TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2018

Stochastic Gradient Descent (SGD)

The Classical Convergence Thoerem

RMSProp, Momentum and Adam

Scaling Learnng Rates with Batch Size

SGD as MCMC and MCMC as SGD

An Original SGD Algorithm

Vanilla SGD

$$\Phi \texttt{ -= } \eta \hat{g}$$

$$\hat{g} = E_{(x,y)\sim\text{Batch}} \nabla_\Phi \text{loss}(\Phi, x, y)$$

$$g = E_{(x,y)\sim\text{Train}} \nabla_\Phi \text{loss}(\Phi, x, y)$$

For theoretical analysis we will focus on the case where the training data is very large — essentially infinite.

Issues

- **Gradient Estimation.** The accuracy of $\hat{g}$ as an estimate of g.

- **Gradient Drift (second order structure).** The fact that g changes as the parameters change.

A One Dimensional Example

Suppose that y is a scalar, and consider

$$\text{loss}(\beta, x, y) = \frac{1}{2}(\beta - y)^2$$

$$g = E_{(x,y)\sim\text{Train}}\, d\,\text{loss}(\beta, x, y)/d\beta = \beta - E_{\text{Train}}[y]$$

$$\hat{g} = E_{(x,y)\sim\text{Batch}}\, d\,\text{loss}(\beta, x, y)/d\beta = \beta - E_{\text{Batch}}[y]$$

SGD as MCMC — The SGD Stationary Distribution

For small batches we have that each step of SGD makes a random move in parameter space.

Even if we start at the training loss optimum, an SGD step will move away from the optimum.

SGD defines an MCMC process with a stationary distribution.

To converge to a local optimum the learning rate must be gradually reduced to zero.

The Classical Convergence Theorem

$$\Phi \mathbf{-=} \eta^t \nabla_\Phi \text{ loss}(\Phi, x_t, y_t)$$

For "sufficiently smooth" non-negative loss with

$$\eta^t > 0 \quad \text{and} \quad \lim_{t \to 0} \eta^t = 0 \quad \text{and} \quad \sum_t \eta^t = \infty,$$

we have that the training loss of Φ converges (in practice Φ converges to a local optimum of training loss).

Rigor Police: One can construct cases where Φ converges to a saddle point or even a limit cycle.

See "Neuro-Dynamic Programming" by Bertsekas and Tsitsiklis proposition 3.5.

Physicist's Proof of the Convergence Theorem

Since $\lim_{t \to 0} \eta^t = 0$ we will eventually get to arbitrarilly small learning rates.

For sufficiently small learning rates any meaningful update of the parameters will be based on an arbitrarily large sample of gradients at essentially the same parameter value.

An arbitrarily large sample will become arbitrarily accurate as an estimate of the full gradient.

But since $\sum_t \eta^t = \infty$, no matter how small the learning rate gets, we still can make arbitrarily large motions in parameter space.

Statistical Intuitions for Learning Rates

For intuition consider the one dimensional case.

At a fixed parameter setting we can sample gradients.

Averaging together N sample gradients produces a confidence interval on the true gradient.

$$g = \hat{g} \pm \frac{2\sigma}{\sqrt{N}}$$

To have the right direction of motion this interval should not contain zero. This gives.

$$N \geq \frac{2\sigma^2}{\hat{g}^2}$$

Statistical Intuitions for Learning Rates

$$N \geq \frac{2\sigma^2}{\hat{g}^2}$$

To average N gradients we need that N gradient updates have a limited influence on the gradient.

This suggests

$$\eta^t \propto \frac{1}{N} \propto \frac{(g^t)^2}{(\sigma^t)^2}$$

The constant of proportionality will depend on the rate of change of the gradient (the second derivative of loss).

Statistical Intuitions for Learning Rates

$$\eta^t \propto \frac{(g^t)^2}{(\sigma^t)^2}$$

This is written in terms of the true (average) gradient g^t at time t and the true standard deviation σ^t at time t.

This formulation is of conceptual interest but is not (yet) directly implementable (more later).

As $g^t \to 0$ we expect $\sigma^t \to \sigma > 0$ and hence $\eta^t \to 0$.

Running Averages

We can try to estimate g^t and σ^t with a running average.

It is useful to review general running averages.

Consider a time series x_1, x_2, x_3, ...

