6
More advanced optimization ideas

In Chapter 4 we established convergence of gradient-based methods. For a function f that is L-smooth, Stochastic Gradient Descent (SGD) with a properly chosen learning rate η is guaranteed to find a near-stationary point. Specifically, after T steps, it finds a point w_t such that the expected squared gradient norm is bounded:

$$\min_{t \in \{1,\ldots,T\}} \mathbb{E}[\|\nabla f(w_t)\|_2^2] \leq O\left(\frac{1}{\sqrt{T}}\right)$$

This analysis, while fundamental, rests on two critical simplifications. First, it assumes a single, scalar learning rate η is used for all d parameters. Second, it requires this learning rate to be set according to a *global* smoothness constant L, which is not only difficult to know in practice but also fails to capture the changing local geometry of the loss landscape.

6.1 Beyond the Basic Smoothness Framework

Modern deep learning optimizers derive their power by moving beyond these two constraints. This chapter explores the key ideas that underpin their success:

1. **Adaptive, Per-Parameter Learning Rates:** Methods like Adam challenge the "one-size-fits-all" learning rate by maintaining separate estimates of the learning rate for each parameter, effectively preconditioning the gradient. This is important since rate of learning (i.e., parameter change) may need to proceed at different rates in context of a vast multilayered networks,

2. **Sophisticated, Time-Varying Schedules:** Rather than using a single fixed learning rate, modern training regimes employ carefully designed schedules that modulate the learning rate over time to navigate the distinct phases of optimization. Sometimes these

schedules are explicit, but sometimes implicit in how they adjust to parameter changes.

We will begin by analyzing the theory and practice of learning rate schedules before turning to the mechanics of adaptive optimizers like AdamW.

6.2 *The Theory and Practice of Learning Rate Schedules*

A constant learning rate, while simple to analyze, is rarely optimal. Modern training regimes almost universally employ a learning rate *schedule*, which adjusts η over time. These schedules may appear to be a collection of heuristics, but their effectiveness can be understood through a single, unified theoretical framework.

The key is a generalization of the standard SGD convergence proof that can handle any time-varying learning rate sequence. This result reveals a fundamental trade-off that every schedule must navigate.

Lemma 6.2.1 (SGD Convergence with a General Schedule)**.** *Let f be an L-smooth function, bounded below by f^*. Let w_1 be the initial parameters and let $\Delta_f = f(w_1) - f^*$. Assume the stochastic gradients $\boldsymbol{g}_t$ are unbiased with bounded second moment, $\mathbb{E}[\|\boldsymbol{g}_t\|_2^2] \leq \sigma^2$. Running SGD for T steps with a sequence of learning rates $\{\eta_t\}_{t=1}^T$ guarantees that the minimum expected squared gradient norm is bounded by:*

$$\min_{t\in\{1,\ldots,T\}} \mathbb{E}[\|\nabla f(w_t)\|_2^2] \leq \frac{\Delta_f + \frac{L\sigma^2}{2}\sum_{t=1}^T \eta_t^2}{\sum_{t=1}^T \eta_t}$$

Proof. The proof is a straightforward generalization of the standard SGD analysis. We start from the stochastic descent lemma, taking expectations over the randomness at step t:

$$\begin{aligned}\mathbb{E}[f(w_{t+1})] &\leq \mathbb{E}[f(w_t)] - \eta_t\mathbb{E}[\|\nabla f(w_t)\|_2^2] + \frac{L\eta_t^2}{2}\mathbb{E}[\|\boldsymbol{g}_t\|_2^2] \\ &\leq \mathbb{E}[f(w_t)] - \eta_t\mathbb{E}[\|\nabla f(w_t)\|_2^2] + \frac{L\sigma^2}{2}\eta_t^2\end{aligned}$$

Rearranging to isolate the gradient term and summing from $t = 1$ to T:

$$\sum_{t=1}^T \eta_t\mathbb{E}[\|\nabla f(w_t)\|_2^2] \leq \sum_{t=1}^T(\mathbb{E}[f(w_t)] - \mathbb{E}[f(w_{t+1})]) + \frac{L\sigma^2}{2}\sum_{t=1}^T \eta_t^2$$

