TTIC 31230, Fundamentals of Deep Learning

David McAllester, Autumn 2020

Stochastic Gradient Descent (SGD)

RMSProp and Adam

RMSProp and Adam

RMSProp and Adam are "adaptive" SGD methods — they use different learning rates for different model parameters where the parameter-specific learning rate is computed from statistics of the data.

Adam is variant of RMSProp with momentum and "debiasing".

RMSProp

RMSProp was introduced in Hinton's class lecture slides.

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

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

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

RMSProp

The second moment of a scalar random variable x is $E\ x^2$

The variance σ^2 of x is $E\ (x - \mu)^2$ with $\mu = E\ x$.

RMSProp uses an estimate $s[i]$ of the second moment of the random scalar $\hat{g}[i]$.

If the mean $g[i]$ is small then $s[i]$ approximates the variance of $\hat{g}[i]$.

There is a "centering" option in PyTorch RMSProp that switches from the second moment to the variance.

RMSProp Motivation

If we assume constant noise covariance and locally positive definite quadratic loss in the neighborhood of the stationary distribution then, in a coordinate system where the Hessian is the identity matrix, vanilla SGD has a stationary distribution of the form

$$p(\Phi) \propto \exp\left(-\sum_i \frac{\Phi_i^2}{\alpha\eta\sigma_i^2}\right)$$

where, because we have used the Hessian-normalized coordinate system, we have

$$\mathcal{L} = ||\Phi||^2$$

RMSProp Motivation

If we set $\eta_i = \eta_0/\sigma_i^2$ this becomes

$$p(\Phi) \propto \exp\left(\frac{-\mathcal{L}}{\alpha\eta_0}\right)$$

The Gibbs stationary distribution seems desirable for exploration.

RMSProp Motivation

However, computing the Hessian is extremely challenging and RMSprop works in the given coordinates.

Also, centered RMSProp divides by σ_i rather than σ_i^2.

RMSProp is Theoretically Mysterious

$$\Phi[i] \text{ -= } \eta \frac{\hat{g}[i]}{\sigma[i]} \quad (1) \qquad\qquad \Phi[i] \text{ -= } \eta \frac{\hat{g}[i]}{\sigma^2[i]} \quad (2)$$

Although (1) seems to work better, (2) is better motivated theoretically. To see this we can consider units.

If parameters have units of "weight", and loss is in bits, then (2) type checks with η having units of bits — the numerical value of η has no dependence on the choice of the weight unit.

Consistent with the dimensional analysis, many theoretical analyses support (2) over (1) contrary to apparent empirical performance.

Adam — Adaptive Momentum

Adam combines momentum and RMSProp.

PyTorch RMSProp also supports momentum. However, it presumably uses the standard momentum learning rate parameter which couples the temperature to both the learning rate and the momentum parameter. Without an understanding of the coupling to temperature, hyper-parameter optimization is then difficult.

Adam uses a momentum parameter that is naturally decoupled from temperature.

Adam also uses "bias correction".

Bias Correction

Consider a standard moving average.

$$\tilde{x}_0 = 0$$

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

For $t < N$ the average $\tilde{x}_t$ will be strongly biased toward zero.

Bias Correction

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

$$\tilde{x}_t = \left(\frac{t-1}{t}\right)\tilde{x}_{t-1} + \left(\frac{1}{t}\right)x_t$$

$$= \left(1 - \frac{1}{t}\right)\tilde{x}_{t-1} + \left(\frac{1}{t}\right)x_t$$

We now have $\tilde{x}_1 = x_1$ independent of any x_0.

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

Bias Correction

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

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

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

Adam (simplified)

$$\tilde{g}_t[i] = \left(1 - \frac{1}{\min(t, N_g)}\right) \tilde{g}_{t-1}[i] + \frac{1}{\min(t, N_g)} \hat{g}_t[i]$$

$$s_t[i] = \left(1 - \frac{1}{\min(t, N_s)}\right) s_{t-1}[i] + \frac{1}{\min(t, N_s)} \hat{g}_t[i]^2$$

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

Decoupling η from ϵ

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

The optimal ϵ is often large. For large ϵ it is useful to set $\eta = \epsilon\eta_0$ in which case we get

$$\Phi_{t+1}[i] = \Phi_t - \frac{\eta_0}{1 + \frac{1}{\epsilon}\sqrt{s_t[i]}} \ \tilde{g}_t[i]$$

We then get standard SGD as $\epsilon \to \infty$ holding η_0 fixed.

Making Adam Independent of B

Making Adam independent of the batch size B is difficult.

Rather than

$$s_t[i] = \left(1 - \frac{1}{N_s}\right) s_{t-1}[i] + \frac{1}{N_s} \hat{g}_t[i]^2$$

we would like

$$s_t[i] = \left(1 - \frac{1}{N_s}\right) s_{t-1}[i] + \frac{1}{N_s} \left(\frac{1}{B} \sum_b \hat{g}_{t,b}[i]^2\right)$$

Making Adam Independent of B

In PyTorch this is difficult because "optimizers" are defined as a function of $\hat{g}_t$.

$\hat{g}_t[i]$ is not sufficient for computing $\sum_b \ \hat{g}_{t,b}[i]^2$.

To compute $\sum_b \ \hat{g}_{t,b}[i]^2$ we need to maintain a batch index in the gradiant attribute of parameters.

END