The Adam Optimizer: A Feynman Explanation of the Code

Author: Manus AI **Date:** Jan 12, 2026

Introduction: The Mountain Climber’s Problem

Imagine you are a blindfolded mountain climber trying to find the lowest point in a vast, bumpy valley. You can only feel the slope right where you are standing. This is the job of a machine learning **optimizer** like the one defined in the `MyOptimizer` class: to adjust parameters (`p`) to minimize a loss function (the valley’s depth).

The simplest way to climb down is to take a step in the steepest direction (the negative **gradient**). But this leads to two problems:

1. **Oscillation:** If the valley is steep in one direction and flat in another, you might overshoot and bounce back and forth.
2. **Slow Start:** Your initial steps are often too small or too large, making the process inefficient.

The **Adam** (Adaptive Moment Estimation) optimizer, which this code implements, solves this by using two clever “compasses” to guide the climb.

The Feynman Explanation: The Two Compasses

The core idea of the `MyOptimizer` is to keep track of two things for every parameter: **where we’ve been** and **how bumpy the road is**.

Compass 1: The Momentum Compass (`exp_avg`)

This compass tracks the average of all the past slopes (gradients). It tells us the **general direction** we’ve been heading and how fast we’ve been moving.

- **Simple Analogy:** If you've been walking downhill for a while, this compass gives you **momentum**. It helps you keep moving in that direction, even if you hit a small, temporary uphill bump. This smooths out the path and prevents the "bouncing" problem.
- **Code Mapping:** This is the `exp_avg` variable, the **first moment vector**. It is controlled by the hyperparameter β_1 (`beta1`), which determines how much of the past direction we remember.

Compass 2: The Terrain Compass (`exp_avg_sq`)

This compass tracks the average of the *squared* slopes. It doesn't care about the direction (since we square the slope), but only about the **magnitude** of the slope. It tells us how much the terrain has been changing—how steep or flat the road is.

- **Simple Analogy:** This compass tells you how **shaky the ground is**. If the ground is very steep (high slope magnitude), you should take smaller, more cautious steps. If the ground is flat (low slope magnitude), you can take bigger steps. This makes the steps **adaptive** to the local terrain.
- **Code Mapping:** This is the `exp_avg_sq` variable, the **second moment vector**. It is controlled by the hyperparameter β_2 (`beta2`), which determines how much of the past terrain information we remember.

Code Breakdown: The Four Steps of the Climb

The `step` method of the optimizer executes four main actions for every parameter (`p`) in the model.

1. State Initialization: The Logbook

```
if len(state) == 0:
    state["step"] = 0
    state["exp_avg"] = torch.zeros_like(p)
    state["exp_avg_sq"] = torch.zeros_like(p)
```

Before the first step, the optimizer needs a **logbook** (`state`) to record its history. It initializes the step counter to zero and sets both compasses (`exp_avg` and `exp_avg_sq`) to zero.

2. Moving Averages: Reading the Compasses

```
exp_avg.mul_(beta1).add_(grad, alpha=1.0 - beta1)
exp_avg_sq.mul_(beta2).addcmul_(grad, grad, value=1.0 - beta2)
```

In this step, we update the two compasses based on the current slope (`grad`).

- **Momentum Compass:** We keep a fraction (β_1) of the old momentum and add a fraction ($1 - \beta_1$) of the new slope.
- **Terrain Compass:** We keep a fraction (β_2) of the old terrain information and add a fraction ($1 - \beta_2$) of the new slope **squared**.

3. Bias Correction: The “Starting Slow” Problem

```
bias_correction1 = 1.0 - (beta1 ** step)
bias_correction2 = 1.0 - (beta2 ** step)
```

Since both compasses start at zero, their initial readings are artificially small. This is the “starting slow” problem.

- **The Fix:** We calculate a **correction factor** based on the current step number. As the number of steps (`step`) increases, the term (β^{step}) approaches zero, and the correction factor approaches 1.0. This means the correction is large at the start and fades away as the compasses accumulate enough history.

4. Parameter Update: Taking the Adaptive Step

```
step_size = lr * (math.sqrt(bias_correction2) / bias_correction1)
denom = exp_avg_sq.sqrt().add_(eps)
p.addcdiv_(exp_avg, denom, value=-step_size)
```

This is the final calculation of **how far** and **in what direction** to step.

Component	**Role in the Step**
$\frac{1}{\text{bias\_correction1}}$	Scales up the step size because the Momentum Compass is still under-reporting the true average direction.
$\sqrt{\text{bias\_correction2}}$	Scales down the step size because the Terrain Compass is still under-reporting the true average magnitude.
exp_avg	The **direction** to move (from the Momentum Compass).
$\sqrt{\text{exp\_avg\_sq}} + \epsilon$	The **denominator** that normalizes the step. This is the Terrain Compass reading. If the terrain is bumpy (large $\sqrt{\text{exp\_avg\_sq}}$), the denominator is large, and the step is small. If the terrain is flat, the step is larger.

The final update is: $\$\text{Parameter} \leftarrow \text{Parameter} - \text{Learning Rate} \times \frac{\text{Momentum Compass (Corrected)}}{\text{Terrain Compass (Corrected)}}\$$

This adaptive step ensures the climber moves quickly and smoothly in flat, consistent areas, and slowly and cautiously in steep, bumpy areas, leading to a much faster and more stable descent to the valley floor.

Conclusion

The `MyOptimizer` code is a robust and efficient implementation of the Adam algorithm. By maintaining and correcting two moving averages—one for the average direction (`exp_avg`) and one for the average terrain magnitude (`exp_avg_sq`)—it intelligently adjusts the learning rate for each parameter. This simple yet powerful mechanism is why Adam remains one of the most popular and effective optimizers in deep learning today.