta_{t+1} = \theta_t + \eta_t (1, 1, 1)$  with  $\eta_t = \alpha_t (1 - x^T \theta_t) / D$ .

And  $D$  is computed from the previous epoch? Actually, careful: " $M_t$  is recomputed over each epoch". That means that at the beginning of an epoch, we compute  $D$  from the previous epoch's  $(1 - x^T \theta)$  values. So if we are in epoch  $k$ , then  $D_k$  is constant throughout that epoch. And then the update in epoch  $k$  is:  $\theta_{t+1} = \theta_t + (\alpha_t / D_k) (1 - x^T \theta_t) (1, 1, 1)$ . And if we use a constant step size  $\alpha_t = \alpha$  within an epoch, then the dynamics in that epoch are:  $\theta$  evolves by adding a constant vector direction. So indeed,  $\theta$  remains of the form  $(a, a, a)$ . And

We can see how meticulous Deepseek's reasoning is. From the step of finding the gradient direction vector to finding the vector of convergence, Deepseek carries out a similar think and doubt and check and repeat process as part (a), even though theoretically speaking everything is the same for part (b), thus the process of going from update vector to convergence vector should be much more straightforward for (b), i.e., just change some numbers. Deepseek, however, revalidates its entire approach.

From the perspective of the end result, this is a plus since better safe than sorry. But connecting to recent lectures on test-time compute, I wonder if this is a point of inefficiency. Since Deepseek is spending more than double the tokens needed in test-time thinking to reach the same quality solution.

then  $x^T \theta = 1.11 a$ . So then the update becomes:  $a_{t+1} = a_t + (\alpha/D_k)(1 - 1.11 a_t)$ . So that is a linear recurrence. For convergence, we need that eventually  $a$  becomes such that  $1 - 1.11 a = 0$ , so  $a = 1/1.11$ . **So the adaptive method converges to  $\theta = (1/1.11, 1/1.11, 1/1.11)$ .**

**But wait: Is that independent of the step sizes? They said "assuming appropriate step-sizes  $\alpha_t > 0$  to give convergence". So yes, we assume that the step sizes are chosen so that the process converges.** And then the fixed point is where the update is zero, which is when  $1 - x^T \theta = 0$ . But on the line  $\theta = (a, a, a)$ , that gives  $a = 1/1.11$ . So the answer is  $\theta = (1/1.11, 1/1.11, 1/1.11)$ . But we can simplify:  $1/1.11 = 100/111$ ? Actually,  $1.11 = 111/100$ , so  $1/1.11 = 100/111$ . So  $\theta = (100/111, 100/111, 100/111)$ .

**But wait, is that the only attractor? What if the process starts at a point not on the line?** But we start at 0, and the update always adds a vector that is  $(1,1,1)$  direction. So indeed, the entire trajectory lies on the line spanned by  $(1,1,1)$ . So the convergent point is uniquely determined as the intersection of that line with the plane  $x^T \theta = 1$ . And that intersection is  $\theta$  such that  $\theta_1 = \theta_2 = \theta_3$  and  $1.11 \theta_1 = 1$ , so  $\theta_1 = 1/1.11$ .

Thus final answer for (b):

$$\theta = (1/1.11, 1/1.11, 1/1.11) = (100/111, 100/111, 100/111).$$

But wait, the problem might expect a vector:

$$[1/1.11, 1/1.11, 1/1.11] \text{ but maybe simplified as } [100/111, 100/111, 100/111].$$

I'll put:

$$\begin{bmatrix} 100 & 100 & 100 \\ 111 & 111 & 111 \end{bmatrix}.$$

But let's check: For part (a), we got:  $\theta = (1/1.0101, 0.1/1.0101, 0.01/1.0101)$ . And  $1.0101 = 1 + 0.01 + 0.0001 = 1.0101$ . So that is different.

