TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2018

SGD

Learning Rate and Batch Size

Momentum and Batch Size

RMSProp and Adam

SGD as MCMC and MCMC as SGD

Second Order Issues

Review of Attention

Neural Machine translation has replaced all previous approaches (at least at Google).

Attention is universal in machine translation and natural language processing (NLP) generally.

When using attention **input sentences are represented by a sequence of vectors** rather than a single thought vector.

$$
\begin{aligned}
\overset{\leftrightarrow}{h}^t &= \text{the vector representing the } t\text{th input position} \\
s^i &= \text{the vector representing the } i\text{th output position} \\
\alpha^{i,t} &= \text{The attention being paid to input } t \text{ when generating } s^{i+1} \\
&= \underset{t}{\text{softmax}}\ \tanh(W^a\ [s^i,\ \overset{\leftrightarrow}{h}^t]) \\
c^i &= \sum_t \alpha^{i,t}\ \overset{\leftrightarrow}{h}^t \\
s^{i+1} &= \text{RNNCELL}_\Phi(s^i,\ [e(y^i),\ c^i])
\end{aligned}
$$

$$Q_\Phi(:\ |\mathbf{x},\ y^1, \ldots, y^i) = Q_\Phi(:\ |s^{i+1}) = \underset{\hat{y}}{\text{softmax}}\ W^y\ s^{i+1}$$

$$\text{loss} = \sum_i -\log Q_\Phi(y^{i+1}|s^{i+1})$$

Vanilla SGD

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

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

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

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

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

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

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

Convergence

For random small batches any finite (non-zero) learning rate will converge to a random walk in parameter space.

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 and

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

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.

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.

The classical convergence theorem does not provide guidance in setting specific learning rate schedules.

Analysis Plan

I will relate learning rate to batch size.

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

I will use B^* and η^* to calculate η_B for $B << B^*$.

Relating Batch Size to Learning Rate

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

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

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

Batch Size and Learning Rate

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

$$= \Phi^t - 2\eta_1 \, \hat{g} \, |_{B=2}$$

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

$$\eta_2 = 2\eta_1$$

$$\eta_B = B\eta_1$$

Deriving Learning Rates

If we can calculate B^* and η^* for optimal loss reduction in a single batch we can calculate η_1.

$$\eta_B = B\ \eta_1$$

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

Calculating B and η_B in One Dimension

I will now calculate values B^* and η^* by optimizing the loss reduction over a single batch update.

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

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

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

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

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

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

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

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 now optimize progress per gradient calculation by optimizing the right hand side divided by B. This gives

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

$$B^* = \frac{16\hat{\sigma}^2}{\hat{g}^2}$$

$$\eta^* = \frac{1}{2L}$$

See the appendix at the end of these slides.

The η_1 Calculation

$$\eta_B = B\eta_1$$

$$\eta_1 = \frac{\eta^*}{B^*} = \frac{\hat{g}^2}{32\hat{\sigma}^2 L}$$

A Derived Algorithm

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

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

A Derived Algorithm

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

$$B^*[c] = 16\tilde{\sigma}[c]^2/\tilde{g}[c]^2$$

$$\eta[c] = B/(2B^*[c]L)$$

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

In Practice

The batch size is typically set to be just large enough to fully utilize available parallelism.

The learning rate can then be tuned.

For the same data set and loss function the optimal learning rate increases with batch size.

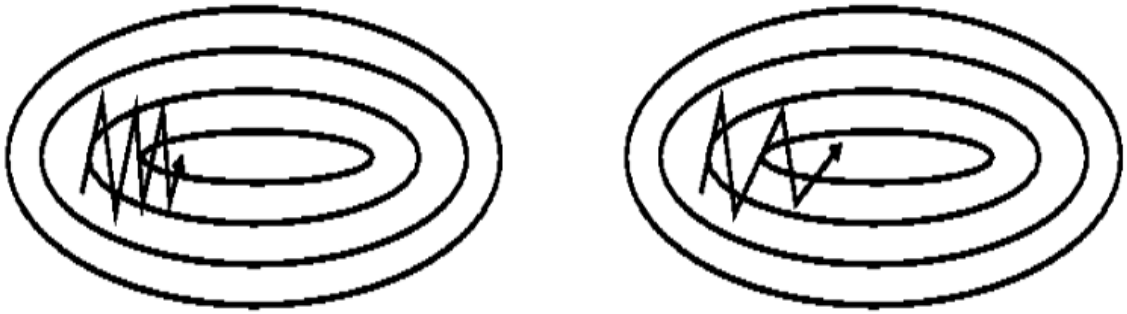
Very Large Batches

For the same model on the same data the optimal learning rate increases with batch size.

Recent work has shown that extremely large batches can lead to very fast training on very large datasets.

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

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 in terms of **gradient estimation**.

Momentum (Nonstandard Parameterization)

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

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

$\tilde{g}$ is a running average of $\hat{g}$

$\tilde{g}$ roughly averages over $1/(1-\mu)$ values of $\hat{g}$.