The first term on the right is a telescoping sum bounded by $\Delta_f = f(w_1) - f^*$. The sum on the left is a weighted average of squared gradients, which must be at least the minimum value times the sum of the weights:

$$\left(\min_t \mathbb{E}[\|\nabla f(w_t)\|_2^2]\right)\left(\sum_{t=1}^T \eta_t\right) \leq \sum_{t=1}^T \eta_t\mathbb{E}[\|\nabla f(w_t)\|_2^2]$$

Combining these inequalities gives the desired result. □

This lemma reveals the essential trade-off. To make the convergence bound tight, a schedule must manage two competing objectives:

1. **Maximize Progress:** Maximize the denominator, $\sum_{t=1}^{T} \eta_t$. This term represents the total progress made against the initial objective gap Δ_f.

2. **Minimize Variance Penalty:** Minimize the numerator's accumulated noise term, $\frac{L\sigma^2}{2} \sum_{t=1}^{T} \eta_t^2$. This term represents the total variance injected by the stochastic updates.

A good schedule manages this balance. We can now analyze common scheduling practices through this lens.

6.2.1 Time-Based Schedules

The simplest schedules are pre-defined functions of the iteration count t.

LR Warmup. At the start of training, the randomly initialized network is often in a chaotic state with a very large local smoothness constant, L_0. The lemma shows why starting with a large target learning rate η_f is dangerous: the variance term $\frac{L_0\sigma^2}{2}\eta_f^2$ would be large, likely causing divergence. Warmup is a direct solution. It starts with a small learning rate and gradually increases it, ensuring the initial η_t^2 terms are tiny. This keeps the variance penalty in check while the optimizer moves to a more stable region of the loss landscape.

LR Decay and the Cosine Schedule. As training progresses, the true gradient shrinks while stochastic noise remains. A constant learning rate η causes the optimizer to bounce around the minimum in a "noise ball" of radius proportional to η. To achieve convergence, the learning rate must decay.

The popular **cosine schedule** is an effective decay strategy.

$$\eta_t = \frac{\eta_0}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right), \quad \text{for } t = 1, \dots, T$$

Its utility can be understood through the lemma. Early in training, $\eta_t \approx \eta_0$, which maximizes the early terms in $\sum_{t=1}^{T} \eta_t$ for rapid progress. Late in training, η_t decays quickly to zero, making the later terms in $\sum_{t=1}^{T} \eta_t^2$ small, which dampens noise and allows the optimizer to settle.

Multi-Stage Schedules. The training process is not uniform. A single, monotonic decay may be insufficient to escape suboptimal regions or plateaus. Multi-stage schedules divide training into several phases. Within each stage, the learning rate decays, but at the boundary between stages, it is reset to a higher value before decaying again.

This approach can be viewed as inducing periodic exploration. The decaying LR within a stage allows the model to converge within a basin (exploitation). The sudden increase in LR at the start of a new stage provides a perturbation, allowing the optimizer to escape the current basin and explore the landscape for a better region (exploration). From the perspective of our lemma, the high-LR phases boost the progress term $\sum_{t=1}^{T} \eta_t$, while the subsequent low-LR phases help control the overall variance penalty $\sum_{t=1}^{T} \eta_t^2$.

6.2.2 Landscape-Aware Schedules

More advanced schedules are not pre-defined, but adapt in real-time to the geometry of the loss landscape. The "River" method formalizes this by making the learning rate inversely proportional to the measured local "roughness" of the training loss.[1] This roughness is quantified by the variance of the loss across individual examples in a mini-batch.

[1] A River down the Valley: A Method for Learning Rate Schedule Inspired by Nature. A. Windle et al. arXiv:2410.05192, 2024.

The learning rate at step t is the product of a global decay schedule, η_t^{global}, and an adaptive multiplier, m_t. The multiplier is computed by comparing a smoothed, short-term estimate of the loss variance (the "current roughness") to a long-term average (the "target roughness"):

$$m_t = \frac{v_{\text{target}}}{v_t + \epsilon}$$

This mechanism connects directly to our theoretical framework.