So answers:

$$(a) \begin{bmatrix} 1 & 0.1 & 0.01 \\ 1.0101 & 1.0101 & 1.0101 \end{bmatrix}$$

$$(b) \begin{bmatrix} 100 & 100 & 100 \\ 111 & 111 & 111 \end{bmatrix}$$

Some more double-checking that we see very often, or in fact, every time it "deep thinks"

## 🧠 Step 1: Problem Setup

We have a single training point defined by the equation:

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

The loss function is the squared error:

$$f(\theta) = (1 - [1, 0.1, 0.01]\theta)^2$$

Let:

$$x = \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$$

Then the loss becomes:

$$f(\theta) = (1 - x^T \theta)^2$$

The gradient is:

$$\nabla f(\theta) = -2(1 - x^T \theta)x$$

We start from:

$$\theta_0 = 0$$

## 🔄 Step 2: Vanilla SGD (Part a)

The update rule for vanilla SGD is:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$$

Substituting the gradient:

$$\theta_{t+1} = \theta_t + 2\alpha(1 - x^T \theta_t)x$$

At convergence,  $\theta_{t+1} = \theta_t = \theta^*$ , so:

$$0 = 2\alpha(1 - x^T \theta^*)x$$

Since  $x \neq 0$ , we must have:

$$x^\top \theta^* = 1$$

This equation has infinitely many solutions. Starting from  $\theta_0 = 0$ , the iterates always move in the direction of  $x$ , so the solution is the minimum-norm solution:

$$\theta^* = \frac{x}{\|x\|^2}$$

Compute:

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

Thus:

$$\theta^* = \frac{1}{1.0101} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$$

### Step 3: Simplified Adam-Style Optimizer (Part b)

The update rule is:

$$\theta_{t+1} = \theta_t - \alpha_t M_t \nabla f(\theta_t)$$

Where  $M_t$  is a diagonal matrix with entries:

$$M_{ii} = \frac{1}{\sqrt{\text{mean squared gradient for coordinate } i}}$$

Since there is only one training point, the gradient is always proportional to  $x$ . The mean squared gradient for coordinate  $i$  is:

$$v_i = 4x_i^2 \cdot \text{mean}[(1 - x^\top \theta)^2]$$

So:

$$M_{ii} = \frac{1}{2x_i \sqrt{\text{mean}[(1 - x^\top \theta)^2]}}$$

Then:

$$M_t \nabla f(\theta) = - \frac{1 - x^\top \theta}{\sqrt{\text{mean}[(1 - x^\top \theta)^2]}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Thus, the update becomes:

$$\theta_{t+1} = \theta_t + \eta_t \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \eta_t = \frac{\alpha_t (1 - x^\top \theta_t)}{\sqrt{\text{mean}[(1 - x^\top \theta_t)^2]}}$$

Since we start from  $\theta_0 = 0$ , all updates are in the direction of  $(1, 1, 1)^\top$ , so:

$$\theta = \begin{bmatrix} a \\ a \\ a \end{bmatrix}$$

Then:

$$x^\top \theta = a(1 + 0.1 + 0.01) = 1.11a$$

At convergence, the update must vanish, so:

$$1 - 1.11a = 0 \quad \Rightarrow \quad a = \frac{1}{1.11} = \frac{100}{111}$$

Thus:



$$\theta^* = \begin{bmatrix} \frac{100}{111} \\ \frac{100}{111} \\ \frac{100}{111} \end{bmatrix}$$

## ✓ Final Answers

(a) Vanilla SGD converges to:

$$\begin{bmatrix} \frac{1}{1.0101} \\ \frac{1.0101}{0.1} \\ \frac{1.0101}{0.01} \end{bmatrix}$$

(b) Simplified Adam-style optimizer converges to:

$$\begin{bmatrix} \frac{100}{111} \\ \frac{100}{111} \\ \frac{100}{111} \end{bmatrix}$$

(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  $\theta$  would it give as its final solution (for use in original coordinates)?

Again, I paste in the part directly without addition formatting fixes or any filler words.

✖ Thought for 61 seconds ▾

- We are given: "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)?"

