Learning Rate Decay

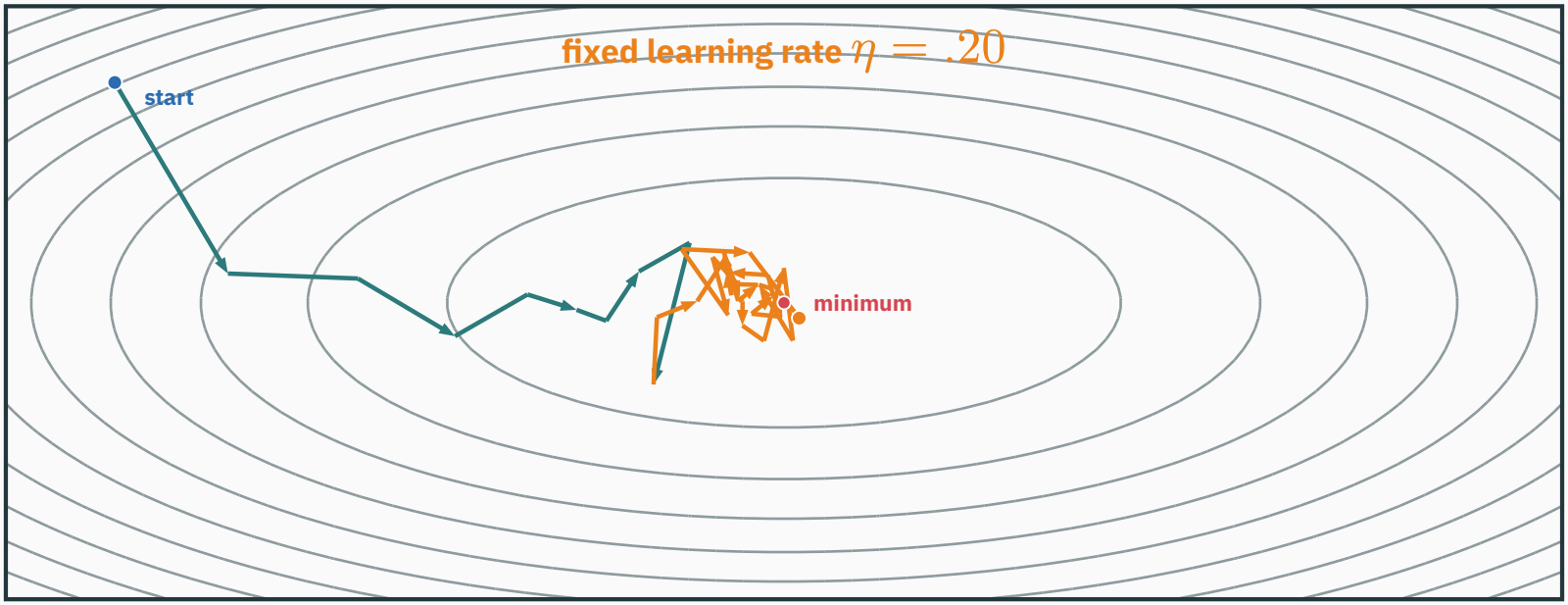
Should epoch 1 and epoch 100 use the same learning rate? · Lecture 5c

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

What changes as SGD gets closer?

