

TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2018

Loose Ends:

Gradient Descent Revisited

Hessian-Vector Products

Coordinates

For a vector space we can make an arbitrary choice of basis vectors b_1, \dots, b_N that are linearly independent and span the space.

A basis defines coordinates $x[i]$ for each vector x .

$$x = x[1]b_1 + \dots + x[N]b_N$$

The basis, and the induced coordinate system, is arbitrary.

Orthogonality is Coordinate-Relative

In two dimensions the vectors represented by $(1, 0)$ and $(0, 1)$ need not be orthogonal.

$(1, 0)$ represents b_1 and $(0, 1)$ represents b_2 .

We only require that b_1 and b_2 are independent.

Hence inner product is coordinate-relative.

Inner Products in Taylor Expansions are Coordinate-Independent

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

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 function from $\Delta\Phi$ to $\Delta\ell$.

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

Coordinate-Free Definition of the Gradient

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

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

$$\frac{\ell(\Phi + \epsilon\Delta\Phi) - \ell(\Phi)}{\epsilon} \approx [\nabla_{\Phi} \ell(\Phi)] (\Delta\Phi)$$

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

No coordinates required.

Dual Vectors

A dual vector is a linear function from vectors to scalars.

The gradient is a dual vector.

Coordinates

We calculate

$$[\nabla_{\Phi} \ell(\Phi)] \Delta\Phi$$

using coordinates.

$$[\nabla_{\Phi} \ell(\Phi)] \Delta\Phi = \sum_i \left[\frac{\partial \ell}{\partial \Phi[i]} \right] \Delta\Phi[i]$$

But this calculation is coordinate-independent.

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

Strange Coordinate Systems

Consider any gradient $\nabla_{\Phi}\ell(\Phi)$ at any value of Φ .

For any such situation, and **any vector $\Delta\Phi$** with $[\nabla_{\Phi}\ell(\Phi)] \Delta\Phi > 0$, and **any learning rate $\eta > 0$** , there exists a coordinate system in which **$\eta\Phi.\text{grad}[c] = \Delta\Phi[c]$** and hence

$$\Phi_{t+1} = \Phi_t - \Delta\Phi$$

Note that gradient decent always yields $[\nabla_{\Phi}\ell(\Phi)] \Delta\Phi > 0$.

Proof

Define the basis vectors b_1, \dots, b_N by

$$b_1 \doteq \frac{\Delta\Phi}{\sqrt{\eta[\nabla_{\Phi}\ell(\Phi)]\Delta\Phi}} \quad [\nabla_{\Phi}\ell(\Phi)] \quad b_i = 0 \quad \text{for } i > 1$$

In this coordinate system we have

$$\Phi = \Phi[1]b_1 + \dots + \Phi[N]b_n$$

$$\frac{\partial\ell(\Phi)}{\partial\Phi[1]} = \frac{\sqrt{[\nabla_{\Phi}\ell(\Phi)]\Delta\Phi}}{\sqrt{\eta}} \quad \frac{\partial\ell(\Phi)}{\partial\Phi[j]} = 0 \quad \text{for } j > 0$$

$$\sum_i \Phi.\text{grad}[i]\Phi[i]b_i = \frac{\sqrt{[\nabla_{\Phi}\ell(\Phi)]\Delta\Phi}}{\sqrt{\eta}} b_1 = \frac{\Delta\Phi}{\eta}$$

Coordinate-Free Versions of SGD

The Gradient Covariance Martix

$$\Sigma \doteq E_t (\hat{g}_t - g)(\hat{g}_t - g)^\top$$

$$\Phi_{t+1} = \Phi_t - \eta \Sigma^{-1} \nabla_{\Phi} \text{Loss}(\Phi, x_t, y_t)$$

Information Geometry and the Natural Gradient

We consider the case where loss is determined by a probability distribution. For example $-\log P(y)$.

The set of all distributions P forms a manifold.

