

EECS 182 Deep Neural Networks
Fall 2026 Gireeja Ranade

Homework 3

1. Optimizers and their convergence

Different optimizers can fit the same training data while selecting different parameter vectors. We will compare three such choices.

We have exactly $n = 1$ training point corresponding to the single equation

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

with a 3-dimensional learnable parameter vector θ . Suppose that we start with $\theta_0 = \mathbf{0}$ and use squared loss $f(\theta) = (1 - [1, 0.1, 0.01]\theta)^2$.

There is only one training example, so ordinary SGD is identical to full-batch gradient descent here. You are asked to find the selected parameter vectors, not to prove convergence. *Hint: Identify which directions each update can move in from the zero initialization.*

- (a) Standard SGD uses the update $\theta_{t+1} \leftarrow \theta_t - \alpha \nabla f(\theta_t)$, with a fixed step size $\alpha > 0$. **What specific vector θ would it converge to, assuming α is small enough to give convergence?**

Solution: SGD in this case is identical to gradient descent since there is only one training point. To get convergence with a constant step size, we need the gradient itself to get to zero. That happens only when (1) is actually solved. Gradient-descent on squared-loss can only move θ in the direction

of $\begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$ and given that we start at $\theta = \mathbf{0}$, this means that we are looking for a solution of the form $\theta = \beta \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$. Plugging this into (1), we get

$$\beta[1, 0.1, 0.01] \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix} = 1 \quad (2)$$

which has the solution $\beta = \frac{1}{1.0101}$ giving us the overall answer $\theta^* = \frac{1}{1.0101} \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}$.

Notice that this is quite insensitive to the second and third coordinates, and if the test-data is like the training data in having small numbers there, the overall impact of anything there will be quite small indeed.

- (b) Consider $\mathcal{O}$: a simplified Adam-style optimizer with instantaneous gradient normalization, no momentum, and no weight decay:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha_t M_t \nabla f(\boldsymbol{\theta}_t) \quad (3)$$

where $\alpha_t > 0$ is the step size (learning rate) at iteration t . Let $g_t = \nabla f(\boldsymbol{\theta}_t)$. If $g_t = 0$, stop updating; otherwise, let M_t be diagonal with $(M_t)_{ii} = 1/|(g_t)_i|$. For this training example, either all gradient coordinates are nonzero or all are zero. This idealized normalization uses only the current gradient, rather than Adam's moving averages.

What specific vector $\boldsymbol{\theta}$ would this optimizer converge to, assuming the step sizes $\alpha_t > 0$ are chosen so that the iterates converge to zero training loss? You do not need to derive such a step-size schedule. *Hint: Write the gradient as a scalar times the training input vector, then apply M_t to each coordinate.*

Solution: Let $r_t = 1 - [1, 0.1, 0.01]\boldsymbol{\theta}_t$ be the prediction error. Then

$$g_t = -2r_t \begin{bmatrix} 1 \\ 0.1 \\ 0.01 \end{bmatrix}.$$

All three gradient coordinates have the same sign. Dividing each by its own magnitude removes their different scales, so for $r_t \neq 0$,

$$M_t g_t = -\text{sign}(r_t) \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t = \alpha_t \text{sign}(r_t) \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Thus each update changes all three parameters by the same amount: they increase when the prediction is below 1 and decrease after an overshoot. Starting from zero, the parameters stay equal, so $\boldsymbol{\theta} = \beta[1, 1, 1]^\top$. At zero training loss,

$$\beta[1, 0.1, 0.01] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1 \quad (4)$$

which has the solution $\beta = \frac{1}{1.11}$ giving us the overall answer $\boldsymbol{\theta}_a^* = \frac{1}{1.11} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

In general, a fixed positive step size oscillates around the solution unless an iterate lands exactly on it and stops. A suitable diminishing step-size schedule allows convergence.

Further notice that this solution $\boldsymbol{\theta}_a^*$ is significantly more sensitive to the second and third coordinates, but the weight on the first coordinate is not that different from the previous SGD solution. Consequently, as long as the test-data is like the training data in having small numbers in the second and third coordinate, by that very nature, they won't impact the final prediction by much.