Q QUESTION



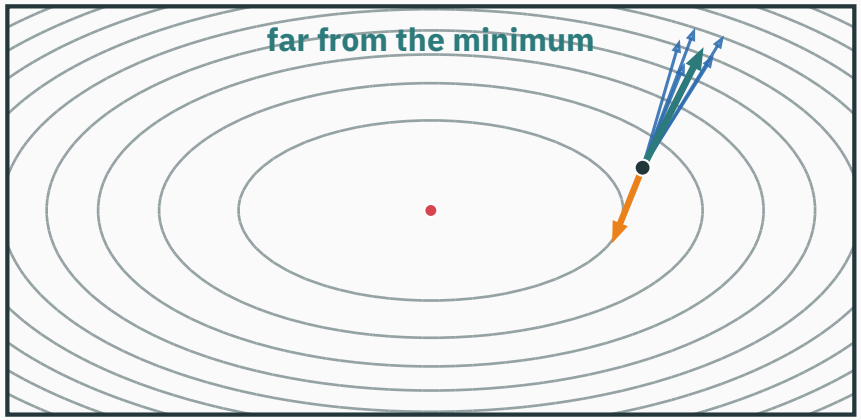
early updates: fast progress

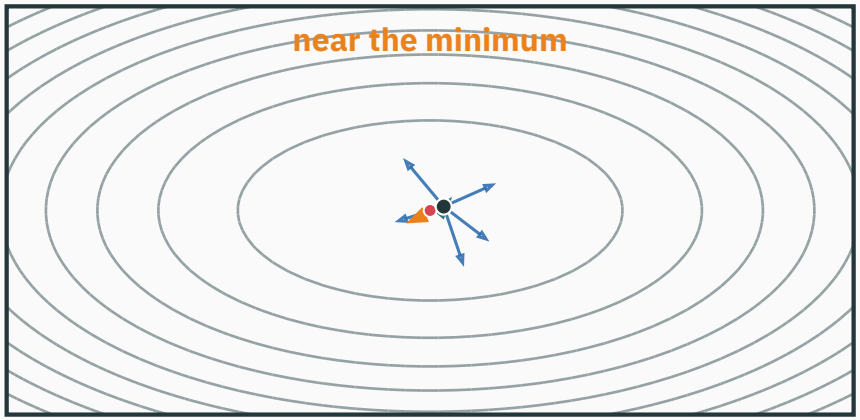
later updates: motion around the minimum

SGD has reached the right region. The same step size now makes it bounce around.

A minibatch gives a noisy gradient

V VISUAL





full-data gradient minibatch gradients chosen update

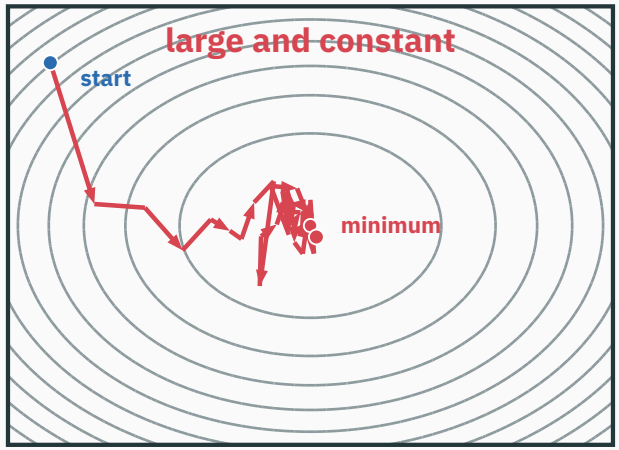
$$g_{B(\theta_t)} = \nabla L(\theta_t) + \varepsilon_B \qquad \theta_{t+1} = \theta_t - \eta_t g_{B(\theta_t)}$$

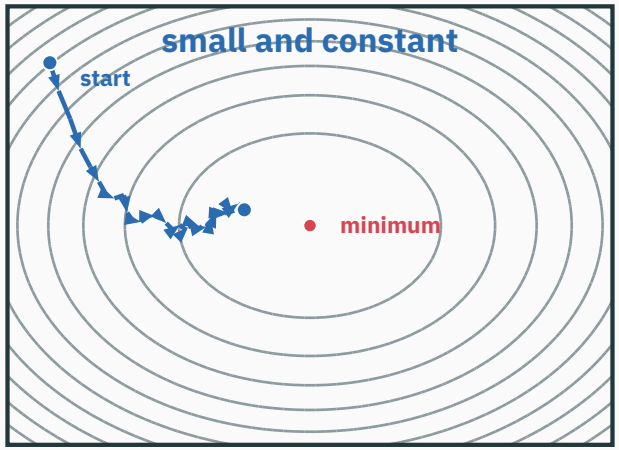
Far away, the gradient signal is large compared with the minibatch noise.

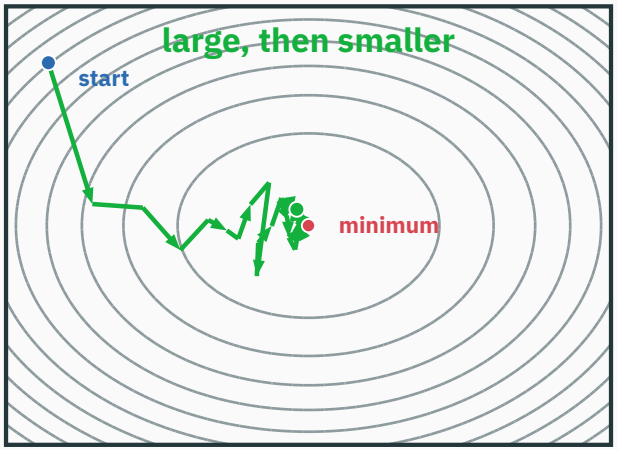
Near the minimum, the gradient signal can be smaller than the minibatch noise.