Suppose that we want to approximate a running average

$$\hat{\mu}_t \approx \frac{1}{N} \sum_{s=t-N+1}^{t} x_s$$

This can be done efficiently with the update

$$\hat{\mu}_{t+1} = \left(1 - \frac{1}{N}\right) \hat{\mu}_t + \left(\frac{1}{N}\right) x_{t+1}$$

Running Averages

More explicitly, for $\hat{\mu}_0 = 0$, the update

$$\hat{\mu}_{t+1} = \left(1 - \frac{1}{N}\right) \hat{\mu}_t + \left(\frac{1}{N}\right) x_{t+1}$$

gives

$$\hat{\mu}_t = \frac{1}{N} \sum_{1 \leq s \leq t} \left(1 - \frac{1}{N}\right)^{-(t-s)} x_s$$

where we have

$$\sum_{n \geq 0} \left(1 - \frac{1}{N}\right)^{-n} = N$$

Back to Learning Rates

In high dimension we can apply the statistical learning rate argument to each dimension (parameter) $\Phi[c]$ of the parameter vector Φ giving a separate learning rate for each dimension.

$$\eta^t[c] \propto \frac{g^t[c]^2}{\sigma^t[c]^2}$$

$$\Phi^{t+1}[c] = \Phi^t[c] - \eta^t[c]\Phi^t[c]$$

RMSProp

RMS — Root Mean Square — was introduced by Hinton and proved effective in practice. We start by computing a running average of $\hat{g}[c]^2$.

$$s^{t+1}[c] = \beta s^t[c] + (1-\beta)\,\hat{g}[c]^2$$

The PyTorch Default for β is .99 which corresponds to a running average of 100 values of $\hat{g}[c]^2$.

If $g^t[c] << \sigma^t[c]$ then $s^t[c] \approx \sigma^t[c]^2$.

RMSProp:

$$\eta^t[c] \propto 1/\sqrt{s^t[c] + \epsilon}$$

RMSProp

RMSProp

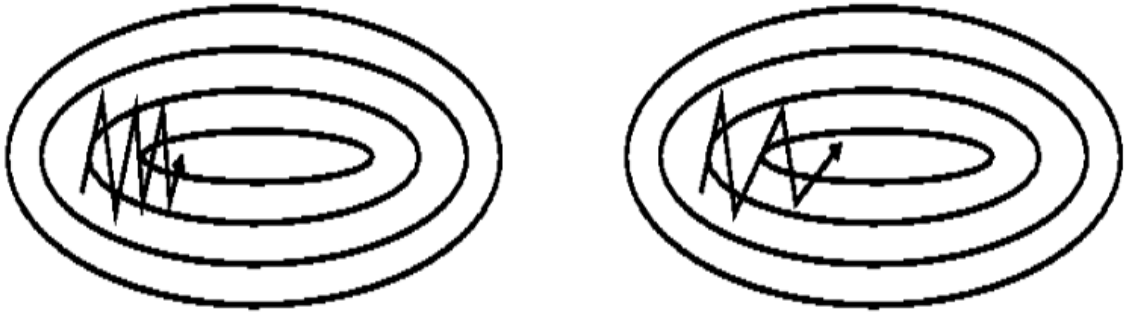
$$\eta^t[c] \propto 1/\sqrt{s^t[c] + \epsilon}$$

bears some similarity to

$$\eta^t[c] \propto g^t[c]^2/\sigma^t[c]^2$$

but there is no attempt to estimate $g^t[c]$.

Momentum



Rudin's blog

The theory of momentum is generally given in terms of **gradient drift** (the second order structure of total training loss).

I will instead analyze momentum as a running average of $\hat{g}$.

Momentum, Nonstandard Parameterization

$$\tilde{g}^{t+1} = \mu\tilde{g}^{t} + (1-\mu)\hat{g} \qquad \mu \in (0,1) \quad \text{Typically } \mu \approx .9$$

$$\Phi^{t+1} = \Phi^{t} - \eta\tilde{g}^{t+1}$$

For $\mu = .9$ we have that $\tilde{g}^t$ approximates a running average of 10 values of $\hat{g}$.

