SCALER School of Technology × Priyansh Saxena

Loss Functions and Optimization

MSE • Cross-Entropy • SGD & Momentum • Adam • LR Schedules

Priyansh Saxena

Scaler School of Technology | Deep Learning I: Neural Networks | Week 2 — Session 3/3

19 May 2026

Agenda & Learning Objectives

Today's Agenda

- **Part 1:** Loss functions — MSE & cross-entropy
- **Part 2:** Gradient descent variants & momentum
- **Break**
- **Part 3:** Adaptive optimizers — Adam
- **Part 4:** Learning rate schedules
- **Closing:** Summary & quiz

By the end, you will be able to:

- Choose the right loss for regression vs. classification
- Explain why mini-batch SGD is the practical standard
- Describe how momentum and Adam improve on vanilla SGD
- Apply a learning rate schedule to improve convergence

Recap & Today's Questions

Session 2: Backpropagation

- Chain rule decomposes gradients
- Computational graphs organize the math
- Backprop = forward → loss → backward → update

Session 3: Two New Questions

- **What** should we minimize? → Loss functions
- **How** should we update weights using gradients? → Optimizers

We have the engine. Today we choose the fuel and learn to drive.

Why the Loss Function Matters

The loss function defines the **optimization landscape**:

- Different losses $\rightarrow$ different gradient shapes $\rightarrow$ different training dynamics
- A poorly chosen loss can make training **impossible**

Regression

Mean Squared Error (MSE)
Penalizes errors quadratically

Classification

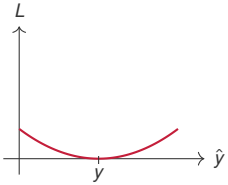
Cross-Entropy
Penalizes confident wrong answers logarithmically

"The loss function is the exam rubric — it decides what counts as a mistake and how harshly to punish it."

Mean Squared Error (MSE)

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- The default for **regression**
- Gradient: $\frac{\partial L}{\partial \hat{y}} = -\frac{2}{n}(y - \hat{y})$
- Penalizes large errors **quadratically**
- Outliers dominate the loss



Key Property

Gradient $\propto$ error.

Gradient **shrinks** as prediction gets closer to target.

$\rightarrow$ Learning slows near the answer.

MSE Gradient: Worked Example

Target $y = 1.0$. Compute MSE loss and gradient for each prediction:

Prediction $\hat{y}$	Error	MSE Loss	Gradient $\frac{\partial L}{\partial \hat{y}}$
0.20	0.80	0.640	-1.60
0.50	0.50	0.250	-1.00
0.80	0.20	0.040	-0.40
0.95	0.05	0.003	-0.10

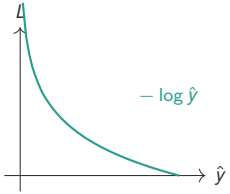
Key Insight

As the prediction approaches the target, the gradient shrinks. Learning slows down the closer you get — fine for regression, problematic for classification.

Cross-Entropy Loss (Binary)

$$L = -\big[y \log \hat{y} + (1-y) \log(1-\hat{y})\big]$$

- The default for **classification**
- When $y = 1$: $L = -\log \hat{y}$
- Gradient: $\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$
- **Catastrophic penalty** for confident wrong answers



The Logarithmic Penalty

$\hat{y} = 0.9 \rightarrow L = 0.105$
$\hat{y} = 0.1 \rightarrow L = 2.303$
$\hat{y} = 0.01 \rightarrow L = 4.605$

Being **confidently wrong** is devastatingly expensive.

MSE vs Cross-Entropy: Side by Side

Target $y = 1.0$. Compare losses and gradients:

$\hat{y}$	MSE Loss	MSE $\lvert\nabla\rvert$	CE Loss	CE $\lvert\nabla\rvert$
0.20	0.640	1.60	1.609	**5.00**
0.50	0.250	1.00	0.693	**2.00**
0.80	0.040	0.40	0.223	**1.25**
0.95	0.003	0.10	0.051	**1.05**

Takeaway

Cross-entropy gradients are **much larger** when predictions are wrong $\rightarrow$ faster learning. Even at $\hat{y} = 0.95$, CE gradient is $10\times$ larger than MSE gradient.