For this parameterization I expect $\eta = B\eta_1$ independent of μ and that momentum has no effect for $1/(1-\mu) << B^*$.

Momentum, Standard Parameterization

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

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

Setting $l_r = \eta(1 - \mu)$ gives $v^t = \eta\tilde{g}^t$ and the same sequence Φ^t as the nonstandard parameterization.

I expect that if we hold l_r at $B\eta_1(1 - \mu)$ and adjust μ then momentum has no effect for $1/(1 - \mu) << B^*$.

Nesterov Momentum

$$\tilde{g}^{t+1} = \mu\tilde{g}^t + (1-\mu)\hat{g}|_{\Phi^t - \eta\mu\tilde{g}^t}$$

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

This is very similar to standard momentum except that the gradient is measured at a "lookahead" parameter value $\Phi^t - \eta\mu\tilde{g}^t$ is different from both Φ^t and Φ^{t+1}.

This is motivated by gradient drift considerations (second order structure) which seems to be dominated by gradient estimation issues.

RMSProp

Adaptive Per-Channel Learning Rates.

RMS — Root Mean Square

$$s^{t+1}[c] = \beta s^t[c] + (1-\beta)\,\hat{g}[c]^2 \quad \text{PyTorch Default: } \beta = .99$$

$s^t[c]$ is a mean square.

$$\Phi^{t+1}[c] = \Phi^t[c] - \frac{l_r}{\sqrt{s^{t+1}[c] + \epsilon}} \quad \hat{g}[c]$$

Adam — Adaptive Momentum

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

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

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

Comments

From empirical experience, Adam is generally recommended.

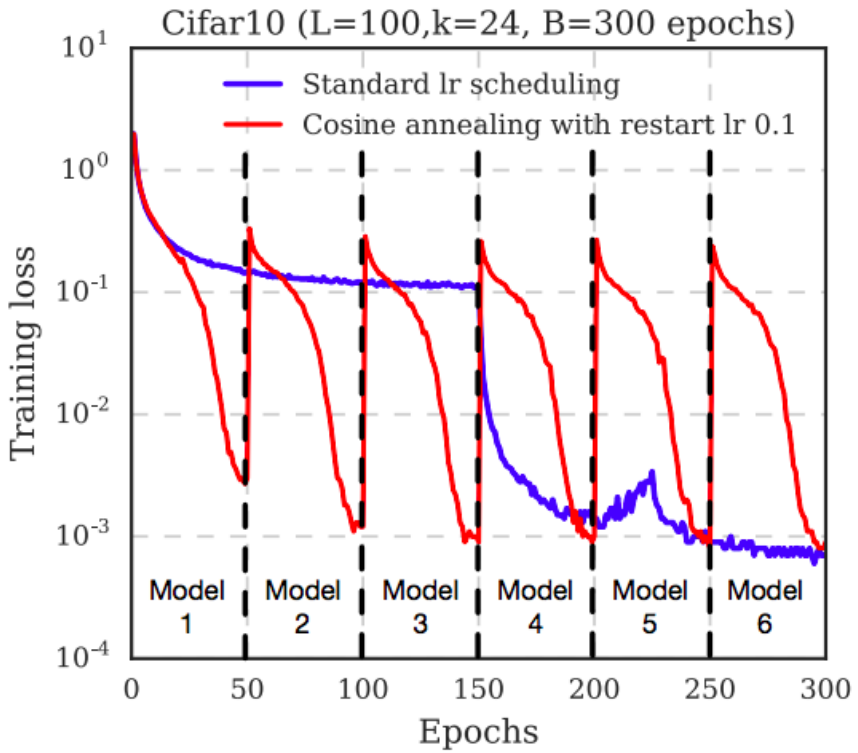
Adam seems less sensitive to its learning rate.

However, vanilla SGD remains competative when η is carefully tuned.

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.

Second Order SGD

The Gradient as a Dual Vector

Newton Updates and Quasi-Newton Methods

Hessian-Vector Products

Complex-Step Differentiation

What is a Gradient? Units of the Gradient.

Φ.grad[c] is rate of change in loss per change in $\Phi[c]$.

Consider log loss in nats $-\ln P$ vs. log loss in bits $-\log_2 P$.

We have different numerical values in nats than in bits.

Consider

$$\Phi[c] \texttt{ -= } \eta\Phi.\text{grad}[c]$$

The update will be a different size if we switch the units on the loss but leave η unchanged.

Abstract Vector Spaces and Coordinate Systems

For a vector space we can make an arbitrary choice of basis vectors (unit vectors) $u_1, \ldots, u_n$ that are linearly independent and span the space.

For any such basis, and for any vector x, there exist unique scalars $\alpha_1, \ldots, \alpha_n$ such that

$$x = \alpha_1 u_1 + \cdots + \alpha_n u_n$$

The values $(\alpha_1, \ \ldots, \ \alpha_n)$ are the numerical coordinates of x under that choice of basis (coordinate system).

