TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2018

Stochastic Gradient Descent (SGD)

The Classical Convergence Thoerem

RMSProp, Momentum and Adam

Scaling Learning Rates with Batch Size

Gradient Flow and Langevin Dynamics

Vanilla SGD

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

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

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

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.
- **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.

A One Dimensional Example

Suppose that y is a scalar, and consider

$$\begin{aligned} \mathrm{loss}(\beta, y) &= \frac{1}{2}(\beta - y)^2 \\ g &= E_{y\sim\mathrm{Pop}}\, \nabla_\beta\, \frac{1}{2}(\beta - y)^2 \\ &= \beta - E_{y\sim\mathrm{Pop}}\, y \\ \hat{g} &= \beta - E_{y\sim\mathrm{Batch}}\, y \end{aligned}$$

Even if β is optimal, for a finite batch we will have $\hat{g} \neq 0$.

The Classical Convergence Theorem

$$\Phi \mathrel{-=} \eta_t \nabla_\Phi \operatorname{loss}(\Phi, x_t, y_t)$$

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 $\lim_{t \rightarrow 0} \ \eta_t = 0$ we will eventually get to arbitrarilly small learning rates.

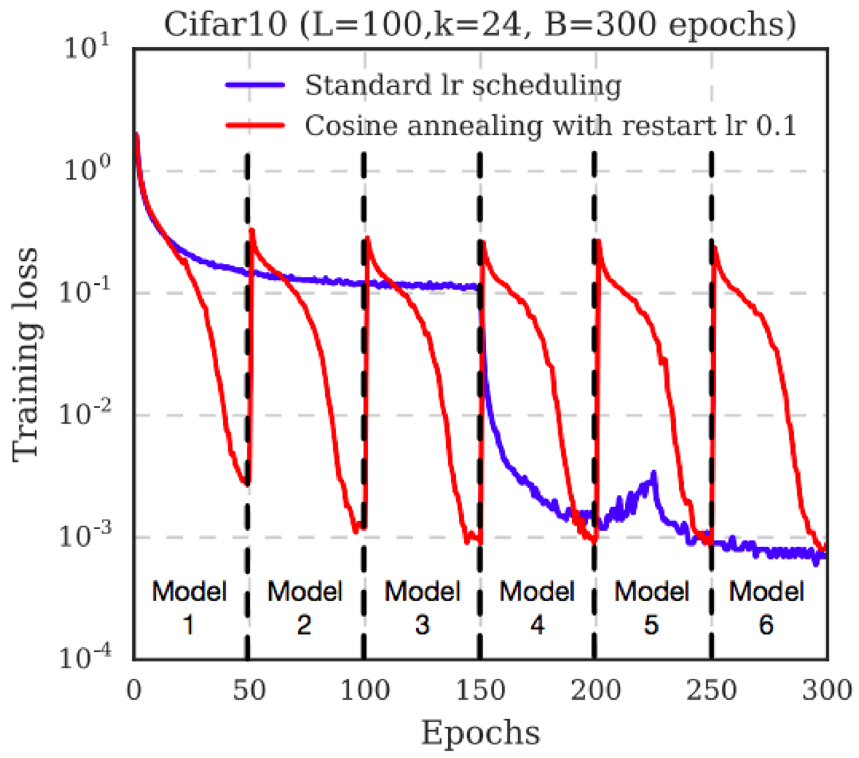
For sufficiently small learning rates any meaningful update of the parameters will be based on an arbitrarily large sample of gradients at essentially the same parameter value.

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.

SGD as a form of MCMC

Learning Rate as a Temperature Parameter



Gao Huang et. al., ICLR 2017

Standard Non-Vanilla SGD Algorithms

Digression on Running Averages

Consider a sequence x_1, x_2, x_3, ...

For $t \geq N$, consider the average of the N most recent values.

