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.

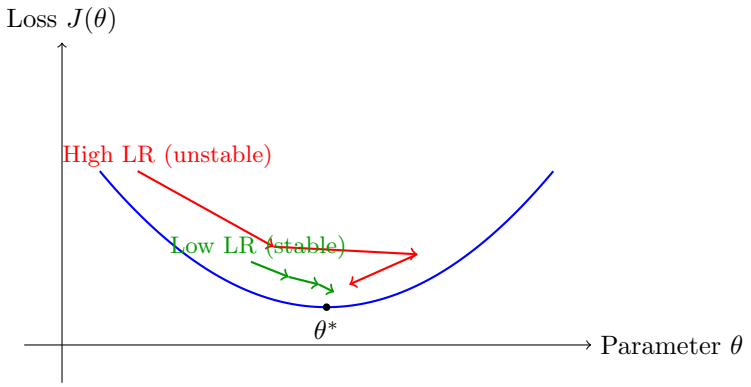

Figure 1: Effect of Learning Rate on Optimization Trajectory

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.

Exercise 1: Tasks 3 and 4

Task 3: Validation and Hyperparameter Tuning

(a) Roles of the datasets

- **Training set:** Used to adjust the model's parameters (weights and biases). The model sees these examples during training and tries to fit them.
- **Validation set:** Used for tuning hyperparameters (e.g., learning rate, batch size) and for model selection. The model doesn't see these examples during training updates but uses them to evaluate performance and adjust hyperparameters iteratively.
- **Test set:** Used to assess the final generalization performance. It should be used only once at the end, not for iterative refinement, to avoid introducing bias in the evaluation.

(b) Why not use the test set for refinement?

If the test set is used to guide decisions (like hyperparameter selection), it effectively becomes part of the training process, leading to an optimistic and biased estimate of the model's true generalization ability. The validation set helps prevent this because it provides an unbiased performance estimate during development. Using the test set repeatedly would underestimate the model's variance (since you're overfitting to that test set), thus leading to misleading performance metrics.

(c) Grid search and the validation set

Concept: Grid search involves defining a set of possible hyperparameter values (a grid) and systematically evaluating all combinations. For example, you might search over different learning rates, batch sizes, or number of layers.

Role of validation set: For each combination of hyperparameters, the model is trained on the training set and evaluated on the validation set. The validation performance guides the choice of the best hyperparameter combination (the one with the highest validation accuracy, for instance).

Task 4: Advanced Optimizers

(a) Core mechanisms

- **i. RMSProp:** RMSProp adapts the learning rate for each parameter by maintaining a moving average of the squared gradients:

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

  The update step for a parameter:

  $$\Delta w_t = -\alpha \frac{g_t}{\sqrt{v_t + \epsilon}}$$

  This helps prevent the learning rate from being too large for parameters with large gradients, stabilizing updates.

- **ii. Momentum:** Momentum accelerates convergence by smoothing the update direction:

  $$m_t = \beta m_{t-1} + (1-\beta) g_t$$

  The parameter update uses this moving average of past gradients:

  $$\Delta w_t = -\alpha m_t$$

  This allows the optimizer to keep moving in relevant directions even when gradients oscillate.

(b) Adam update step

Given:

$$\beta_1 = 0.9, \quad \beta_2 = 0.99, \quad \alpha = 0.01, \quad \epsilon = 10^{-8}$$
$$m_{t-1} = 0.5, \quad v_{t-1} = 0.2, \quad g_t = 2.0$$

1. **Updated first moment:**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t = 0.9 \cdot 0.5 + 0.1 \cdot 2.0 = 0.45 + 0.2 = 0.65$$

2. **Updated second moment:**

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 = 0.99 \cdot 0.2 + 0.01 \cdot (2.0)^2 = 0.198 + 0.04 = 0.238$$

3. **Update step:**

$$\Delta w_t = -\alpha \frac{m_t}{\sqrt{v_t} + \epsilon} = -0.01 \cdot \frac{0.65}{\sqrt{0.238} + 10^{-8}}$$
$$\sqrt{0.238} \approx 0.4879$$
$$\Delta w_t \approx -0.01 \cdot \frac{0.65}{0.4879} = -0.01 \cdot 1.332 \approx -0.01332$$

(c) Effect of larger second moment history (comparing with w')

- $v'_{t-1} = 20$ is much larger than $v_{t-1} = 0.2$.
- Consequently, v'_t will also be much larger than v_t.
- The denominator $\sqrt{v'_t}$ becomes large, so the update step magnitude $|\Delta w'_t|$ will be **smaller** than $|\Delta w_t|$.

Implication: Adam automatically adapts the learning rate for each parameter. Parameters with large past gradients (large v_t) get smaller updates, helping prevent overshooting in areas with large variance. This stabilizes training and improves convergence.