- When the landscape is "rough," the mini-batch loss variance v_t is high. This signals that the local smoothness L may be large and the stochastic noise σ^2 is high. The method automatically reduces the learning rate by making $m_t < 1$, suppressing the variance penalty term $\sum_{t=1}^{T} \eta_t^2$ precisely when it is most dangerous.

- When the landscape is "smooth" (e.g., a plateau), the loss variance v_t is low. The method increases the learning rate by making $m_t > 1$. This boosts the progress term $\sum_{t=1}^{T} \eta_t$ to move faster across stable regions.

This landscape-aware approach thus replaces time-based heuristics with a responsive, measurement-based control system that directly manages the trade-off in the convergence bound.

REPLACE

6.3 Adaptive Optimizers: The Adam Family

The schedules discussed so far modify a single learning rate for all parameters over time. Adaptive methods take this a step further by computing an individual, time-varying learning rate for each parameter. This is a form of preconditioning, where the update for each parameter is normalized by an estimate of its historical gradient magnitudes. The prominent example is Adam.

6.3.1 Adam: Adaptive Moment Estimation

Adam, proposed by Kingma and Ba, combines the ideas of momentum (a moving average of the first moment of the gradient) and adaptive learning rates (based on a moving average of the second moment).[2] It maintains two state vectors: $\boldsymbol{m}_t$ for the first moment and $\boldsymbol{v}_t$ for the second moment.

[2] ADAM: A Method For Stochastic Optimization. D. Kingma and J. Ba. ICLR 2015

The algorithm proceeds as follows:

$$\begin{aligned} \boldsymbol{g}_t &= \nabla f_t(w_t) \\ \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t && \text{(First moment estimate)} \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)(\boldsymbol{g}_t \odot \boldsymbol{g}_t) && \text{(Second moment estimate)} \end{aligned}$$

Here, β_1 and β_2 are decay rates, typically close to 1. Because $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ are initialized as zero vectors, they are biased towards zero, especially during the initial steps. The authors introduce bias-corrected estimates:

$$\begin{aligned} \widehat{\boldsymbol{m}}_t &= \frac{\boldsymbol{m}_t}{1-\beta_1^t} \\ \widehat{\boldsymbol{v}}_t &= \frac{\boldsymbol{v}_t}{1-\beta_2^t} \end{aligned}$$

The final parameter update uses these corrected estimates. The update direction is determined by the momentum $\widehat{\boldsymbol{m}}_t$, and the step size is scaled element-wise by the square root of the second moment estimate $\widehat{\boldsymbol{v}}_t$.

$$w_{t+1} = w_t - \eta \frac{\widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{v}}_t} + \epsilon}$$

The term $\sqrt{\widehat{\boldsymbol{v}}_t}$ provides a per-parameter learning rate. Parameters with historically large gradients receive smaller updates, while those with small gradients receive larger updates.

6.3.2 AdamW: Decoupling Weight Decay from Adaptation

A common issue with the standard Adam algorithm arises from its interaction with L2 regularization. An insight formalized by

Loshchilov and Hutter is that for adaptive optimizers, **L2 regularization is not equivalent to weight decay**.[3]

[3] Decoupled Weight Decay Regularization. I. Loshchilov and F. Hutter. ICLR 2019.

In SGD, adding an L2 regularization term $\frac{\lambda}{2}\|w\|_2^2$ to the loss results in an update rule that uniformly shrinks the parameters toward zero at each step:

$$w_{t+1} = (1 - \eta\lambda)w_t - \eta\nabla f(w_t)$$

The term $(1 - \eta\lambda)$ is a "weight decay," applied independently of the gradient.

In standard Adam, the L2 regularization gradient, λw_t, is simply included in g_t. This couples the regularization effect to the adaptive moment estimates. The second moment term v_t now mixes gradient statistics with the magnitude of the parameters themselves. The effective weight decay is suppressed for parameters with large, frequent gradients—often the opposite of the desired behavior.

AdamW decouples the two operations. The optimizer computes the adaptive step using only the loss function's gradient, $\nabla f(w_t)$. Then, weight decay is applied directly as a separate step.