Running Averages Revisited

Consider any sequence y_t derived from x_t by

$$y_{t+1} = \left(1 - \frac{1}{N}\right) y_t + f(x_t) \quad \text{for any function } f$$

We note that any such equation defines a running average of $Nf(x_t)$.

$$y_{t+1} = \left(1 - \frac{1}{N}\right) y_t + \left(\frac{1}{N}\right) (Nf(x_t))$$

Momentum, Standard Parameterization

$$v^{t+1} = \mu v^t + \eta * \hat{g} \qquad \mu \in (0, 1)$$

$$\Phi^{t+1} = \Phi^t - v^{t+1}$$

By the preceding slide v^t is a running average of $(\eta/(1-\mu))\hat{g}$ and hence the above definition is equivalent to

$$\tilde{g}^{t+1} = \mu \tilde{g}^t + (1-\mu)\hat{g} \qquad \mu \in (0, 1)$$

$$\Phi^{t+1} = \Phi^t - \left(\frac{\eta}{1-\mu}\right) \tilde{g}^{t+1}$$

Momentum

$$\tilde{g}^{t+1} = \mu\tilde{g}^{t} + (1-\mu)\hat{g} \qquad \mu \in (0,1), \text{ typically } .9$$

$$\Phi^{t+1} = \Phi^{t} - \left(\frac{\eta}{1-\mu}\right)\tilde{g}^{t+1} \quad \text{standard parameterization}$$

$$\Phi^{t+1} = \Phi^{t} - \eta'\tilde{g}^{t+1} \qquad \text{nonstandard parameterization}$$

The total contribution of a gradient value $\hat{g}^t$ is $\eta/(1-\mu)$ in the standard parameterization and is η' in the nonstandard parameterization (independent of μ). This suggests that the optimal value of η' is independent of μ and that the μ does nothing.

Adam — Adaptive Momentum

$$\tilde{g}^{t+1}[c] = \beta_1 \tilde{g}^t[c] + (1-\beta_1)\hat{g}[c] \quad \text{PyTorch Default: } \beta_1 = .9$$

$$s^{t+1}[c] = \beta_2 s^t[c] + (1-\beta_2)\hat{g}[c]^2 \quad \text{PyTorch Default: } \beta_2 = .999$$

$$\Phi^{t+1}[c] \texttt{ -= } \frac{l_r}{\sqrt{(1-\beta_2^t)s^{t+1}[c]} + \epsilon}\ (1-\beta_1^t)\tilde{g}^{t+1}[c]$$

Given the argument that momentum does nothing, this should be equivalent to RMSProp. However, implementations of RMSProp and Adam differ in details such as default parameter values and, perhaps most importantly, RMSProp lacks the "initial bias correction terms" $(1-\beta^t)$ (see the next slide).

Bias Correction in Adam

Adam takes $\tilde{g}^0 = s^0 = 0$.

For $\beta_2 = .999$ we have that s^t is very small for $t << 1000$.

To make $s^t[c]$ a better average of $g^t[c]^2$ we replace $s^t[c]$ by $(1 - \beta_2^t)s^t[c]$.

For $\beta_2 = .999$ we have $\beta_2^t \approx \exp(-t/1000)$ and for $t >> 1000$ we have $(1 - \beta_2^t) \approx 1$.

Similar comments apply to replacing $g^t[c]$ by $(1 - \beta_1^t)g^t[c]$.

Learning Rate Scaling

Recent work has show that by scaling the learning rate with the batch size very large batch size can lead to very fast (highly parallel) training.

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, Goyal et al., 2017.

Learning Rate Scaling

Consider two consecutive updates for a batch size of 1 with learning rate η_1.

$$\Phi^{t+1} = \Phi^t - \eta_1 \nabla_\Phi \text{loss}(\Phi^t, x^t, y^t)$$

$$\begin{aligned}
\Phi^{t+2} &= \Phi^{t+1} - \eta_1 \nabla_\Phi \text{loss}(\Phi^{t+1}, x^{t+1}, y^{t+1}) \\
&\approx \Phi^{t+1} - \eta_1 \nabla_\Phi \text{loss}(\Phi^t, x^{t+1}, y^{t+1}) \\
&= \Phi^t - \eta_1((\nabla_\Phi \text{loss}(\Phi^t, x^t, y^t)) + (\nabla_\Phi \text{loss}(\Phi^t, x^{t+1}, y^{t+1})))
\end{aligned}$$

Learning Rate Scaling

Let η_B be the learning rate for batch size B.