For a given point (distribution) P we can consider the ball

$$B_\epsilon(P) = \{Q \mid KL(P, Q) \leq \epsilon\}$$

$$\Delta P = \operatorname{argmin}_{\Delta P \in B_\epsilon(P)} \ell(P + \Delta P)$$

Distance Functions Define a Point-Wise Inner Product

For any smooth (doubly differentiable) function $d(x, y)$ with $d(x, y) \geq 0$ and $d(x, x) = 0$ we must have

$$\nabla_{\Delta x} d(x, x + \Delta x)|_{\Delta x=0} = 0$$

$$d(x, x + \Delta x) \approx \Delta x^\top H \Delta x$$

$$H \doteq \nabla_{\Delta x} \nabla_{\Delta x} d(x, x + \Delta x)|_{\Delta x=0}$$

The coordinate-independent gradient direction defined by $d(x, y)$ is then

$$H^{-1} \nabla_x \ell(x)$$

Information Geometry and Gradient Descent

For KL divergence H is diagonal with

$$\Delta P^\top H \Delta P = \sum_y \frac{\Delta P(y)^2}{P(y)}$$

Although KL is not symmetric, H happens to be the same for $KL(P, P + \Delta P)$ and $KL(P + \Delta P, P)$.

Newton's Method

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 independent of coordinates.

$$\Delta\Phi_1^{\top} H \Delta\Phi_2 = (\nabla_{\Phi} ((\nabla_{\Phi} \ell_t(\Phi)) \cdot \Delta\Phi_1)) \cdot \Delta\Phi_2$$

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)

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

Newton Updates

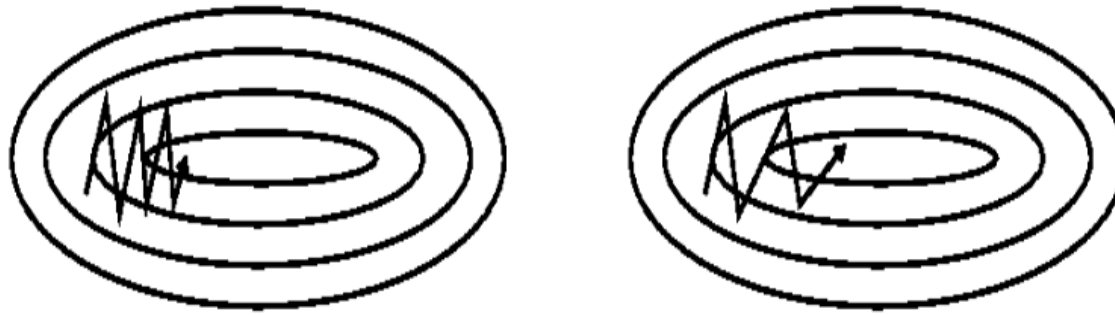
It seems safer to take smaller steps. So it is common to use

$$\Phi \ -= \ \textcolor{red}{\eta} \ H^{-1} \ \nabla_{\Phi} \ \ell(\Phi)$$

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

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

Second Order SGD



Rudin's blog

Focusing on the Hessian.

Quasi-Newton Methods

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

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

Repeat:

- $\Phi \leftarrow \Phi - \eta M \nabla_{\Phi} \ell(\Phi)$ (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 $H\Delta\Phi$.

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

$$\Phi_{t+1} = \Phi_t - \eta_t \nabla_{\Phi} \text{loss}(\Phi, x_t, y_t)$$

For sufficiently smooth loss functions, and holding the coordinate system constant through time, and for

$$\eta_t > 0 \quad \text{and} \quad \lim_{t \rightarrow \infty} \eta_t = 0 \quad \text{and} \quad \sum_t \eta_t = \infty,$$

the loss value of SGD converges.

Structure From Motion

See the Videos

The Levenberg-Marquart Algorithm (Bundle Adjustment)

$$\text{Loss} = E_{(x,y) \sim \text{Train}} \frac{1}{2} ||f_{\Phi}(x) - y||^2$$

$$f_{\Phi+\Delta\Phi}(x_t) \approx f_{\Phi}(x_t) + J_t \Delta\Phi$$

$$J_t = \nabla_{\Phi} f_{\Phi}(x_t)$$

$$\hat{g}_t = (f_{\Phi}(x_t) - y_t) J_t = r_t J_t$$

$$r_t = f_{\Phi}(x_t) - y_t$$

The Levenberg-Marquart Algorithm

$$\text{Loss}(\Phi + \Delta\Phi) \approx E_t \frac{1}{2} \|r_t + J_t \Delta\Phi\|^2$$