The Gradient Problem: MSE + Sigmoid

For classification with sigmoid output $\hat{y} = \sigma(z)$:

MSE Gradient w.r.t. z

$$\frac{\partial L}{\partial z} = -2(y - \hat{y}) \cdot \underbrace{\sigma'(z)}_{\leq 0.25}$$

$\sigma'(z)$ vanishes when neuron is saturated.

Weakest signal when most wrong!

Cross-Entropy Gradient w.r.t. z

$$\frac{\partial L}{\partial z} = \hat{y} - y$$

The $\sigma'(z)$ **cancels out!**

Strong signal when most wrong.

Cross-entropy + sigmoid are designed to work together.

The sigmoid derivative cancels — this is not a coincidence.

Multi-Class Cross-Entropy with Softmax

Softmax: $\hat{y}_k = \dfrac{e^{z_k}}{\sum_j e^{z_j}}$

CE loss: $L = -\sum_k y_k \log \hat{y}_k$

Gradient (beautifully simple):

$$\frac{\partial L}{\partial z_k} = \hat{y}_k - y_k$$

Softmax + log derivatives cancel: gradient = **prediction** − **target**.

Strong signal when wrong.

Example: 3 Classes

Logits: $[2.0,\ 1.0,\ 0.5]$

Target: class 1 $\rightarrow [0, 1, 0]$

Softmax: $[0.506, 0.265, 0.229]$

$L = -\log(0.265) = 1.328$

Gradient: $[0.506, -0.735, 0.229]$

Push class 1 up, push others down.

PyTorch tip: `nn.CrossEntropyLoss` expects raw logits — it applies softmax internally.

Quiz 1: Loss Functions

1. You're training a model to predict **house prices**. Which loss: MSE or cross-entropy?
2. With binary cross-entropy, your model predicts $\hat{y} = 0.01$ for a positive example. Is the loss closer to **0.01**, **0.5**, or **4.6**?
3. Why is MSE a **poor choice** for classification with sigmoid outputs?

Take 2 minutes. Discuss with your neighbor.

Quiz 1 — Answers

1. House prices $\rightarrow$ **MSE** (regression, not classification)
2. $-\log(0.01) = \mathbf{4.605}$. Cross-entropy devastates confident wrong answers.
3. MSE gradient includes $\sigma'(z) \leq 0.25$, which **vanishes** when the neuron is saturated. You get the weakest signal exactly when you need it most.

Discussion

"If mini-batch is the best of both worlds, why not batch size 1 (pure SGD)?"

You lose: GPU utilization, stable gradient estimates, and batch normalization requires batches.

The Gradient Descent Update Rule

$$w \leftarrow w - \eta \cdot \nabla L(w)$$

Three knobs to turn:

1. How many examples?
batch / SGD / mini-batch

2. Learning rate η?
fixed / adaptive / scheduled

3. Any extras?
momentum / adaptation

Today we turn all three knobs.

Batch Gradient Descent

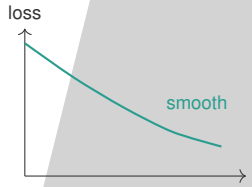
Compute gradient over the **entire dataset**:

$$\nabla L = \frac{1}{N} \sum_{i=1}^{N} \nabla L_i$$

- **One update** per epoch
- Gradient is exact — no noise
- Smooth, predictable convergence

Problem

60,000 images $\rightarrow$ 60,000 forward+backward passes for **one step**. Extremely slow.