$$\Phi^{t+2} \approx \Phi^t - \eta_1((\nabla_\Phi \text{loss}(\Phi^t, x^t, y^t)) + (\nabla_\Phi \text{loss}(\Phi^t, x^{t+1}, y^{t+1})))$$

$$= \Phi^t - 2\eta_1 \, \hat{g} \quad \text{for} \quad B = 2$$

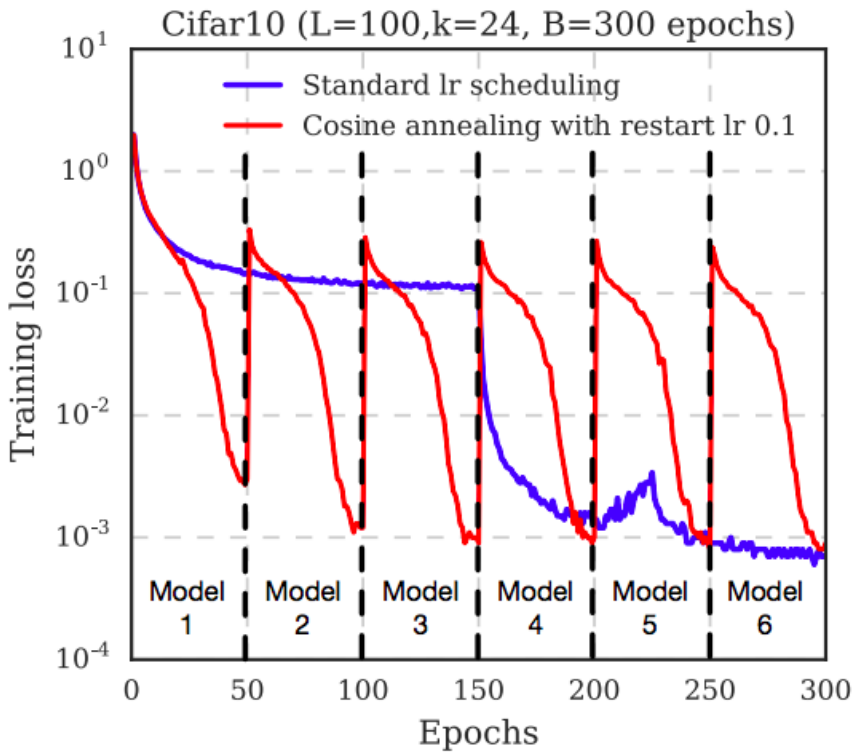
Hence two updates with $B = 1$ at learning rate η_1 is the same as one update at $B = 2$ and learning rate $2\eta_1$.

$$\eta_2 = 2\eta_1, \qquad \eta_B = B\eta_1$$

SGD as MCMC and MCMC as SGD

- **Gradient Estimation.** The accuracy of $\hat{g}$ as an estimate of g.
- **Gradient Drift (second order structure).** The fact that g changes as the parameters change.
- **Convergence.** To converge to a local optimum the learning rate must be gradually reduced toward zero.
- **Exploration.** Since deep models are non-convex we need to search over the parameter space. SGD can behave like MCMC.

Learning Rate as a Temperature Parameter



Gao Huang et. al., ICLR 2017

Gradient Flow

Total Gradient Descent: Φ `-=` ηg

Note this is g and not $\hat{g}$. Gradient flow is defined by

$$\frac{d\Phi}{dt} = -\eta g$$

Given $\Phi(0)$ we can calculate $\Phi(t)$ by taking the limit as $N \to \infty$ of Nt discrete-time total updates Φ `-=` $\frac{\eta}{N} g$.

The limit $N \to \infty$ of Nt **batch updates** Φ `-=` $\frac{\eta}{N} \hat{g}$ also gives $\Phi(t)$.

Gradient Flow Guarantees Progress

$$\begin{aligned}
\frac{d\ell}{dt} &= (\nabla_\Phi \, \ell(\Phi)) \cdot \frac{d\Phi}{dt} \\
&= -(\nabla_\Phi \, \ell(\Phi)) \cdot (\nabla_\Phi \, \ell(\Phi)) \\
&= -||\nabla_\Phi \, \ell(\Phi)||^2 \\
&\leq 0
\end{aligned}$$

If $\ell(\Phi) \geq 0$ then $\ell(\Phi)$ must converge to a limiting value.

This does not imply that Φ converges.

An Original Algorithm Derivation

We will derive a learning rate by maximizing a lower bound on the rate of reduction in training loss.

We must consider

- **Gradient Estimation.** The accuracy of $\hat{g}$ as an estimate of g.

- **Gradient Drift (second order structure).** The fact that g changes as the parameters change.