Interpretation: We have a single training point:  $[1, 0.1, 0.01]$   $\theta = 1$ . But then we do feature rescaling. The features are: first feature: 1, second: 0.1, third: 0.01. Their second-moments? Actually "training input feature rescaling" so that each feature had unit second-moment. But careful: There is only one training point. So the second-moment of a feature is just the square of that feature value? But then "unit second-moment" means we scale each feature so that its squared value becomes 1? But then there is only one data point. Alternatively, maybe we consider the dataset? But it's only one point. So then the second-moment of feature  $j$  is  $E[x_j^2]$ . With one point, that is just  $x_j^2$ . So to have unit second-moment, we divide feature  $j$  by  $|x_j|$ ? But then careful: "so that each feature had unit second-moment" means that if originally feature  $j$  has values  $x_j$ , then we want to transform them to  $x_j'$  such that  $(x_j')^2 = 1$ ? But then that would be:  $x_j' = x_j / |x_j|$ ? But then the data point becomes:  $[1/1, 0.1/0.1, 0.01/0.01] = [1, 1, 1]$ . But wait: That is if we use the second-moment from the training data? But then the problem says: "training input feature rescaling" so that each feature had unit second-moment. That typically means we compute the standard deviation of each feature over the training set and then divide by that standard deviation. But here there is only one training point. So the second-moment (variance? Actually second-moment is  $E[x^2]$  if mean is 0? But here mean might not be 0? But "second-moment" usually means the average of squares. With one data point, the average is just that square. So indeed, we want to scale each feature so that its squared value is 1. But careful: There is also the intercept? But here we have three features. **So we rescale: Let new features be:  $u_1 = x_1 / s_1$ ,  $u_2 = x_2 / s_2$ ,  $u_3 = x_3 / s_3$ , where  $s_1^2 = (1^2) = 1$ ,  $s_2^2 = (0.1^2) = 0.01$ ,  $s_3^2 = (0.01^2) = 0.0001$ . So  $s_1 = 1$ ,  $s_2 = 0.1$ ,  $s_3 = 0.01$ .**

**Then the transformed input becomes:  $u = [1/1, 0.1/0.1, 0.01/0.01] = [1, 1, 1]$ .**

So in the rescaled coordinates, the equation becomes:  $[1, 1, 1] \cdot \phi = 1$ , where  $\phi$  is the parameter in the rescaled space. And we run SGD to convergence. But careful: The loss in rescaled coordinates? Actually, we have: original:  $f(\theta) = (1 - x^T \theta)^2$ . After rescaling, we define new features  $u$  such that  $u_i = x_i / s_i$ . And then we have  $\theta$  such that? Actually, if we let  $\theta$  be the original parameters, then  $x^T \theta = \sum_i x_i \theta_i$ . And if we define new parameters  $\phi$  such that  $\theta_i = \phi_i / s_i$ ? Because then  $x^T \theta = \sum_i x_i ( \phi_i / s_i ) = \sum_i u_i \phi_i = u^T \phi$ . So indeed, the problem in rescaled coordinates is: minimize  $(1 - u^T \phi)^2$ , with  $u = [1, 1, 1]$ . And then we run SGD on  $\phi$ . And as computed in part (a) for vanilla SGD, starting from  $\phi = 0$ , we get convergence to the minimum norm solution:  $\phi^* = u / \|u\|^2$ . And  $\|u\|^2 =$

$1^2 + 1^2 + 1^2 = 3$ . So  $\phi^* = [1/3, 1/3, 1/3]$ . Then convert back to original coordinates:  $\theta_i = \phi_i / s_i$ . But careful:  $s_i$  are:  $s_1=1, s_2=0.1, s_3=0.01$ . So then  $\theta^* = [1/3, 1/(3*0.1)=1/0.3 = 10/3, 1/(3*0.01)=100/3]$

Actually:  $1/(3*0.1) = 1/0.3 = 10/3$ , and  $1/(3*0.01)=1/0.03 = 100/3$ . So  $\theta^* = [1/3, 10/3, 100/3]$

But wait check: Then  $x^T \theta = 1*(1/3) + 0.1*(10/3) + 0.01*(100/3) = 1/3 + 1/3 + 1/3 = 1$  So that works.

But wait: Is that the answer? Possibly yes.