The learning rate sets the size of the bounce

Q QUESTION







$\eta = .20$: arrives quickly, then jitters

$\eta = .04$: quieter, but slower

$.20 \rightarrow .04$: fast start, quiet finish

For the same noise term $|\varepsilon_B| = .5$:

$\eta = .20 \rightarrow \eta|\varepsilon_B| = .10$ $\eta = .04 \rightarrow \eta|\varepsilon_B| = .02$

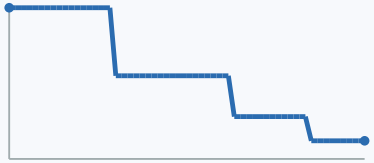
A schedule keeps the useful large steps and reduces the late noise-driven moves.

Four useful decay rules

V VISUAL

Step decay

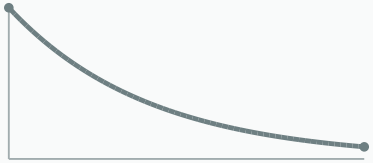
$$\eta_t = \eta_0 \gamma^{\lfloor \frac{t}{K} \rfloor}$$



Use it when: you know the milestones and want drops that are easy to see.

Exponential decay

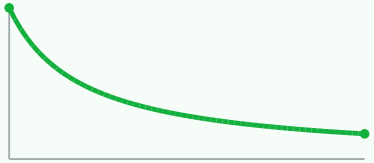
$$\eta_t = \eta_0 \exp(-kt)$$



Use it when: you want a steady percentage decrease with no abrupt jumps.

Inverse-time decay

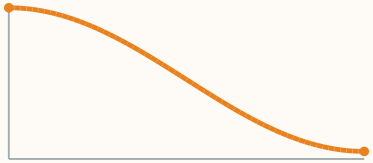
$$\eta_t = \frac{\eta_0}{1+kt}$$



Use it when: you want an early reduction followed by a long, gentle tail.

Cosine decay $(0 \le t \le T)$

$$\eta_t = \eta_{\min} + \frac{\eta_0 - \eta_{\min}}{2}\left(1 + \cos\left(\pi \frac{t}{T}\right)\right)$$



Use it when: training ends at T.

Feedback rule: reduce on plateau. Use it when validation progress, not the clock, should trigger a drop. Patience prevents one noisy check from changing η.

PyTorch: two scheduler patterns

I INTERACTIVE

```
from torch.optim import lr_scheduler as lrs
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
```

FIXED CLOCK: COSINE

```
cosine = lrs.CosineAnnealingLR(
    optimizer,
    T_max=num_epochs,
    eta_min=1e-4,
)

for epoch in range(num_epochs):
    train_one_epoch(...)
    cosine.step()  # after optimizer updates
```

Clock: `T_max` counts calls to `cosine.step()`. Here, one call means one epoch.

VALIDATION FEEDBACK: PLATEAU

```
plateau = lrs.ReduceLROnPlateau(
    optimizer, mode="min",
    factor=0.1, patience=3,
)

for epoch in range(num_epochs):
    train_one_epoch(...)
    val_loss = validate(...)
    plateau.step(val_loss)
```

Metric: `factor=.1` makes each drop 10x smaller. `patience=3` waits before reacting to a short stall.

The call tells you what drives the schedule: `step()` uses time; `step(val_loss)` uses validation.

PyTorch: one epoch, step by step

I INTERACTIVE

```python
from torch.optim import lr_scheduler as lrs

optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
scheduler = lrs.CosineAnnealingLR(
    optimizer,
    T_max=num_epochs,
    eta_min=1e-4,
)

for epoch in range(num_epochs):
    model.train()
    for x, y in train_loader:
        optimizer.zero_grad()          # 1
        y_hat = model(x)               # 2
        loss = loss_fn(y_hat, y)       # 3
        loss.backward()                # 4
        optimizer.step()               # 5

    scheduler.step()                   # 6
```

One call per epoch means `T_max=num_epochs`. If you call the scheduler every batch, its clock must count batches instead.
Timing: `optimizer.step()` moves the model; `scheduler.step()` sets the rate for future moves.