Analysis Plan

We will calculate a batch size B^* and learning rate η^* by optimizing an improvement guarantee for a single batch update.

We then use learning rate scaling to derive the learning rate η_B for a batch size $B << B^*$.

Deriving Learning Rates

If we can calculate B^* and η^* for optimal loss reduction in a single batch we can calculate η_B.

$$\eta_B = B\ \eta_1$$

$$\eta^* = B^* \eta_1$$

$$\eta_1 = \frac{\eta^*}{B^*}$$

$$\eta_B = \frac{B}{B^*}\ \eta^*$$

Calculating B^* and η^* in One Dimension

We will first calculate values B^* and η^* by optimizing the loss reduction over a single batch update in one dimension.

$$g = \hat{g} \pm \frac{2\hat{\sigma}}{\sqrt{B}}$$

$$\hat{\sigma} = \sqrt{E_{(x,y)\sim\text{Batch}}\left(\frac{d\ \text{loss}(\beta, x, y)}{d\beta} - \hat{g}\right)^2}$$

The Second Derivative of $\text{loss}(\beta)$

$$\text{loss}(\beta) = E_{(x,y)\sim\text{Train}}\, \text{loss}(\beta, x, y)$$

$$d^2\text{loss}(\beta)/d\beta^2 \leq L \quad \text{(Assumption)}$$

$$\text{loss}(\beta - \Delta\beta) \leq \text{loss}(\beta) - g\Delta\beta + \frac{1}{2}L\Delta\beta^2$$

$$\text{loss}(\beta - \eta\hat{g}) \leq \text{loss}(\beta) - g(\eta\hat{g}) + \frac{1}{2}L(\eta\hat{g})^2$$

A Progress Guarantee

$$\begin{aligned}
\text{loss}(\beta - \eta\hat{g}) &\leq \text{loss}(\beta) - g(\eta\hat{g}) + \frac{1}{2}L(\eta\hat{g})^2 \\
&= \text{loss}(\beta) - \eta(\hat{g} - (\hat{g} - g))\hat{g} + \frac{1}{2}L\eta^2\hat{g}^2 \\
&\leq \text{loss}(\beta) - \eta\left(\hat{g} - \frac{2\hat{\sigma}}{\sqrt{B}}\right)\hat{g} + \frac{1}{2}L\eta^2\hat{g}^2
\end{aligned}$$

Optimizing B and η

$$\text{loss}(\beta - \eta\hat{g}) \leq \text{loss}(\beta) - \eta\left(\hat{g} - \frac{2\hat{\sigma}}{\sqrt{B}}\right)\hat{g} + \frac{1}{2}L\eta^2\hat{g}^2$$

We optimize progress per gradient calculation by optimizing the right hand side divided by B. The derivation at the end of the slides gives

$$B^* = \frac{16\hat{\sigma}^2}{\hat{g}^2}, \quad \eta^* = \frac{1}{2L}$$

$$\eta_B = \frac{B}{B^*}\eta^* = \frac{B\hat{g}^2}{32\hat{\sigma}^2 L}$$

Recall this is all just in one dimension.

Estimating $\hat{g}_{B^*}$ and $\hat{\sigma}_{B^*}$

$$\eta_B = \frac{B\hat{g}^2}{32\hat{\sigma}^2 L}$$

We are left with the problem that $\hat{g}$ and $\hat{\sigma}$ are defined in terms of batch size $B^* >> B$.

We can estimate $\hat{g}_{B^*}$ and $\hat{\sigma}_{B^*}$ using a running average with a time constant corresponding to B^*.

Estimating $\hat{g}_{B^*}$

$$\hat{g}_{B^*} = \frac{1}{B^*} \sum_{(x,y)\sim \text{Batch}(B^*)} \frac{d\,\text{Loss}(\beta, x, y)}{d\beta}$$

$$= \frac{1}{N} \sum_{s=t-N+1}^{t} \hat{g}^s \quad \text{with } N = \frac{B^*}{B} \text{ for batch size } B$$

$$\tilde{g}^{t+1} = \left(1 - \frac{B}{B^*}\right) \tilde{g}^t + \frac{B}{B^*} \hat{g}^{t+1}$$

We are still working in just one dimension.

