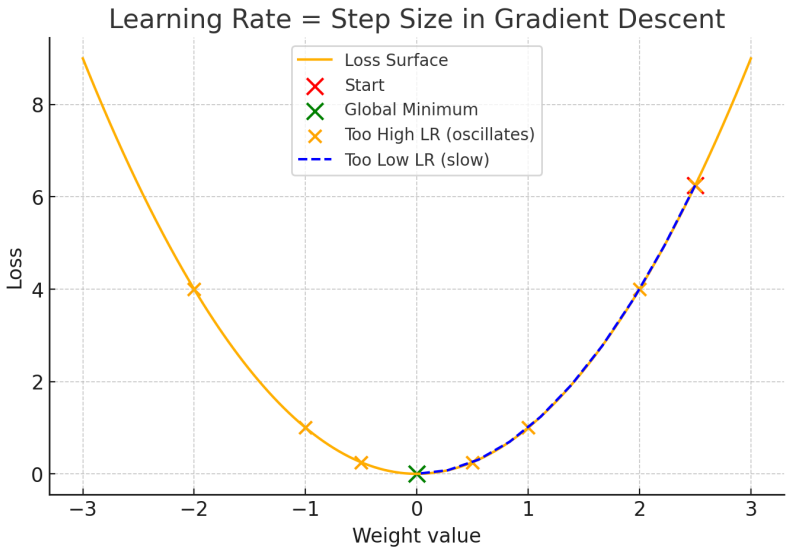
F. Choosing the Learning Rate (α)

Tuesday, 9 September 2025 3:39 PM

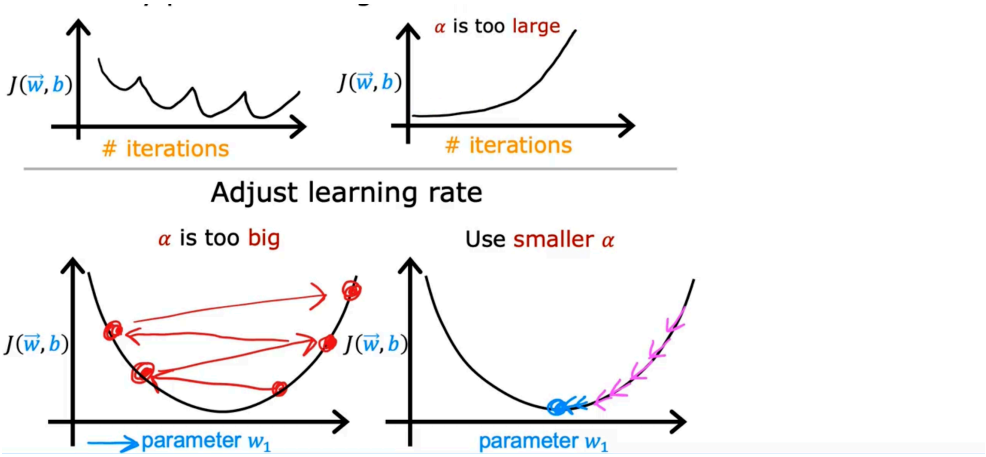
1. Introduction:

What is Learning Rate?

- A **hyperparameter** that controls how much we update model weights after each step of gradient descent.
- Symbol: usually **η** or **α**.
- Think of it as the "step size" when moving downhill on the loss surface.



If we plot a graph of Cost Function (J) vs the no. of iterations, and notice that the cost sometimes goes up and sometimes goes down, that surely means that the Gradient Descent is not working properly. This could mean the α is too large or there's a bug in your code.

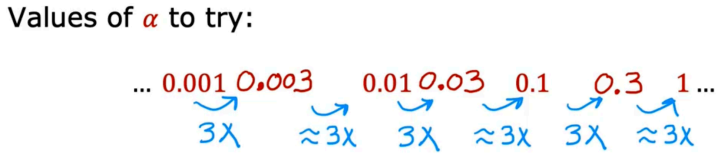


Why It Matters

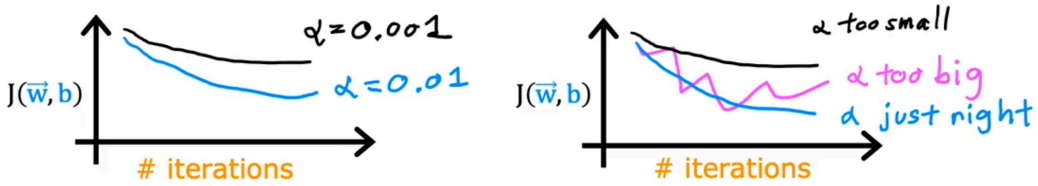
- **Too small** → model learns very slowly, needs many epochs, high compute cost.
- **Too large** → model overshoots minima, oscillates, or diverges completely.

