TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

Second Order Optimization Methods

Review of CNNs

$$L_{i+1} = \text{Relu}\left(\text{Conv}(L_i, f, p, s)\right)$$

L_i has shape (H, W, C), f has shape (F, F, C, C') (for a square filter). p is padding, s is stride

$$L_i' = \text{Pad}(L_i, p)$$

$$L_{i+1}[x, y, c'] = \text{Relu}\left(\sum_{u,v,c} f[u, v, c, c'] L_i'[sx + u, sy + v, c]\right)$$

L_{i+1} has shape (H', W', C') where $H' = \lfloor (H + 2p - F)/s \rfloor + 1$

Second Order SGD

The Gradient as a Dual Vector

Newton Updates and Quasi-Newton Methods

Hessian-Vector Products

Complex-Step Differentiation

Second Order Adaptive Descent (Speculative)

Review of SGD Central Issues

Consider a parameter vector Θ.

- **Gradient Estimation.** Estimating the gradient at a fixed Θ.

- **Gradient Drift.** The gradient changes as Θ changes.

- **Exploration.** At large learning rates SGD can behave like MCMC.

What is a Gradient? Units of the Gradient.

$\partial \ell / \partial \Theta_i$ is a change in cost (dollars or yen) per change in Θ_i.

Consider log loss in nats $\ln 1/P$ vs. log loss in bits $\log_2 1/P$.

This will have a different numerical value if we use nats than if we use bits.

Consider

$$\Theta_i \mathrel{-}= \eta(\partial \ell / \partial \Theta_i)$$

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, ..., u_n that are linearly independent and span the space.

For any such basis, and for any vector x, there exist unique scalars α_1, ..., α_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 $\nabla_\Theta \, \ell(\Theta)$ is the change in ℓ per change in Θ.

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

$$\ell(\Theta + \Delta\Theta) \approx \ell(\Theta) + [\nabla_\Theta \, \ell(\Theta)] \, (\Delta\Theta)$$

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

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 $\nabla_\Theta \ell$ is a dual vector.

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

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

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

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

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

$$\Delta\Theta_1^\top\; H\; \Delta\Theta_2 = \left(\nabla_\Theta \left((\nabla_\Theta\, \ell^t(\Theta)) \cdot \Delta\Theta_1\right)\right) \cdot \Delta\Theta_2$$

Newton's Method

We consider the first order expansion of the gradient.

$$\nabla_\Theta \, \ell(\Theta) \text{ @ } (\Theta + \Delta\Theta) \approx (\nabla_\Theta \, \ell(\Theta) \text{ @ } \Theta) + H\Delta\Theta$$

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

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

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

This gives Newton's method (without coordinates)

$$\Theta \mathrel{-}= H^{-1} \, \nabla_\Theta \, \ell(\Theta)$$

Newton Updates

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

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

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:

- Θ `-=` $\eta M \nabla_\Theta\, \ell(\Theta)$ (η is often optimized in this step).
- Restimate M.

The restimation of M typically involves a finite difference

$$\left(\nabla_\Theta\, \ell(\Theta) \text{ @ } \Theta^{t+1}\right) - \left(\nabla_\Theta\, \ell(\Theta) \text{ @ } \Theta^{t}\right)$$

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

Quasi-Newton Methods

Conjugate Gradient

BFGS

Limited Memory BFGS

Issues with Quasi-Newton Methods

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

We cannot use

$$\left(\nabla_\Theta\; \ell^{t+1}(\Theta) \;@\; \Theta^{t+1}\right) - \left(\nabla_\Theta\; \ell^{t}(\Theta) \;@\; \Theta^{t}\right)$$

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

Review of Adam

$$\hat{g} \ = \ \beta_1 \hat{g} + (1 - \beta_1) \nabla_\Theta \, \ell^t(\Theta)$$

$$\Theta \ \texttt{-=} \ \eta \odot \hat{g}$$

Here $\hat{g}$ is a gradient estimate — it is an average over a large sample of gratients.

It turns out that $H^t(\eta \odot \hat{g})$ can be computed exactly by a variant of backpropagation.

$$H^t = \nabla_\Theta \nabla_\Theta \, \ell^t(\Theta)$$

Estimating Gradient Drift

We have

$$\dot{g} = H(\eta \odot \hat{g}) = \mathrm{E}_i \left[H^i(\eta \odot \hat{g}) \right]$$

Here $\dot{g}$ is the rate of change of the gradient — the gradient drift.