1. Adam step (no L2 term): $\Delta w_t = \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
2. Weight decay step: $w_{t+1} = (1 - \eta'\lambda)w_t - \Delta w_t$

Here η' is the learning rate from the schedule. This restores weight decay to its intended behavior: a predictable shrinkage of weights, independent of gradient statistics. This small change often leads to improved generalization and has made AdamW a common default optimizer.

6.3.3 *The Challenge of Proving Adam's Convergence*

While empirically powerful, the original Adam algorithm is not guaranteed to converge, even for simple convex problems. A key paper by Reddi et al.[4] constructed a counterexample and identified the theoretical flaw.

[4] On the Convergence of Adam and Beyond. S. J. Reddi, S. Kale, and S. Kumar. ICLR 2018.

The issue stems from the short-term memory of the second moment estimate, v_t. A history of large gradients can inflate v_t, which may then excessively shrink the update when a subsequent, smaller gradient provides more useful information. The history of gradients can provide a poor scaling factor for the present, leading to non-convergence.

To remedy this, Reddi et al. proposed AMSGrad. The sole change is to ensure the second moment estimate used for normalization is non-decreasing. This gives the algorithm a "long-term memory" of the largest gradients seen.

The AMSGrad update modifies the v_t calculation:

$$\begin{aligned} \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)(\boldsymbol{g}_t \odot \boldsymbol{g}_t) \\ \widehat{\boldsymbol{v}}_t &= \max(\widehat{\boldsymbol{v}}_{t-1}, \boldsymbol{v}_t) \quad \longleftarrow \text{The only change} \\ w_{t+1} &= w_t - \frac{\eta}{\sqrt{\widehat{\boldsymbol{v}}_t} + \epsilon} \odot \boldsymbol{m}_t \end{aligned}$$

By enforcing monotonicity on $\widehat{v}_t$, one can provide a rigorous proof of convergence in the online convex optimization setting. While AMSGrad fixed the theoretical convergence issue, it has not replaced Adam in practice, as it is often found to perform similarly or slightly worse empirically. This highlights a common gap between optimization theory and deep learning practice.

While our primary focus is non-convex optimization, the convergence issue with Adam is most clearly demonstrated in the simpler online convex setting. The standard analysis in this area is not to show that the gradient vanishes, but to bound the **regret**, which measures the algorithm's performance against the best single fixed parameter vector w^* in hindsight. The following theorem and proof sketch are included to illustrate the analytical technique and to pinpoint the instability in the original Adam algorithm.

Theorem 6.3.1 (Regret Bound for AMSGrad)**.** *Let $f_1, \dots, f_T$ be a sequence of convex loss functions. Assume the domain of parameters $\mathcal{D}$ has diameter D_∞, and that for any $w \in \mathcal{D}$ and any t, the infinity norm of the gradient is bounded, $\|\boldsymbol{g}_t\|_\infty \le G_\infty$. For an appropriate choice of parameters, the AMSGrad algorithm satisfies:*

$$\sum_{t=1}^{T} f_t(w_t) - \sum_{t=1}^{T} f_t(w^*) \le O(D_\infty G_\infty \sqrt{dT \log T})$$

where d is the number of parameters. This implies that the average regret converges to 0 as $T \to \infty$.

Proof Sketch. The core ideas of the proof are as follows.

1. **Start with Convexity:** For any convex function f_t, we have the standard inequality:

$$f_t(w_t) - f_t(w^*) \le \boldsymbol{g}_t^\top (w_t - w^*)$$

2. **Analyze the Update Step:** Let $\boldsymbol{\delta}_t = \frac{\eta}{\sqrt{\widehat{\boldsymbol{v}}_{t+\epsilon}}} \odot \boldsymbol{m}_t$ be the update. We can relate the distance to the optimum at step $t+1$ to the distance at step t:

$$\|w_{t+1} - w^*\|_2^2 = \|w_t - \boldsymbol{\delta}_t - w^*\|_2^2 = \|w_t - w^*\|_2^2 - 2\boldsymbol{\delta}_t^\top (w_t - w^*) + \|\boldsymbol{\delta}_t\|_2^2$$

3. **Bound the Regret:** Rearranging the previous expression and summing over T allows us to bound the regret. The term $\|w_t - w^*\|_2^2 - \|w_{t+1} - w^*\|_2^2$ creates a telescoping sum, which is bounded by the diameter of the space. The main challenge lies in carefully bounding the remaining terms involving the momentum $\boldsymbol{m}_t$ and the update step δ_t.