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

This can be approximated more efficiently with

$$\hat{\mu}_0 = 0$$

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

$$\hat{\mu}_t \approx \tilde{\mu}_t \quad t > N$$

Running Averages

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

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

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

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

RMSProp

RMSProp is based on a running average of $\hat{g}[i]^2$ for each real-valued model parameter i.

$$s_t[i] \;=\; \left(1 - \frac{1}{N_s}\right) s_{t-1}[i] + \frac{1}{N_s}\hat{g}_t[i]^2 \quad N_s \text{ typically 100 or 1000}$$

$$\Phi_{t+1}[i] \;=\; \Phi_t[i] - \frac{\eta}{\sqrt{s_t[i] + \epsilon}}\;\hat{g}_t[i]$$

$$s[i] \;\approx\; E\;\hat{g}[i]^2 = g[i]^2 + \sigma[i]^2$$

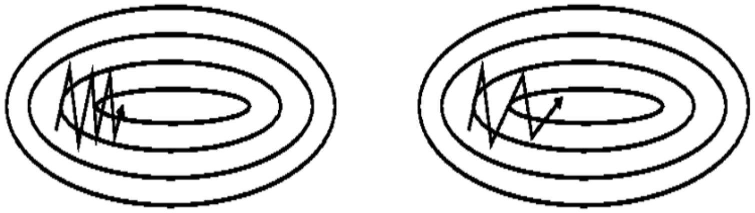
We should expect $\sigma[i] >> g[i]$.

Momentum

$$v_t = \left(1 - \frac{1}{N}\right) v_{t-1} + \eta * \hat{g}_t$$

$$\Phi_{t+1} = \Phi_t - v_t$$

The theory of momentum is generally given in terms of second order structure.



Rudin's blog

Momentum

However, second order analyses are controversial in Deep Learning.

We can perhaps get insight by reparameterizing the momentum equations.

A Reparameterization

$$\begin{aligned}
v_t &= \left(1 - \frac{1}{N}\right) v_{t-1} + \eta \hat{g}_t \\
&= \left(1 - \frac{1}{N}\right) v_{t-1} + \frac{1}{N}(N\eta\hat{g}_t) \\
&= N\eta\mu_t \ \ \text{for } \mu_t = \left(1 - \frac{1}{N}\right)\mu_{t-1} + \frac{1}{N}\hat{g}_t \\
\Phi_{t+1} &= \Phi_t - v_t \\
&= \Phi_t - N\eta\mu_t
\end{aligned}$$

A Reparameterization

$$\mu_t = \left(1 - \frac{1}{N_\mu}\right)\mu_{t-1} + \frac{1}{N_\mu}\hat{g}_t \quad N_\mu \text{ typically 10 or 100}$$

$$\Phi_{t+1} = \Phi_t - N\eta\mu_t \tag{1}$$

$$= \Phi_t - \eta'\mu_t \tag{2}$$

My intuition: For the parameterization N and η' defined by (2) it should be possible to optimize N and η' largely independently. I would not expect this to be true for the N and η defined by (1).

Adam — Adaptive Momentum

Adam combines momentum and RMSProp.

It also uses "bias correction" of running averages.

A Digression on Bias Correction of Running Averages

Consider a sequence $x_1,\ x_2,\ x_3, \ldots$ and consider the following for N large.

$$\hat{\mu}_0 \;=\; 0$$

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

For $\mu \doteq E\ x$ we have $E\ \hat{\mu}_1 = \mu/N$.

For $t << N$ we have $E\ \hat{\mu}_t \approx (t/N)\mu$.

Bias Correction of Running Averages

The following running average maintains the invariant that $\hat{\mu}_t$ is exactly the average of $x_1, \ldots, x_t$.

$$\begin{aligned} \hat{\mu}_0 &= 0 \\ \hat{\mu}_t &= \left(1 - \frac{1}{t}\right)\hat{\mu}_{t-1} + \left(\frac{1}{t}\right)x_t \end{aligned}$$

But this fails to track a moving average for $t >> N$.

Bias Correction of Running Averages

The following avoids the initial bias toward zero while still tracking a moving average.

$$\hat{\mu}_0 = 0$$

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

The published version of Adam has a more obscure form of bias correction which yields essentially the same effect.