- (c) Consider a learning approach that first rescales the training input features: for feature j , divide by its empirical root-mean-square value $s_j = \sqrt{\frac{1}{n} \sum_{i=1}^n x_{ij}^2}$, without subtracting a mean. Here $n = 1$ and $[x_{11}, x_{12}, x_{13}] = [1, 0.1, 0.01]$. The approach then initializes the rescaled parameters to zero, runs SGD to zero training loss, and converts 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)?**

Solution: With one training example, each feature's RMS is its absolute value: $(s_1, s_2, s_3) = (1, 0.1, 0.01)$. Dividing each feature by its RMS therefore multiplies the second feature by 10 and the third by 100: $[1/1, 0.1/0.1, 0.01/0.01] = [1, 1, 1]$. This gives us a new “normalized” problem:

$$[1, 1, 1]\tilde{\theta} = 1 \quad (5)$$

with initial condition $\tilde{\theta} = \mathbf{0}$. Immediately by symmetry (or by following the same argument above),

this has the solution $\tilde{\theta} = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$. To express the same predictions using the original features, use

$\sum_j \tilde{\theta}_j (x_j/s_j) = \sum_j (\tilde{\theta}_j/s_j) x_j$. Thus the original-coordinate weights are $\theta_j = \tilde{\theta}_j/s_j$, giving the final answer: $\theta_r^* = \frac{1}{3} \begin{bmatrix} 1 \\ 10 \\ 100 \end{bmatrix}$

Notice that this is far more sensitive to the second and third coordinates. Even if test data had small numbers like the training data in those positions, they would significantly impact the predictions.

This illustrates the different inductive biases of these three different approaches to solving an underdetermined equation.

2. Gradient Descent Doesn't Go Nuts with Ill-Conditioning

Consider a linear regression problem with n training points and d features. When $n = d$, the feature matrix $F \in \mathbb{R}^{n \times n}$ has some maximum singular value α and an extremely tiny minimum singular value. We have noisy observations $\mathbf{y} = F\mathbf{w}^* + \epsilon$. If we compute $\hat{\mathbf{w}}_{inv} = F^{-1}\mathbf{y}$, then due to the tiny singular value of F and the presence of noise we observe that $\|\hat{\mathbf{w}}_{inv} - \mathbf{w}^*\|_2 = 10^{10}$.

Suppose instead of inverting the matrix we decide to use gradient descent instead. We run k iterations of gradient descent to minimize the loss $\ell(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - F\mathbf{w}\|_2^2$ starting from $\mathbf{w}_0 = \mathbf{0}$. We use a learning rate η which is *small enough* that gradient descent cannot possibly diverge for the given problem. **(This is important. You will need to use this.)**

The gradient-descent update for $t > 0$ is:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \left(F^\top (F\mathbf{w}_{t-1} - \mathbf{y}) \right).$$

We are interested in the error $\|\mathbf{w}_k - \mathbf{w}^*\|_2^2$. We want to show that in the worst case, this error can grow at most linearly with iterations k and in particular $\|\mathbf{w}_k - \mathbf{w}^*\|_2 \leq k\eta\alpha\|\mathbf{y}\|_2 + \|\mathbf{w}^*\|_2$.

i.e. The error cannot go “nuts,” at least not very fast.

For the purposes of the homework, you only have to prove the key idea, since the rest follows by applying induction and the triangle inequality.

Show that for $t > 0$, $\|\mathbf{w}_t\|_2 \leq \|\mathbf{w}_{t-1}\|_2 + \eta\alpha\|\mathbf{y}\|_2$.

(HINT: What do you know about $(I - \eta F^\top F)$ if gradient descent cannot diverge? What are its eigenvalues like? Use this fact.)

Solution: We have,

$$\begin{aligned}\|w_t\|_2 &= \|w_{t-1} - \eta(F^\top(Fw_{t-1} - y))\|_2 \\ &= \|(I - \eta F^\top F)w_{t-1} + \eta F^\top y\|_2 \\ &\leq \|(I - \eta F^\top F)w_{t-1}\|_2 + \|\eta F^\top y\|_2 \\ &\leq \sigma_{\max}(I - \eta F^\top F)\|w_{t-1}\|_2 + \eta \sigma_{\max}(F)\|y\|_2 \\ &\leq \|w_{t-1}\|_2 + \eta \alpha \|y\|_2,\end{aligned}$$

where we used the fact that η is chosen so that gradient descent does not diverge so the maximum eigenvalue of $(I - F^\top F)$ cannot have an absolute value greater than 1. But $I - F^\top F$ is a real symmetric matrix and the spectral theorem tells us that the singular values are therefore just the absolute values of the eigenvalues. This means that the maximum singular value also must have absolute value less than 1.

3. Initialization and Optimizers

Use the supplied experiments in the [Fall 2026 notebook](#), named `hw2_optimizer_init.ipynb`, to explore initialization and optimizers. The implementations and experiment settings are provided. Run the notebook and use its **Initialization** section to answer the following discussion question in your homework PDF.

- (a) In the plot titled “**Mean of the Gradient Norm,**” compare zero, random Gaussian, and He initialization. Explain the differences and relate them to the observed training behavior. The plotted value is the average, over all weight and bias tensors, of the Euclidean norm of each flattened loss-gradient tensor. It is not the norm of the average gradient.

Hint: Consider how signals and gradients pass through many ReLU layers, and whether the output bias can change even if hidden features do not.

Solution: In this experiment the fixed random initialization uses standard deviation 0.01 in a network with ten hidden layers of width 50. Signals and weight gradients can become very small across this depth, slowing learning. He initialization scales the weights with the layer width to better preserve signal and gradient magnitudes.