4. **Use the AMSGrad Property:** The critical step where AMSGrad's property is used is in bounding the sum of the effective squared step sizes. Because $\widehat{v}_t$ is non-decreasing by construction, the effective learning rate for each parameter, $\eta_t^{(i)} = \eta/(\sqrt{\widehat{v}_t^{(i)}} + \epsilon)$, is also non-increasing. This property is essential for bounding the sum of update norms, $\sum_{t=1}^T \|\delta_t\|_2^2$, which ultimately leads to the $\sqrt{T}$ term in the regret bound. In standard Adam, because $\boldsymbol{v}_t$ can decrease, this sum cannot be bounded in the same way, which is why the proof fails.

$\square$

OLD STUFF

6.3.4 *Implications for AdamW and Non-Convex Optimization*

While AMSGrad fixed the theoretical convergence issue, it has not replaced Adam in practice, as it is often found to perform slightly worse empirically. This highlights a fascinating gap between current theory and practice.

For Adam and AdamW in the general non-convex setting, convergence can be proven, but it requires stronger assumptions, such as the second moment of the stochastic gradients being bounded. Under these assumptions, one can show that $\frac{1}{T}\sum_{t=1}^T \mathbb{E}[\|\nabla f(w_t)\|_2^2] \to 0$, similar to SGD, but the resulting convergence rate is often worse than that of SGD in its dependence on problem parameters.

The analysis for AdamW is largely the same as for Adam. The decoupled weight decay term can be seen as adding a simple convex quadratic term to the objective, $f'(w) = f(w) + \frac{\lambda'}{2}\|w\|_2^2$. The convergence analysis for Adam can then be applied to this modified, strongly-convexified objective. The core challenges related to the adaptive step normalization remain.

6.4 *The Role of the Learning Rate Schedule*

A constant learning rate, while easy to analyze, is rarely optimal. Modern training regimes almost universally employ a learning rate *schedule*, which adjusts the value of η over the course of training. These schedules typically consist of two phases: a brief initial "warmup" period where the LR increases, followed by a long "decay" period where it decreases. While these may seem like empirical heuristics, their effectiveness can be understood through simple models and theoretical analysis.

LR Warmup. At the start of training, the randomly initialized network is often in a chaotic state with a very large local smoothness constant, L_0. The lemma shows why starting with a large target learning rate η_f is dangerous: the variance term $\frac{L_0\sigma^2}{2}\eta_f^2$ would be large, likely causing divergence. Warmup is a direct solution. It starts with a small learning rate and gradually increases it, ensuring the initial η_t^2 terms are tiny. This keeps the variance penalty in check while the optimizer moves to a more stable region of the loss landscape.

We can formalize this requirement. Consider a linear warmup schedule over T_w steps, where the learning rate is $\eta_t = (t/T_w)\eta_f$ for $t \in \{1, \ldots, T_w\}$. The Descent Lemma requires the learning rate to be bounded by the local smoothness. The most stringent condition is at the first step, where the smoothness L_0 is maximal. To ensure stability, we must satisfy:

$$\eta_1 = \frac{1}{T_w}\eta_f < \frac{2}{L_0}$$

This implies a formal condition on the minimum length of the warmup period:

$$T_w > \frac{\eta_f L_0}{2}$$

The duration must be proportional to the target learning rate and the initial smoothness. While L_0 is unknown in practice, this relation provides a theoretical basis for the warmup heuristic, which must be long enough for the optimizer to exit the initial chaotic region.