Second Order Adam (Speculation)

We can estimate the gradient drift $\dot{g}$ as part of the algorithm.

$$\begin{aligned}\hat{g} &= \beta_1 \hat{g} + (1-\beta_1)\nabla_\Theta\, \ell^t(\Theta) \\ \widehat{\dot{g}} &= \beta_3 \widehat{\dot{g}} + (1-\beta_3) H^t(\eta \odot \hat{g}) \\ \Theta &\mathrel{-}= \eta \odot \hat{g}\end{aligned}$$

It seems likely that knowledge of the current gradient drift $\dot{g}$ should help in setting η_i.

Here we need to compute $H^t(\eta \odot \hat{g})$.

Hessian-Vector Products

There is a general set of optimization methods, **Krylov methods**, that involve computations of products the form $H\ \Delta\Theta$ for the Hessian H and a vector $\Delta\Theta$.

It turns out that backpropagation can be modified to compute $H^t\Delta\Theta$ as follows.

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

Hessian-Vector Products

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

This is supported by Theano and Tensor flow which are symbol-to-symbol frameworks but not other frameworks (including EDF) which are symbol-to-number.

A symbol-to-symbol framework constructs a computation graph for the computing the gradient. We can then do backpropagation on the gradient graph to get a second derivative (the Hessian).

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

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

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

But there is much better way of computing $H^t v$.

Complex-Step Differentiation

Consider a function $f : \mathbb{R} \rightarrow \mathbb{R}$ defined by a computer program.

Assume this program can be run on complex numbers simply by changing the data type of x.

Technically, we need that $f(x)$ is an **analytic** function.

James Lyness and Cleve Moler, Numerical Differentiation of Analytic Functions SIAM J. of Numerical Analysis, 1967.

Complex-Step Differentiation

Consider $f(x+i\epsilon)$ at real input x and consider the first order Taylor expansion.

$$f(x+i\epsilon) = f(x) + i(df/dx)\epsilon$$

Note that $f(x)$ and df/dx must both be real. Therefore

$$\mathrm{Im}(f(x+i\epsilon)) = \epsilon(df/dx)$$

$$\frac{df}{dx} = \frac{\mathrm{Im}(f(x+i\epsilon))}{\epsilon}$$

Complex-Step Differentiation

$$\frac{df}{dx} = \frac{\mathrm{Im}(f(x + i\epsilon))}{\epsilon}$$

This is vastly better than

$$\frac{df}{dx} \approx \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

The point is that in complex arithmetic the real and imaginary parts have independent floating point representations.

In 64 bit floating point arithmetic ϵ can be taken to be 2^{-50}.

For $\epsilon = 2^{-50}$, division by ϵ simply changes the exponent of the floating point representation leaving the mantissa unchanged.

First Order Polynomial Arithmetic

Numerically, complex-step differentiation is equivalent to first order polynomial arithmetic.

$$(a + b\epsilon)(a' + b'\epsilon) = (a + a') + (ab' + a'b)\epsilon$$

Differentiation based on first order polynomial arithmetic is exact.

Equivalence to Polynomial Arithmetic

$$(a + ib\epsilon)(a' + ib'\epsilon) = (a + a' - bb'\epsilon^2) + i(ab' + a'b)\epsilon$$

$$\epsilon = 2^{-50}$$

Here the ϵ^2 term is below the precision of $a + a'$.

Numerically, complex-step arithmetic and first order polynomial arithmetic are the same.

Hessian-Vector Products

We are interested in computing $H^t v$ for $v = (\eta \odot \hat{g})$.