- **Too large** → model overshoots minima, oscillates, or diverges completely. Model may not even converge.
- Good learning rate balances **speed of convergence** with **stability**.

With a small enough α, J(w, b) should decrease on every iteration. So if Gradient Descent isn't working, a debugging step we can use is to set α to be a very small number and see if that causes the cost to decrease on every iteration. Note that setting α to be very small is meant for testing purposes only, since it isn't efficient in practice for training your learning algo.



For each choice of α, we need to use it for a handful no. of iterations and plot the cost function as a function of the no. of iterations. After trying a few different values, you may pick α that decreases the learning rate rapidly, but also consistently. This approach will usually give a good learning rate for your model.



Common Approaches:

1. Fixed Learning Rate

- Same rate for all steps.
- Simple but not adaptive—may be inefficient.

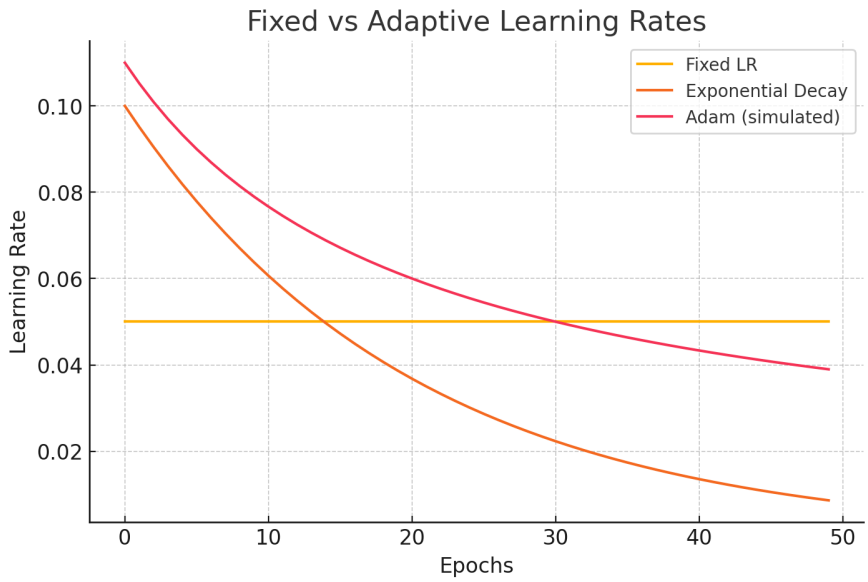
2. Learning Rate Schedules (Decay)

- Start high → gradually reduce.
- Methods:
  - Step decay (drop rate every few epochs).
  - Exponential decay (rate shrinks continuously).
  - Inverse time decay (decays with epoch count).
  - Polynomial decay (slower, smoother drop).

3. Adaptive Methods

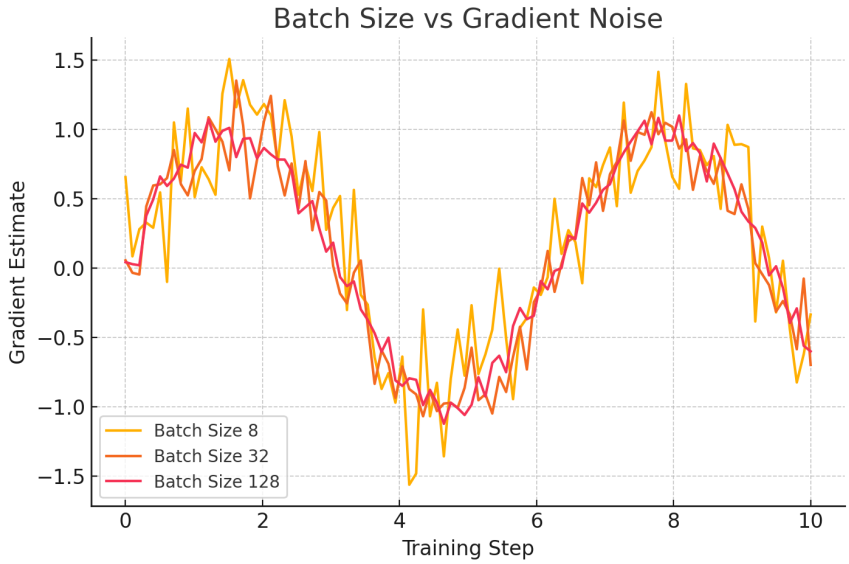
- Adjust automatically during training:
  - **AdaGrad** – per-parameter adjustment, good for sparse data, but decays too quickly.
  - **RMSprop** – balances AdaGrad with smoothing.