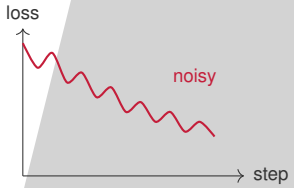
"Like reading the entire textbook before writing one sentence of your essay."

Stochastic Gradient Descent (SGD)

Compute gradient on **one random example**:

$$\nabla L \approx \nabla L_i \quad \text{(one sample)}$$

- N **updates** per epoch — start learning immediately
- Noise helps **escape local minima**
- But: very noisy, may never settle



"Like editing your essay after reading one random sentence."

Mini-Batch Gradient Descent — The Standard

Compute gradient on a **batch of B examples** (typically 32–256): $\nabla L \approx \frac{1}{B} \sum_{i=1}^{B} \nabla L_i$

The Sweet Spot

- N/B updates per epoch
- Accurate enough gradient estimate
- Fast enough per step
- Efficient GPU utilization

Common Batch Sizes

- 32 — small models, limited GPU
- 64 — general default
- 128–256 — large models, big GPUs
- Powers of 2 align with GPU memory

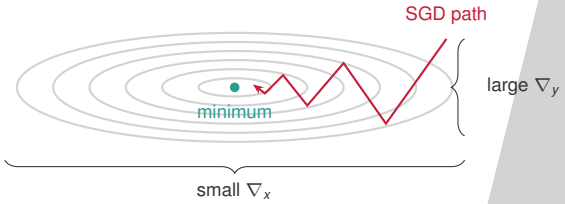
"Like reading a chapter before revising — enough context, not too slow."

Discussion: Why not batch size 1?

You lose: GPU utilization • stable gradient estimates • batch normalization requires batches.

The Problem: Elongated Loss Surfaces

Most real loss surfaces are **elongated valleys**, not circular bowls.



SGD **oscillates** across the narrow dimension and **crawls** along the long dimension.

We need something smarter. Enter: momentum.

SGD with Momentum

Add a **velocity term** that accumulates gradient history:

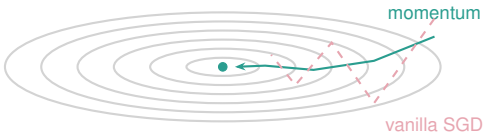
Update Rule

$$v_t = \beta \cdot v_{t-1} + \nabla L(w_t)$$

$$w_{t+1} = w_t - \eta \cdot v_t$$

$\beta = 0.9$ typically (90% memory)

- Consistent directions **build speed**
- Oscillating directions **cancel out**
- Effective window $\approx 1/(1-\beta) = 10$ steps



"A ball rolling downhill,
not a person stopping at every step."

Coming Up Next

Part 3: Adaptive Learning Rate Methods — AdaGrad, RMSProp, and Adam

The Problem with a Single Learning Rate

Different parameters live in different parts of the landscape:

	w_1	w_2
Gradient	10.0	0.01
$\eta = 0.01$	0.1 (too much!)	0.0001 (tiny)
$\eta = 0.001$	0.01 (ok)	0.00001 (frozen)

No Single η Works

A single learning rate is always a compromise.

Solution: per-parameter learning rates that adapt to gradient history.

"One shoe size for all feet doesn't work. We need custom-fit learning rates."

AdaGrad (Duchi et al., 2011)

Key idea: give each parameter its own learning rate, scaled by accumulated gradient history.

Update Rule

$G_t = G_{t-1} + g_t^2$ (accumulated squared gradients) $\quad w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t} + \varepsilon} \cdot g_t$

What It Does

Parameters with **large past gradients** $\rightarrow$ smaller effective lr.

Parameters with **small past gradients** $\rightarrow$ larger effective lr.

Perfect for **sparse features** (NLP embeddings).

The Fatal Flaw

G_t **only grows** — it never shrinks.

After many steps, $\frac{\eta}{\sqrt{G_t}} \rightarrow 0$

lr decays to zero, training stops — even without converging.