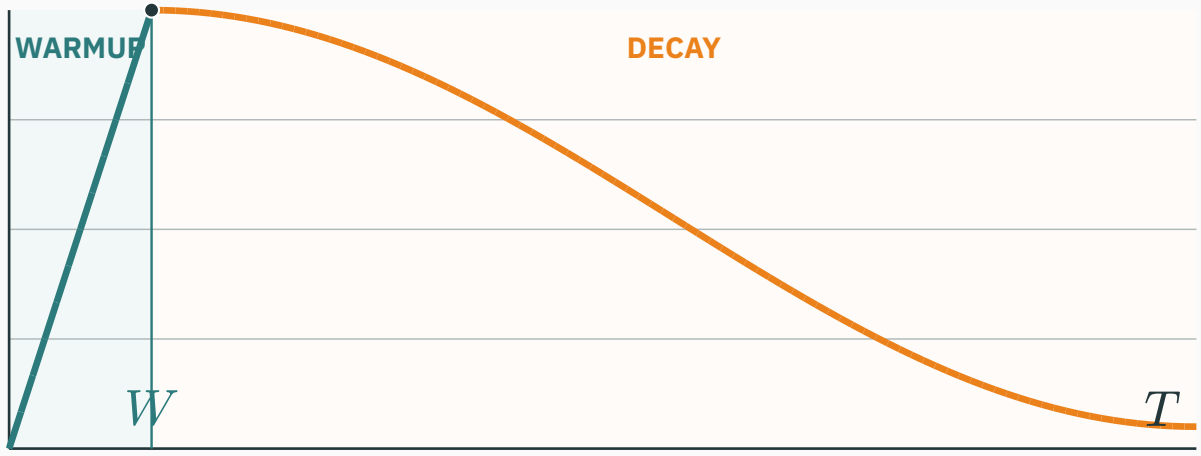
WHAT EACH NUMBER DOES

1. **Clear old gradients.** PyTorch adds gradients unless we reset them.
2. **Predict.** Run the minibatch through the model.
3. **Measure the error.** Compare the prediction with the target.
4. **Backpropagate.** Fill each parameter's `.grad`.
5. **Move the weights.** Use the current learning rate.
6. **Advance the schedule.** Set the rate for the next epoch.

If validation should decide: keep steps 1-5. After validation, use `val_loss = evaluate(model, val_loader)` and then `plateau.step(val_loss)`.

Warm up, then decay

V VISUAL



Example: warm up for the first 12% of updates, then use cosine decay to the finish.

Why it helps: Early gradients can be unusually large. Since $|\Delta\theta_t| = \eta_t |g_t|$, a small η_t keeps the first parameter changes small and reduces the risk of a loss spike or divergence.

1. Start small

$\eta_t = \eta_{\text{peak}} \frac{t}{W}$

Raise the rate gradually over the first W updates.

2. Then decay

$\eta_t \to \eta_{\min}$

Once the early updates are stable, shrink the rate using the chosen rule.

Look for this: the loss spikes or becomes NaN when training starts at the full learning rate.

What should you remember?

Q QUESTION

Far from the minimum

The gradient signal is often strong. A large learning rate can make fast progress.

Near the minimum

Minibatch noise can dominate. A smaller learning rate reduces the resulting motion.

Fixed training budget

Use a time-based schedule such as cosine decay.

Validation should decide

Use reduce on plateau after improvement stalls.

$$g_{B(\theta_t)} = \nabla L(\theta_t) + \varepsilon_B \qquad \Delta\theta_t = -\eta_t g_{B(\theta_t)}$$

Large steps help early. Smaller steps give SGD more control near the minimum.

Choose the simplest rule that matches the run. Add warmup only when startup is fragile.

Sources and evidence

- MIT OCW · Lecture 25: Stochastic Gradient Descent: fast early progress, followed by bouncing near the solution.
- Fabian Pedregosa · The Stochastic Gradient Method: interactive 2D paths for constant and decreasing step sizes.
- Fabian Pedregosa · Gradient Descent: interactive contour plots showing sensitivity to step size.
- Andrew Ng · Learning Rate Decay: one contour picture and one numerical decay example.
- PyTorch · Learning-rate schedulers: scheduler call order and metric-based decay.