- **RMSprop** – balances AdaGrad with smoothing.
- **Adam** – combines momentum + RMSprop, most widely used.



Learning Rate & Batch Size

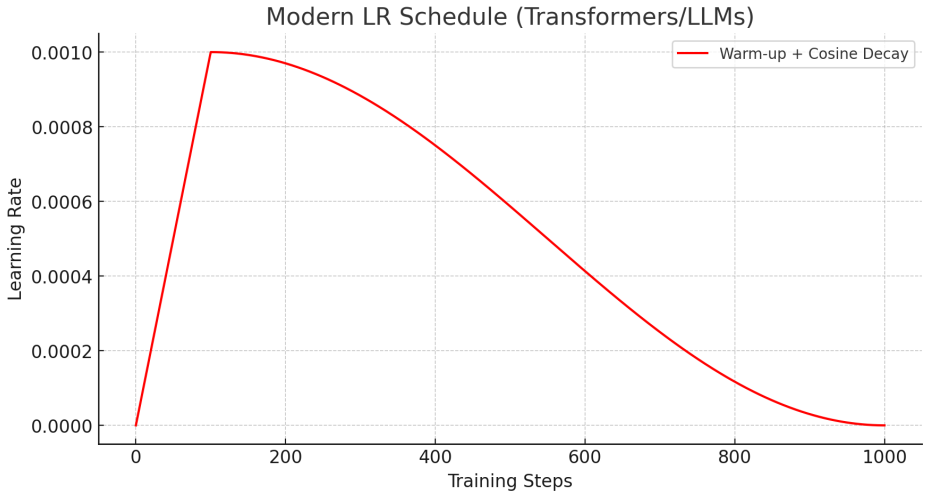
- Small batch → noisier gradient → usually need smaller LR.
- Large batch → smoother gradient → can use bigger LR.
- Rules of thumb:
  - **Linear scaling**: LR ∝ batch size.
  - **Square-root scaling**: LR ∝ √(batch size).

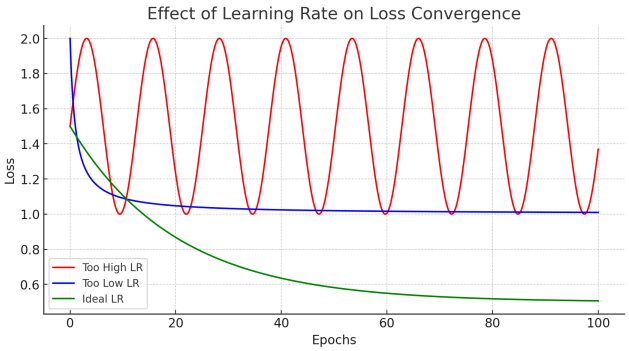


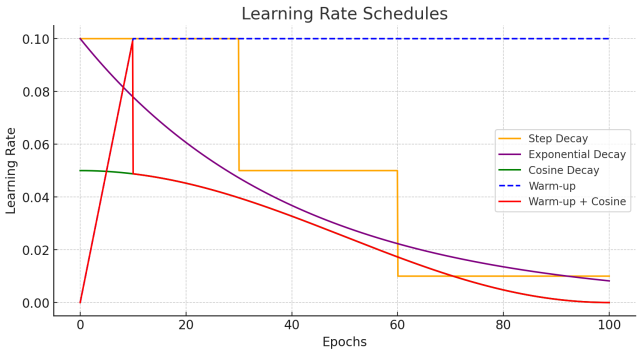
Modern Practice (esp. for LLMs)

- **Warm-up phase**: gradually increase LR from 0 to target over first few steps/epochs → stabilizes training.
- **Cosine decay**: after warm-up, reduce LR smoothly in a half-cosine curve → avoids oscillations, better fine-tuning.

- This combo (warm-up + cosine) is widely used in training transformers and large language models.







Quick Takeaways

- Learning rate = **knob that controls learning speed vs stability**.
- Always tune it—don't just trust defaults.
- Use **schedules** or **adaptive optimizers** for better performance.
- Adjust together with **batch size**.
- For big models → **warm-up + cosine decay** is the current best practice.