Adam (simplified)

$$
\begin{aligned}
\mu_0[i] &= s_0[i] = 0 \\
\mu_t[i] &= \left(1 - \frac{1}{\min(t, N_g)}\right)\mu_{t-1}[i] + \frac{1}{\min(t, N_g)}\hat{g}_t[i] \\
s_t[i] &= \left(1 - \frac{1}{\min(t, N_s)}\right)s_{t-1}[i] + \frac{1}{\min(t, N_s)}\hat{g}_t[i]^2 \\
\Phi_{t+1}[i] &= \Phi_t - \frac{\eta}{\sqrt{s_t[i] + \epsilon}}\ \mu_t[i]
\end{aligned}
$$

Scaling η, N_μ and N_s Batch Size

Recent work has show that scaling hyper-parameters with the batch size can lead to effective learning with very large (highly parallel) batches.

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

Don't Decay the Learning Rate, Increase the Batch Size, Smith et al., 2018

Scaling η

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

$$\begin{aligned}
\Phi_{t+1} &= \Phi_t - \eta_1 \nabla_\Phi \mathrm{loss}(\Phi_t, x_t, y_t) \\
\Phi_{t+2} &= \Phi_{t+1} - \eta_1 \nabla_\Phi \mathrm{loss}(\Phi_{t+1}, x_{t+1}, y_{t+1}) \\
&\approx \Phi_{t+1} - \eta_1 \nabla_\Phi \mathrm{loss}(\Phi_t, x_{t+1}, y_{t+1}) \\
&= \Phi_t - \eta_1((\nabla_\Phi \mathrm{loss}(\Phi_t, x_t, y_t)) + (\nabla_\Phi \mathrm{loss}(\Phi_t, x_{t+1}, y_{t+1})))
\end{aligned}$$

Scaling η

Let η_B be the learning rate for batch size B.

$$\Phi_{t+2} \approx \Phi_t - \eta_1((\nabla_\Phi \mathrm{loss}(\Phi_t, x_t, y_t)) + (\nabla_\Phi \mathrm{loss}(\Phi_t, x_{t+1}, y_{t+1})))$$

$$= \Phi_t - 2\eta_1 \, \hat{g} \quad \text{for} \quad B = 2$$

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

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

Scaling N_μ

Let $N_{\mu,B}$ be the momentum parameter to be used with batch size B.

For batch size B, $\hat{\mu}_t$ is averaging over $N_{\mu,B}B$ gradient values.

Holding the number of included gradients constant gives

$$N_{\mu,B}B = N_{\mu,1} \quad \text{or} \quad N_{\mu,B} = N_{\mu,1}/B$$

Scaling N_s

The simple analysis for N_μ fails for N_s.

To estimate $E\ g[i]^2$ we should average $\hat{g}_t[i]^2$ over batch elements rather than batch averages.

The parameter N_s should be a number of gradients (batch elements) rather than a number of batches,

Under this semantics, N_s should be constant independent of batch size.

Gradient Flow and Langevin Dynamics

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 $N \rightarrow \infty$ of Nt discrete-time total updates Φ `-=` $\frac{\eta}{N} g$.

The limit $N \rightarrow \infty$ 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 $\mathrm{loss}(\beta)$

$$\begin{aligned}
\mathrm{loss}(\beta) &= E_{(x,y)\sim\mathrm{Train}}\ \mathrm{loss}(\beta, x, y) \\
d^2\mathrm{loss}(\beta)/d\beta^2 &\leq L \quad \text{(Assumption)} \\
\mathrm{loss}(\beta - \Delta\beta) &\leq \mathrm{loss}(\beta) - g\Delta\beta + \frac{1}{2}L\Delta\beta^2 \\
\mathrm{loss}(\beta - \eta\hat{g}) &\leq \mathrm{loss}(\beta) - g(\eta\hat{g}) + \frac{1}{2}L(\eta\hat{g})^2
\end{aligned}$$

A Progress Guarantee

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

Optimizing B and η

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

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

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

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

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

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

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

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

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)

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

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 $\Phi[i]$ of a high dimensional parameter vector Φ independently using the one dimensional analysis.

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

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

A Complete Algorithm

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

A Complete Algorithm

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

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

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

Appendix: Optimizing B and η

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

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

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

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