The choice of basis (coordinates) is fundamentally arbitrary.

What is a Gradient?

The gradient $\nabla_\Phi \ell(\Phi)$ is the change in ℓ per change in Φ.

More formally, $\nabla_\Phi \ell(\Phi)$ is a linear map from $\Delta\Phi$ to $\Delta\ell$.

$$\ell(\Phi + \Delta\Phi) \approx \ell(\Phi) + [\nabla_\Phi \ell(\Phi)] (\Delta\Phi)$$

$$[\nabla_\Phi \ell(\Phi)] (\Delta\Phi) \equiv \lim_{\epsilon \to 0} \frac{\ell(\Phi + \epsilon\Delta\Phi) - \ell(\Phi)}{\epsilon}$$

No coordinates required.

Coordinates and Gradients

The dual of a vector space over the reals is the set of linear functions form the vector space to the reals.

The gradient $\nabla_\Phi \ell$ is a dual vector.

> **Observation**: Consider a gradient vector (dual vector) $\nabla_\Phi \ell(\Phi)$ and consider **any** direction $\Delta\Phi$ such that $[\nabla_\Phi \ell(\Phi)](\Delta\Phi) > 0$.
>
> There exists a coordinate system (a basis) in which $\nabla_\Phi \ell(\Phi)$ has the same coordinates as $\Delta\Phi$.

For an abstract vector space there is no natural or canonical update direction corresponding to a gradient.

Newton's Method: The Hessian

We can make a second order approximation to the loss function

$$\ell(\Phi + \Delta\Phi) \approx \ell(\Phi) + (\nabla_\Phi \, \ell(\Phi))\Delta\Phi + \frac{1}{2}\Delta\Phi^\top H \Delta\Phi$$

where H is the second derivative of ℓ, the Hessian, equal to $\nabla_\Phi \nabla_\Phi \, \ell(\Phi)$.

Again, no coordinates are needed — we can define the operator ∇_Φ generally indpendent of coordinates.

$$\Delta\Phi_1^\top \; H \; \Delta\Phi_2 = \Big(\nabla_\Phi \Big((\nabla_\Phi \, \ell^t(\Phi)) \cdot \Delta\Phi_1\Big)\Big) \cdot \Delta\Phi_2$$

Newton's Method

We consider the first order expansion of the gradient.

$$\nabla_\Phi \, \ell(\Phi) \, |_{\Phi+\Delta\Phi} \approx (\nabla_\Phi \, \ell(\Phi) \, |_\Phi) + H\Delta\Phi$$

We approximate Φ^* by setting this gradient approximation to zero.

$$0 = \nabla_\Phi \, \ell(\Phi) + H\Delta\Phi$$

$$\Delta\Phi = -H^{-1} \, \nabla_\Phi \, \ell(\Phi)$$

This gives Newton's method (without coordinates)

$$\Phi \texttt{ -= } H^{-1} \, \nabla_\Phi \, \ell(\Phi)$$

Newton Updates

It seems safer to take smaller steps. So it is common to use

$$\Phi \;\texttt{-=}\; \eta\, H^{-1}\, \nabla_{\Phi}\, \ell(\Phi)$$

for $\eta \in (0, 1)$ where η is naturally dimensionless.

Most second order methods attempt to approximate making updates in the Newton direction.

Quasi-Newton Methods

It is often faster and more effective to approximate the Hessian.

Maintain an approximation $M \approx H^{-1}$.

Repeat:

- Φ -= $\eta M \nabla_\Phi \ell(\Phi)$ (η is often optimized in this step).
- Restimate M.

The restimation of M typically involves a finite difference

$$\left(\nabla_\Phi \ell(\Phi) \mid_{\Phi^{t+1}}\right) - \left(\nabla_\Phi \ell(\Phi) \mid_{\Phi^t}\right)$$

As a numerical approximation of $H\Delta\Phi$.

Quasi-Newton Methods

Conjugate Gradient

BFGS

Limited Memory BFGS

Issues with Quasi-Newton Methods

In SGD the gradients are random even when Φ does not change.

We cannot use

$$\left(\nabla_\Phi \, \ell^{t+1}(\Phi)|_{\Phi^{t+1}}\right) - \left(\nabla_\Phi \, \ell^{t}(\Phi) \, |_{\Phi^{t}}\right)$$

as an estimate of $H\Delta\Phi$.

Hessian-Vector Products

$$H\Delta\Phi = \nabla_\Phi \left((\nabla_\Phi \, \ell^t(\Phi)) \cdot \Delta\Phi \right)$$

This is supported in PyTorch — in PyTorch Φ.grad is a variable while Φ.grad.data is a tensor.

Hessian-Vector Products

For backpropagation to be efficient it is important that the value of the graph is a scalar (like a loss). But note that for v fixed we have that

$$(\nabla_\Phi \, \ell^t(\Phi)) \cdot v$$

is a scalar and hence its gradient with respect to Φ, which is Hv, can be computed efficiently.

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