LR Decay and the Cosine Schedule. As training progresses, the true gradient shrinks while stochastic noise remains. A constant learning rate η causes the optimizer to bounce around the minimum in a "noise ball" of radius proportional to η. To achieve convergence, the learning rate must decay. The popular **cosine schedule** is an effective decay strategy.

$$\eta_t = \frac{\eta_0}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right), \quad \text{for } t = 1, \ldots, T$$

Its utility can be understood through the lemma. Early in training, $\eta_t \approx \eta_0$, which maximizes the early terms in $\sum_{t=1}^{T} \eta_t$ for rapid progress. Late in training, η_t decays quickly to zero, making the later terms in $\sum_{t=1}^{T} \eta_t^2$ small, which dampens noise and allows the optimizer to settle.

We can gain more precise insight by analyzing the schedule on a simple quadratic model, $f(w) = \frac{\lambda}{2}w^2$. The GD update is $w_{t+1} = (1 - \eta_t \lambda)w_t$. Using a Taylor expansion for the cosine term reveals two distinct phases. For early steps where $t \ll T$, the argument of cosine is small, and $\cos(x) \approx 1 - x^2/2$. This gives $\eta_t \approx \eta_0(1 - (t\pi)^2/(4T^2)) \approx \eta_0$. The schedule approximates a large, constant learning rate, leading to rapid geometric convergence. For late steps where $t \to T$, let $t = T - k$ for small k. We find that $\eta_{T-k} \approx \frac{\eta_0 \pi^2}{4}(k/T)^2$. The learning rate decays quadratically to zero, providing a soft landing that allows the iterates to settle precisely at the minimum. The schedule thus smoothly interpolates between an aggressive initial phase and a conservative final phase.

Multi-Stage Schedules. The training process is not uniform. A single, monotonic decay may be insufficient to escape suboptimal regions or plateaus. Multi-stage schedules divide training into several phases. Within each stage, the learning rate decays, but at the boundary between stages, it is reset to a higher value before decaying again. This approach can be viewed as inducing periodic exploration. The decaying LR within a stage allows the model to converge within a basin (exploitation). The sudden increase in LR at the start of a new stage provides a perturbation, allowing the optimizer to escape the current basin and explore the landscape for a better region (exploration). From the perspective of our lemma, the high-LR phases boost the progress term $\sum_{t=1}^{T} \eta_t$, while the subsequent low-LR phases help control the overall variance penalty $\sum_{t=1}^{T} \eta_t^2$.

We can make this trade-off concrete with a numerical example. Assume a training run of $T = 100$ steps, with an initial function gap $\Delta_f = 100$ and a landscape difficulty term $\frac{L\sigma^2}{2} = 1$. The convergence bound from the lemma is:

$$\min_t \mathbb{E}[\|\nabla f(w_t)\|_2^2] \leq \frac{100 + \sum_{t=1}^{100} \eta_t^2}{\sum_{t=1}^{100} \eta_t}$$

Let us compare three strategies based on a peak learning rate of $\eta_{\text{peak}} = 0.1$.

- **Strategy 1: Aggressive Constant LR.** Set $\eta_t = 0.1$ for all steps.
  - $\sum \eta_t = 100 \times 0.1 = 10$.
  - $\sum \eta_t^2 = 100 \times (0.1)^2 = 1$.
  - Bound: $\frac{100+1}{10} = \mathbf{10.1}$.
- **Strategy 2: Conservative Constant LR.** Set $\eta_t = 0.025$ for all steps.

  - $\sum \eta_t = 100 \times 0.025 = 2.5$.
  - $\sum \eta_t^2 = 100 \times (0.025)^2 = 0.0625$.
  - Bound: $\frac{100+0.0625}{2.5} = \mathbf{40.025}$.

- **Strategy 3: Simplified Multi-Stage.** Use the aggressive rate for the first 50 steps and the conservative rate for the second 50 steps.
  - $\sum \eta_t = (50 \times 0.1) + (50 \times 0.025) = 6.25$.
  - $\sum \eta_t^2 = (50 \times 0.1^2) + (50 \times 0.025^2) \approx 0.531$.
  - Bound: $\frac{100+0.531}{6.25} \approx \mathbf{16.085}$.