But let's double-check: The process: Feature rescaling: We want each feature to have unit second-moment. That means we compute the second moment of each feature over the training set. Here there is only one training point, so the second moment for feature  $j$  is  $x_j^2$ . So we scale feature  $j$  by  $1/|x_j|$ . So then the transformed feature vector becomes:  $[1/1, 0.1/0.1, 0.01/0.01] = [1, 1, 1]$ . Then we run SGD on the transformed problem. The transformed problem is: minimize  $(1 - [1, 1, 1]\phi)^2$ . Vanilla SGD from  $\phi=0$  converges to  $\phi$  such that  $[1, 1, 1]\phi = 1$  and  $\phi$  is in the direction of  $[1, 1, 1]$  (minimum norm solution). That gives  $\phi = (1/3, 1/3, 1/3)$ . Then we convert back to original coordinates. The conversion: originally, we had  $\theta$  such that  $x^T \theta = (s_1 u_1) \theta_1 + \dots$  but careful: The rescaling transformation: We define  $u_i = x_i / s_i$ , and then we have  $\theta_i = \phi_i / s_i$ ? Actually, check: We want:  $x^T \theta = \sum x_i \theta_i$ . And if we set  $\theta_i = \phi_i / s_i$ , then  $x_i \theta_i = (s_i u_i)(\phi_i / s_i) = u_i \phi_i$ . So yes. So then  $\theta_i = \phi_i / s_i$ . So  $\theta = [1/3 / 1, 1/3 / 0.1, 1/3 / 0.01] = [1/3, 10/3, 100/3]$ . So answer:  $\theta = [1/3, 10/3, 100/3]$ .

Thus final answer for (c):

$$\begin{bmatrix} 1 & 10 & 100 \\ 3 & 3 & 3 \end{bmatrix}$$

## 🧠 Step 1: Understand the Rescaling Process

We have a single training point defined by:

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

Let:

$$x = \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$$

The loss function is:

$$f(\theta) = (1 - x^T \theta)^2$$

We rescale each feature so that it has **unit second-moment**. Since there is only one data point, the second-moment for feature  $i$  is simply  $x_i^2$ . Thus, we define scaling factors:

$$s_1 = 1, \quad s_2 = 0.1, \quad s_3 = 0.01$$

The rescaled feature vector becomes:

$$u = \begin{bmatrix} 1/s_1 \\ 0.1/s_2 \\ 0.01/s_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

## 🔄 Step 2: Reformulate the Problem in Rescaled Coordinates

We introduce new parameters  $\phi$  such that:

$$\theta_i = \frac{\phi_i}{s_i}$$

Then:

$$x^T \theta = \sum x_i \theta_i = \sum (s_i u_i) \left( \frac{\phi_i}{s_i} \right) = u^T \phi$$

So the loss in rescaled coordinates is:

$$f(\phi) = (1 - u^T \phi)^2$$

with  $u = [1, 1, 1]^T$ .

## 🎯 Step 3: Run SGD in Rescaled Coordinates

Pretty straightforward chain of thoughts seen here for part (c). Not much large scale double checking until the very end. Thinking process was also fairly quick as we can see. Correct final answer found.

Intermediate observations are all correct, up to certain notation ambiguities, but I've found this is prone to happen during thinking mode where the audience is not really meant to be the user but the model itself. This makes sense since using simpler notation potentially decreases test-time compute by cutting down on number of tokens needed to express the same thought (as opposed to all the additional syntax tokens needed for LaTeX).

We start from  $\phi_0 = 0$  and use vanilla SGD. The update rule is:

$$\phi_{t+1} = \phi_t + 2\alpha(1 - u^\top \phi_t)u$$

At convergence,  $\phi_{t+1} = \phi_t = \phi^*$ , so:

$$0 = 2\alpha(1 - u^\top \phi^*)u$$

Since  $u \neq 0$ , we must have:

$$u^\top \phi^* = 1$$

This equation has infinitely many solutions. Starting from  $\phi_0 = 0$ , the iterates move in the direction of  $u$ , so the solution is the **minimum-norm solution**:

$$\phi^* = \frac{u}{\|u\|^2}$$

Compute:

$$\|u\|^2 = 1^2 + 1^2 + 1^2 = 3$$

Thus:

$$\phi^* = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

#### Step 4: Convert Back to Original Coordinates

Recall:

$$\theta_i = \frac{\phi_i}{s_i}$$

So:

$$\theta_1 = \frac{1/3}{1} = \frac{1}{3}, \quad \theta_2 = \frac{1/3}{0.1} = \frac{10}{3}, \quad \theta_3 = \frac{1/3}{0.01} = \frac{100}{3}$$

#### Final Answer

$$\begin{bmatrix} \frac{1}{3} \\ \frac{10}{3} \\ \frac{100}{3} \end{bmatrix}$$