Minimizing this squared error over the choice of $\Delta\Phi$ is a least squares regression problem.

$$0 = E_t (r_t + J_t \Delta\Phi) J_t^\top$$

$$0 = (E_t r_t J_t^\top) + E_t J_t^\top J_t \Delta\Phi$$

$$(E_t J_t^\top J_t) \Delta\Phi = -E_t \hat{g}_t$$

$$\Delta\Phi = -(E_t J_t^\top J_t)^{-1} (E_t \hat{g}_t)$$

A General Loss Function

$$\text{Loss}_t(f(\Phi + \Delta\Phi)) \approx L_t + \hat{g}_t^\top \Delta\Phi + (J_t \Delta\Phi)^\top H_t J_t \Delta\Phi$$

$$J_t \doteq \nabla_\Phi f(\Phi)$$

$$H_t \doteq \nabla_f \nabla_f \text{Loss}_t(f)$$

Setting the gradient to zero and solving for $\Delta\Phi$:

$$\Delta\Phi_t = -(E_t \ J_t^\top H_t J_t)^{-1} \hat{g}_t$$

The Levenberg-Marquart Algorithm for Log Loss

$$\Phi_{t+1} = \Phi_t - \eta (E_t \ J_t^\top H_t \ J_t)^{-1} \hat{g}_t$$

$$\text{Loss}(\Phi, x, y) = -\log Q_{f_\Phi(x)}(y)$$

$$Q_f(y) = \text{softmax}_y f(y)$$

$$H_t = E_{y \sim Q_{f_t}} (\delta_y - Q_{f_t})(\delta_y - Q_{f_t})^\top$$

$$= \text{Diag}(Q_{f_t}) - Q_{f_t} Q_{f_t}^T$$

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 $\Phi.\text{grad}$ is a variable while $\Phi.\text{grad.data}$ 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.

Complex-Step Differentiation

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

For C code on a CPU we can run program the program on complex numbers simply by changing the data type of x .

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

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

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

Complex-Step Differentiation

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

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_tv for $v = (\eta \odot \hat{g})$.

$$H_tv = \frac{\text{Im}(\nabla_{\Phi} \ell(\Phi)|_{\Phi+i\epsilon v})}{\epsilon}$$

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

Phrase Based Statistical Machine Translation (SMT)

Step I: Learn a phrase table — a set of triples (p, q, s) where

- p is a (short) sequence of source words.
- q is a (short) sequence of target words.
- s is a score.

(“au”, “to the”, .5) (“au banque”, “the the bank”, .01)

For a phrase P we will write P .source for the source phrase, P .target for the target phrase, and P .score for the score.

Derivations

Consider an input sentence x of length T .

We will write $x[s : t]$ for the substring $x[s], \dots, x[t - 1]$.

A derivation d from x is a sequence $(P_1, s_1, t_1,), \dots, (P_K, s_K, t_K)$ where $P_k.\text{source} = x[s_k : t_k]$.

The substrings $x[s_k : t_k]$ should be disjoint and “cover” x .

For $d = [(P_1, s_1, t_1,), \dots, (P_L, s_K, t_K)]$ we define

$$y(d) \equiv P_1.\text{target} \cdots P_K.\text{target}$$

We let $D(x)$ be the set of derivations from x .

Scoring

For $d \in D(x)$ we define a score $s(d)$

$$s(d) = \alpha \ln P_{\text{LM}}(y(d)) + \Phi \sum_k P_k.\text{score} + \gamma \text{distortion}(d)$$

where $P_{\text{LM}}(y)$ is the probability assigned to string y under a language model for the target language

and $\text{distortion}(d)$ is a measure of consistency of word ordering between source and target strings as defined by the indices $(s_1, t_1), \dots, (s_K, t_K)$.

Translation

$$y(x) = y(d^*(x))$$

$$d^*(x) = \operatorname{argmax}_{d \in D(x)} s(d)$$

END