The multi-stage schedule manages the trade-off effectively. It makes significantly more progress than the conservative schedule (denominator is 6.25 vs 2.5) while accumulating only about half the variance penalty of the aggressive schedule (numerator term is 0.531 vs 1.0). The initial high-LR phase drives down the large initial error, while the subsequent low-LR phase allows for stable final convergence.
OLD SUBSECTION

6.5 *Landscape-Aware Schedules: Formalizing the River Method*

The schedules discussed so far, from cosine to WSD-LR, are **agnostic**. They are pre-defined functions of time, blind to the actual dynamics of the training process. A more powerful approach is to have the learning rate adapt in real-time to the geometry of the loss landscape.

The "River" method formalizes this by making the learning rate inversely proportional to the measured local "roughness" of the training loss.[5] This roughness is quantified by the variance of the loss across individual examples in a mini-batch. A high variance signals a turbulent, unpredictable landscape, while a low variance suggests a smooth, stable basin.

[5] A River down the Valley: A Method for Learning Rate Schedule Inspired by Nature. A. Windle et al. arXiv:2410.05192, 2024.

The Core Mechanism in Four Steps. The final learning rate at step t, η_t^{final}, is the product of a global decay schedule, η_t^{global}, and a new adaptive multiplier, m_t.

$$\eta_t^{\text{final}} = \eta_t^{\text{global}} \times m_t$$

The adaptive multiplier m_t is computed as follows:

Step 1: Measure Instantaneous Roughness. At each step t, for a mini-batch B_t of size N, we first compute the loss $L_t(i)$ for each example $i \in B_t$. The instantaneous roughness is the statistical variance of these loss values.

$$\bar{L}_t = \frac{1}{N} \sum_{i \in B_t} L_t(i) \qquad \text{(Mean mini-batch loss)}$$

$$\text{Var}_t = \frac{1}{N} \sum_{i \in B_t} (L_t(i) - \bar{L}_t)^2 \qquad \text{(Instantaneous roughness)}$$

Step 2: Smooth the Roughness Estimate. The variance from a single mini-batch is a very noisy signal. To obtain a stable estimate of the local landscape roughness, we compute an exponential moving average (EMA) of the instantaneous variance, denoted v_t.

$$v_t = \beta_v v_{t-1} + (1 - \beta_v)\text{Var}_t$$

Here, β_v is a new hyperparameter, typically close to 1 (e.g., 0.99), that controls the memory of the roughness estimate.

Step 3: Convert Roughness to a Multiplier. The core inverse relationship is implemented by comparing the current smoothed roughness v_t to a target roughness v_{target}. This target value is itself a long-term, slow-moving EMA of v_t, which represents the "average" roughness of the landscape encountered so far.

$$m_t = \frac{v_{\text{target}}}{v_t + \epsilon}$$

where ϵ is a small constant for numerical stability. In practice, this multiplier is often clipped to a pre-defined range, e.g., $[m_{\text{min}}, m_{\text{max}}]$, to prevent extreme values.

Step 4: Modulate the Global LR. The final learning rate is then computed by modulating the pre-defined global schedule (e.g., cosine) with this adaptive multiplier.

A Formal Explanation for Emergent Behaviors. This set of formulas provides a concrete mechanism for the behaviors we previously described with the river metaphor.

- **Automatic Warmup:** At $t = 0$, the network is chaotic. The initial mini-batch loss variance, Var_0, is typically very high. This makes the smoothed roughness v_t large. Since v_t is much larger than the slowly-updating v_{target}, the multiplier m_t is automatically driven to a small value, effectively suppressing the learning rate. This is a natural, data-driven warmup.

- **Plateau Acceleration:** If the optimizer enters a wide, flat plateau, the loss becomes very consistent across examples in a mini-batch. Var_t drops, causing v_t to fall below v_{target}. The multiplier m_t then becomes greater than 1, automatically increasing the learning rate to accelerate across the stable region.

This landscape-aware approach thus replaces manual, time-based heuristics with a responsive, measurement-based control system, representing a significant step towards more autonomous and intelligent optimization algorithms.