A Complete Calculation of η (in One Dimension)

$$
\begin{aligned}
\tilde{g}^{t+1} &= \left(1 - \frac{B}{B^*(t)}\right) \tilde{g}^t + \frac{B}{B^*(t)} \hat{g}^{t+1} \\
\tilde{s}^{t+1} &= \left(1 - \frac{B}{B^*(t)}\right) \tilde{s}^t + \frac{B}{B^*(t)} (\hat{g}^{t+1})^2 \\
\tilde{\sigma}^t &= \sqrt{\tilde{s}^t - (\tilde{g}^t)^2} \\
B^*(t) &= \begin{cases} K & \text{for } \; t \leq K \\ 16(\tilde{\sigma}^t)^2 / ((\tilde{g}^t)^2 + \epsilon) & \text{otherwise} \end{cases}
\end{aligned}
$$

A Complete Calculation of η (in One Dimension)

$$\eta^t = \begin{cases} 0 & \text{for } \; t \leq K \\ \dfrac{(\tilde{g}^t)^2}{32(\tilde{\sigma}^t)^2 L} & \text{otherwise} \end{cases}$$

As $t \to \infty$ we expect $\tilde{g}^t \to 0$ and $\tilde{\sigma}^t \to \sigma > 0$ which implies $\eta^t \to 0$.

The High Dimensional Case

So far we have been considering just one dimension.

We now propose treating each dimension $\Phi[c]$ of a high dimensional parameter vector Φ independently using the one dimensional analysis.

We can calculate $B^*[c]$ and $\eta^*[c]$ **for each individual parameter** $\Phi[c]$.

Of course the actual batch size B will be the same for all parameters.

A Complete Algorithm

$$\tilde{g}^{t+1}[c] = \left(1 - \frac{B}{B^*(t)[c]}\right) \tilde{g}^t[c] + \frac{B}{B^*(t)[c]} \hat{g}^{t+1}[c]$$

$$\tilde{s}^{t+1}[c] = \left(1 - \frac{B}{B^*(t)[c]}\right) \tilde{s}^t[c] + \frac{B}{B^*(t)[c]} \hat{g}^{t+1}[c]^2$$

$$\tilde{\sigma}^t[c] = \sqrt{\tilde{s}^t[c] - \tilde{g}^t[c]^2}$$

$$B^*(t)[c] = \begin{cases} K & \text{for } \ t \leq K \\ \lambda_B \tilde{\sigma}^t[c]^2 / (\tilde{g}^t[c]^2 + \epsilon) & \text{otherwise} \end{cases}$$

A Complete Algorithm

$$\eta^t[c] = \begin{cases} 0 & \text{for } \ t \leq K \\ \dfrac{\lambda_\eta \tilde{g}^t[c]^2}{\tilde{\sigma}^t[c]^2} & \text{otherwise} \end{cases}$$

$$\Phi^{t+1}[c] = \Phi^t[c] - \eta^t[c]\hat{g}^t[c]$$

Here we have meta-parameters K, λ_B, ϵ and λ_η.

Appendix: Optimizing B and η

$$\text{loss}(\beta - \eta\hat{g}) \leq \text{loss}(\beta) - \eta\hat{g}\left(\hat{g} - \frac{2\hat{\sigma}}{\sqrt{B}}\right) + \frac{1}{2}L\eta^2\hat{g}^2$$

Optimizing η we get

$$\hat{g}\left(\hat{g} - \frac{2\hat{\sigma}}{\sqrt{B}}\right) = L\eta\hat{g}^2$$

$$\eta^*(B) = \frac{1}{L}\left(1 - \frac{2\hat{\sigma}}{\hat{g}\sqrt{B}}\right)$$

Inserting this into the guarantee gives

$$\text{loss}(\Phi - \eta\hat{g}) \leq \text{loss}(\Phi) - \frac{L}{2}\eta^*(B)^2\hat{g}^2$$

Optimizing B

Optimizing progress per sample, or maximizing $\eta^*(B)^2/B$, we get

$$\begin{aligned}
\frac{\eta^*(B)^2}{B} &= \frac{1}{L^2}\left(\frac{1}{\sqrt{B}} - \frac{2\hat{\sigma}}{\hat{g}B}\right)^2 \\
0 &= -\frac{1}{2}B^{-\frac{3}{2}} + \frac{2\hat{\sigma}}{\hat{g}}B^{-2} \\
B^* &= \frac{16\hat{\sigma}^2}{\hat{g}^2} \\
\eta^*(B^*) = \eta^* &= \frac{1}{2L}
\end{aligned}$$

END