With all weights and biases initialized to zero, this deep ReLU network has zero weight gradients and cannot learn differentiated hidden features. Its final output bias can still receive a nonzero gradient from a minibatch with unequal class frequencies. Because the plotted average includes biases, the zero-initialization curve need not be exactly zero. Answers should explain the actual curves rather than assume that every random initialization trains well.

4. 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: Arrange the n workers in a logical ring. Each worker computes its gradient and splits it into n equally sized chunks. In each round, every worker sends one chunk to the next worker and receives one from the previous worker. During the first $n - 1$ rounds, workers add contributions to the received chunks; at the end, each worker holds one fully aggregated chunk. During the next $n - 1$ rounds, workers pass these aggregated chunks around the ring so that every worker obtains the complete aggregate. Each worker then averages the gradients and updates its local model parameters. This is illustrated in Fig. 1.

Fill in the missing entries in the table for one synchronous training iteration. Assume $n \geq 2$ workers and p model parameters, with p divisible by n . The parameter server is separate from the n workers. Count each directed message once across the entire system, and measure message size in scalar entries, ignoring protocol overhead.

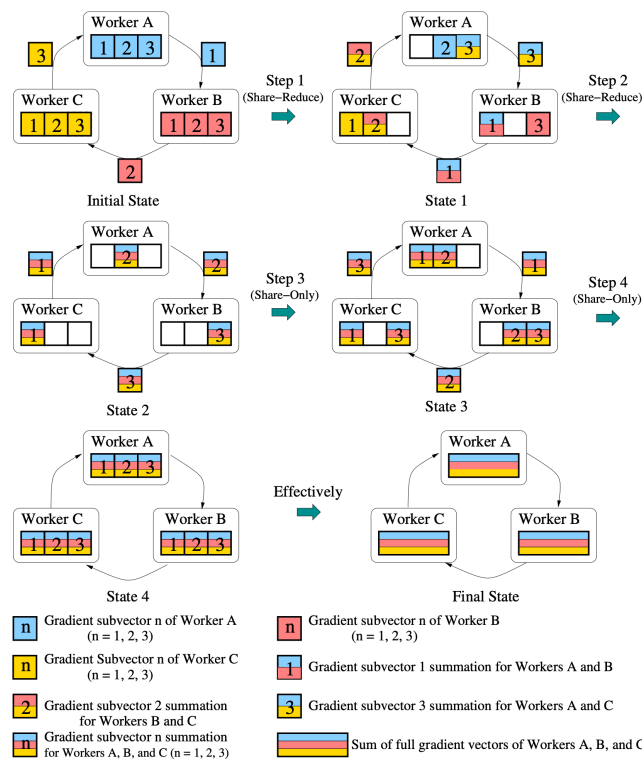


Figure 1: Example of Ring All-Reduce in a 3 worker setup. Source: Mu Et. al, *GADGET: Online Resource Optimization for Scheduling Ring-All-Reduce Learning Jobs*

Solution: All-to-All Communication: Each of the n workers sends its full gradient to $n - 1$ other workers: $n(n - 1)$ messages of p entries each.

Parameter Server: The server receives n gradients and sends updated parameters to n workers: $2n$ messages of p entries each.

Ring All-Reduce: Each of the n workers sends one chunk in each of $2(n - 1)$ rounds: $2n(n - 1)$ messages of p/n entries each. Thus the total traffic is $2(n - 1)p$, comparable to the parameter server's $2np$, while all-to-all uses $n(n - 1)p$ entries.

	Number of Messages Sent	Size of each message
All-to-All	Solution: $n(n - 1)$	p
Parameter Server	$2n$	Solution: p
Ring All-Reduce	$n(2(n - 1))$	Solution: $\frac{p}{n}$

5. Read Two Research Papers: Generalization and AlexNet

Read the following two required papers; the third paper is optional:

- Zhang, Bengio, Hardt, Recht, and Vinyals. “**Understanding deep learning requires rethinking generalization.**” *ICLR*, 2017.
- Krizhevsky, Sutskever, and Hinton. “**ImageNet Classification with Deep Convolutional Neural Networks.**” *NeurIPS*, 2012.
- **[Optional]** Chen, Peng, Chang, Sitawarin, and Wagner. “**SecPO: Principled Adversarial Training for Prompt Injection Security.**” Related to the September 17 guest lecture.

Write out three notable things about each required paper. You can comment on the figures, the different new techniques used, any implications of the paper etc.

Contributors:

- Matteo Guarrera.
- Mert Cemri.
- Sizhe Chen.
- Luke Jaffe.
- Anant Sahai.
- Kevin Li.
- Hao Liu.
- Sheng Shen.
- Andrew Ng.
- Linyuan Gong.
- Romil Bhardwaj.