MLE Sheet 5

June 2025

Exercise 1: Optimization and Training Tricks

Task 1

Batch Gradient Descent (BGD)

Computes the gradient of the cost function using the entire training dataset.

Update Rule:

$$\theta := \theta - \eta \cdot \nabla J(\theta)$$

where η is the learning rate, and $\nabla J(\theta)$ is the gradient computed on the full dataset.

Advantages:

- Produces stable and accurate gradient estimates.
- Smooth convergence in convex problems.

Disadvantages:

- Computationally expensive for large datasets (each update requires a full pass over the data).
- Slow iteration updates.
- Not suitable for online or streaming data.

Stochastic Gradient Descent (SGD)

Computes the gradient using only one randomly chosen sample at each update step.

Update Rule:

$$\theta := \theta - \eta \cdot \nabla J(\theta; x_i, y_i)$$

Advantages:

- Very fast per update (only one sample needed).
- Can escape local minima due to high variance in updates.

Disadvantages:

- Noisy updates can cause the algorithm to oscillate around the minimum.
- May require more epochs to converge.
- Less stable; harder to choose an optimal learning rate.

Mini-batch SGD

A compromise between BGD and SGD; updates are based on the gradient from a small batch of samples.

Update Rule:

$$\theta := \theta - \eta \cdot \nabla J(\theta; \text{mini-batch})$$

Advantages:

- Faster updates than BGD, more stable than SGD.
- Leverages parallelism (vectorization on GPUs).
- Good balance between noisy updates and computational efficiency.

Disadvantages:

- Still somewhat noisy, depending on batch size.
- Performance depends on mini-batch size (too small: noisy; too big: slow).

Task 2

A

A fixed learning rate may not be ideal throughout the entire training process because:

- In the early stages of training, a high learning rate helps the model make large, rapid improvements.
- In later stages, the same high learning rate can cause the optimization to overshoot the minima or oscillate, preventing convergence.
- Conversely, a learning rate that is too low in the early stages results in slow progress.

Thus, dynamically adjusting the learning rate helps the model transition from rapid exploration to fine-tuning. This can be visualized by a loss landscape where:

- Large steps are useful at the beginning to escape plateaus or poor local minima.
- Small steps are necessary near the optimum to avoid bouncing around.

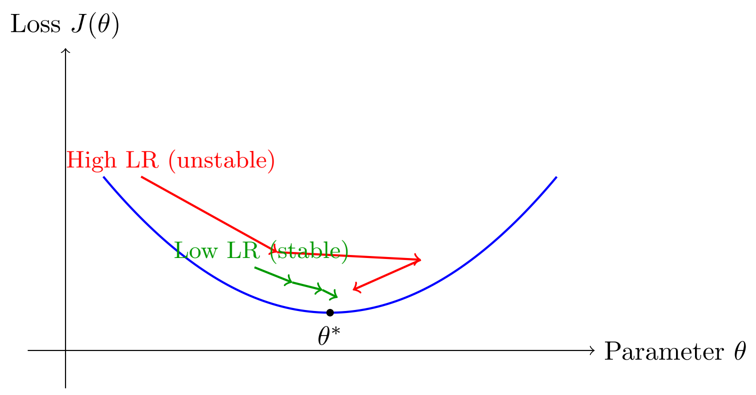

Figure 1: Effect of Learning Rate on Optimization Trajectory

B

A learning rate schedule is a strategy where the learning rate η is adjusted during training based on the epoch number or iteration step. The goal is to improve convergence speed and accuracy by adapting the step size to the training stage.

Example: Exponential Decay

One commonly used schedule is the *exponential decay*, where the learning rate decreases exponentially over time:

$$\eta_t = \eta_0 \cdot e^{-\lambda t}$$

- η_0: initial learning rate
- λ: decay rate
- t: current epoch or iteration number

Effect on Training:

- Initially, the high learning rate allows for fast convergence.
- As training progresses, the reduced learning rate allows the model to fine-tune its parameters and converge to a minimum with more stability.

Other popular schedules include:

- **Step decay**: reduce the learning rate by a factor every few epochs.
- **Cosine annealing**: slowly reduces the learning rate following a cosine curve, possibly with warm restarts.
- **Polynomial decay** and **1/t decay**: use different mathematical functions to decrease the rate.

Learning rate schedules are critical in helping models escape poor local minima early on and then settle efficiently into good minima later in training.