RMSProp (Hinton, 2012)

AdaGrad fix: use an **exponential moving average** of squared gradients instead of a forever-growing sum.

Update Rule

$$s_t = \gamma \cdot s_{t-1} + (1-\gamma) \cdot g_t^2 \qquad (\gamma = 0.9)$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{s_t} + \varepsilon} \cdot g_t$$

Why It Works

Recent gradients matter more.
Old gradients **fade away**.
Learning rate **recovers** when gradients shrink.

Fun Fact

Never formally published!

Hinton introduced it in slide 29 of his Coursera lecture (2012).

Most influential *unpublished* idea in DL.

Adam (Kingma & Ba, 2015)

Combine momentum + RMSProp + bias correction:

Update Rule

Momentum: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

RMSProp: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Bias correction: $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$

Update: $w_{t+1} = w_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$

Default Hyperparameters (work for ∼90% of problems)

$\beta_1 = 0.9, \quad \beta_2 = 0.999, \quad \varepsilon = 10^{-8}, \quad \eta = 0.001$

Adam: The Default Optimizer

Why Adam wins:

1. Momentum smooths noisy gradients
2. RMSProp adapts per-parameter
3. Bias correction handles cold start
4. Defaults work out of the box

The Practical Rule

Start with Adam, $\eta = 0.001$**.**
Only switch if you have a specific reason.

When to Consider Alternatives

- **Large-scale training:** SGD+momentum may generalize better (flatter minima)
- **GANs:** RMSProp sometimes preferred (Adam's momentum can cause oscillations)
- **Competition fine-tuning:** SGD+cosine schedule for last 0.5% accuracy

Optimizer Comparison

Optimizer	Key Idea	Weakness	Use When
SGD	$w \leftarrow w - \eta g$	Oscillates on elongated surfaces	Rarely alone
+Momentum	Velocity accumulates past gradients	Fixed lr for all params	Strong baseline
RMSProp	$\div \sqrt{\text{EMA}(g^2)}$ — recent grads matter more	No momentum	RNNs, GANs
Adam	Momentum + RMSProp + bias correction	May find sharp minima	**Default**

SGD → add velocity → adapt lr → combine both = Adam

Quiz 2: Optimizers

1. What does “adaptive” mean in adaptive learning rate methods?
2. AdaGrad’s learning rate always decreases. Why is this a **problem**?
3. Adam combines two ideas — name them.
   Bonus: What are the default values of β_1 and β_2?

Take 2 minutes. Discuss with your neighbor.

Quiz 2 — Answers

1. **Adaptive** = the learning rate is adjusted **per-parameter** based on gradient history.
2. s_t only grows $\rightarrow$ effective lr decays to zero $\rightarrow$ **training stops**.
3. **Momentum** (1st moment) + **RMSProp** (2nd moment). $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Discussion

"If Adam is the default, why does anyone still use SGD?"

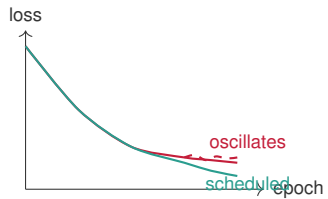
SGD with momentum sometimes finds **flatter minima** that generalize better.
Adam can converge to **sharp minima** — great training loss, worse test accuracy.

Active research area (Wilson et al., 2017).

Why Schedule the Learning Rate?

Even Adam benefits from scheduling:

- **Early training:** large lr to explore broadly
- **Late training:** small lr to settle precisely
- Constant lr **overshoots** the minimum in later epochs



Analogy

Searching for your car in a parking lot:

- First: walk fast, scan rows (high lr)
- Then: slow down, check each space (low lr)

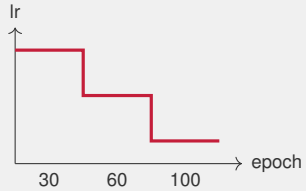
Learning Rate Schedules

Step Decay

Drop lr by $\times 0.1$ at fixed epochs.

