TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2020

Stochastic Gradient Descent (SGD)

RMSProp and Adam and Decoupled Versions

RMSProp and Adam

RMSProp and Adam are "adaptive" SGD methods — the effective learning rate is computed from statistics of the data rather than set as a fixed hyper-parameter (although the adaptation algorithm itself still has hyper-parameters).

Different effective learning rates are used for different model parameters.

Adam is typically used in NLP while Vanilla SGD is typically used in vision. This may be related to the fact that batch normalization is used in vision but not in NLP.

RMSProp

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 $\hat{g}[i]$.

For $g[i]$ small $s[i]$ 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

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

One interpretation of RMSProp is that a low variance gradient has less statistical uncertainty and hence needs less averaging before making the update.

RMSProp is Theoretically Mysterious

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

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 inverse 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 (although PyTorch RMSProp also supports momentum).

Adam also uses "bias correction" which probably accounts for it's popularity over RMSProp in practice.

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)

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

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

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

Decoupling Hyperparametera

The following reparameterization should be helpful for Adam.

$$N_g = min(1, N_g^0/B)$$

$$\eta = \epsilon B \eta_0$$

Empirically, tuning ϵ is important.

Some coupling remains between η_0 and ϵ.

N_s should also be adapted to B but this is problematic — see the next slide.

Making $s_t[i]$ batch size invariant

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

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

$$N_s = \min\left(1, N_s^0/B\right) \quad N_s^0 \text{ optimal for } B = 1$$

Making $s_t[i]$ Batch Size Invariant

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 modify the backward method of all PyTorch objects!

Summary

We have considered Vanilla SGD, Momentum, RMSProp and Adam.

Vanilla tends to be used in vision while Adam tends to be used in NLP.

Reparameterization of the PyTorch hyperparameters can decouple hyperparameters simplifying hyperparameter search.

END