$$H^t v = \frac{\mathrm{Im}(\nabla_\Theta \ell(\Theta) \ @ \ (\Theta + i\epsilon v))}{\epsilon}$$

$$\epsilon = 2^{-50}$$

Adaptive Descent

$$\Theta \mathrel{-}= \eta \odot \hat{g}$$

$$\sigma_i = \sqrt{s_i - (\hat{g}_i)^2} \qquad k_i = \left(\frac{2\sigma_i}{|\hat{g}_i|}\right)^2 \qquad \eta_i = \frac{1}{2L}\left(\frac{B}{k_i}\right)$$

$$\hat{g}_i = \left(1 - \frac{B}{k_i}\right)\hat{g}_i + \left(\frac{B}{k_i}\right)\left(\nabla_\Theta\, \ell^t(\Theta)\right)_i$$

$$s_i = \beta s_i + (1-\beta)\left(\nabla_\Theta\, \ell^t(\Theta)\right)_i^2$$

Second Order Adaptive Descent (Speculative)

$$\Theta \;\texttt{-=}\; \eta \odot \hat{g}$$

$$\sigma_i = \sqrt{s_i - (\hat{g}_i)^2} \qquad k_i = \left(\frac{2\sigma_i}{|\hat{g}_i|}\right)^2 \qquad \eta_i = \frac{1}{2\,|\widehat{\dot{g}}_i|}\left(\frac{B}{k_i}\right)$$

$$\hat{g}_i \;=\; \left(1 - \frac{B}{k_i}\right)\hat{g}_i + \left(\frac{B}{k_i}\right)\left(\nabla_\Theta\, \ell^t(\Theta)\right)_i$$

$$s_i \;=\; \beta s_i + (1-\beta)\left(\nabla_\Theta\, \ell^t(\Theta)\right)_i^2$$

$$\widehat{\dot{g}} \;=\; \beta_2\,\widehat{\dot{g}} \;+ (1-\beta_2)\, H^t(\eta \odot \hat{g})$$

Second Order Adaptive Descent (Speculative)

$$\Theta \;\; \text{-=} \;\; \eta \odot \hat{g}$$

$$\sigma_i = \sqrt{s_i - (\hat{g}_i)^2} \qquad k_i = \left(\frac{2\sigma_i}{|\hat{g}_i|}\right)^2 \qquad \eta_i = \frac{1}{2\,|\widehat{\dot{g}}_i|}\left(\frac{B}{k_i}\right)$$

$$\hat{g}_i \;=\; \left(1 - \frac{B}{k_i}\right)\hat{g}_i + \left(\frac{B}{k_i}\right)\left(\nabla_\Theta\, \ell^t(\Theta)\right)_i$$

$$s_i \;=\; \beta s_i + (1-\beta)\left(\nabla_\Theta\, \ell^t(\Theta)\right)_i^2$$

$$\widehat{\dot{g}} \;=\; \beta_2\, \widehat{\dot{g}} \;+ (1-\beta_2)\, H^t(\eta \odot \hat{g})$$

Second Order Adaptive Descent (Speculative)

$$\Theta \;\; \texttt{-=} \;\; \eta \odot \hat{g}$$

$$\sigma_i = \sqrt{s_i - (\hat{g}_i)^2} \qquad\qquad k_i = \left(\frac{2\sigma_i}{|\hat{g}_i|}\right)^2$$

$$\hat{g}_i \;=\; \left(1 - \frac{B}{k_i}\right)\hat{g}_i + \left(\frac{B}{k_i}\right)\left(\nabla_\Theta\, \ell^t(\Theta)\right)_i$$

$$s_i \;=\; \beta s_i + (1-\beta)\left(\nabla_\Theta\, \ell^t(\Theta)\right)_i^2$$

$$\widehat{\dot{g}} \;=\; \beta_2\, \widehat{\dot{g}} \;+ (1-\beta_2)\, H^t(\eta \odot \hat{g})$$

$$\eta_i \;=\; \beta_2 \eta_i + (1-\beta_2)\frac{1}{2\,|\widehat{\dot{g}}_i|}\left(\frac{B}{k_i}\right)$$

Summary

The Gradient as a Dual Vector

Newton and Quasi-Newton Methods

Hessian-Vector Products

Complex-Step Differentiation

Second Order Adaptive Descent (Speculative)

Postscript on Analytic Functions

$f : \mathbb{C} \to \mathbb{C}$ is analytic if it has a complex-valued derivative df/dx.

Note that a function from complex numbers maps two numbers (the real and imaginary part) to two numbers (a real and imaginary part).

Note that for $f(x) : \mathbb{R}^2 \to \mathbb{R}^2$ we have that $\nabla_x f(x)$ is a 2×2 Jacobian matrix with four degrees of freedom.

However, if it is possible to calculate an expression for the derivative over the complex numbers then the derivative is a single complex number (with two degrees of freedom).

For example, the derivative of x^2 is $2x$.

END