Pro: Simple

Con: Must choose *when* to decay

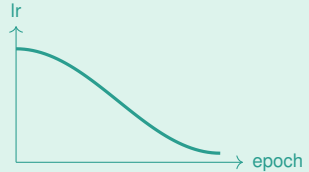


Cosine Annealing

Smooth half-cosine decay.

Pro: No extra hyperparameters, smooth

Use: ResNets, Transformers



Warmup + Decay

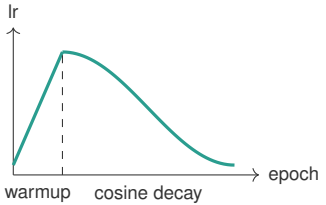
Start small, ramp up, then decay:

Why Warmup?

Early gradients are **unreliable**:
random weights $\rightarrow$ random directions.

Large lr amplifies noise.

Warmup lets the model stabilize before taking big steps.



Typical warmup: 1–5% of total training.

Critical for:

- Transformers (BERT, GPT)
- Large batch training

The Practical Recipe

Classification? $\rightarrow$ Cross-Entropy

Regression? $\rightarrow$ MSE (or Huber)

Optimizer: Adam, $\eta = 0.001$

Schedule: Cosine annealing

Batch size: Largest power of 2 on GPU

Training stalls? $\rightarrow$ Reduce lr

Training diverges? $\rightarrow$ Reduce lr

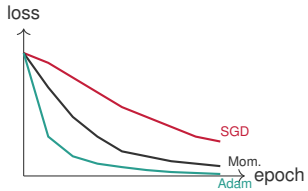
Underfitting? $\rightarrow$ Bigger model

Overfitting? $\rightarrow$ **Session 5!**

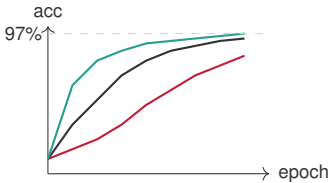
Case Study: Optimizer Showdown on MNIST

Architecture: 784 $\rightarrow$ 128 $\rightarrow$ 64 $\rightarrow$ 10 (from Session 2)

Training Loss



Test Accuracy



Takeaway

Adam converges **fastest**. All three reach $\sim$97% accuracy.
On simple problems, optimizer affects **speed** more than final performance.

5 Key Takeaways

1. **Loss function = what to optimize.** MSE for regression, cross-entropy for classification.
2. **Cross-entropy + sigmoid/softmax = clean gradients.** MSE + sigmoid = vanishing gradients.
3. **Mini-batch SGD** balances speed and stability. **Momentum** smooths the path.
4. **Adam** adapts per-parameter learning rates — the best default. Start with $\eta = 0.001$.
5. **Learning rate schedules** (cosine annealing, warmup) improve convergence: start large, end small.

Final Quiz

1. Name two loss functions and when to use each.
2. Why does cross-entropy work better than MSE for classification with sigmoid?
3. What problem does **momentum** solve?
4. What is Adam's key advantage over vanilla SGD?
5. Why do we **decrease** the learning rate during training?

Final Quiz — Answers

1. **MSE** for regression, **cross-entropy** for classification.
2. The $\sigma'(z)$ term **cancels** in cross-entropy $\rightarrow$ gradient = $\hat{y} - y$.
   With MSE, $\sigma'(z)$ remains $\rightarrow$ gradients vanish at saturation.
3. Momentum **dampens oscillations** across narrow dimensions and **accelerates** progress along consistent directions.
4. Adam adapts **per-parameter learning rates** + builds velocity + handles cold start. Works out of the box.
5. Early: large lr to **explore** broadly. Late: small lr to **settle** precisely. Constant lr overshoots.

Thank You!

Questions?

Priyansh Saxena

All lecture materials available at:

www.priyanshsaxena.com/sst-deep-learning-nn