TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2020

Stochastic Gradient Descent (SGD)

The Classical Convergence Theorem

The Learning Rate as Temperature

Temperature, Batch Size, Momentum, and Adam

Vanilla SGD

$$\Phi_{t+1} = \Phi_t - \eta_t \hat{g}$$

$$\hat{g} \;=\; E_{(x,y)\sim \mathrm{Batch}}\; \nabla_\Phi \; \mathcal{L}(\Phi, x, y)$$

$$g \;=\; E_{(x,y)\sim \mathrm{Pop}}\; \nabla_\Phi \; \mathcal{L}(\Phi, x, y)$$

η_t is the learning rate for time t.

Issues

- **Gradient Noise:** 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:** Convergence requires $\eta_t \to 0$.

- **Exploration:** Reducing η slowly allows better exploration of model parameters.

The Classical Convergence Theorem

$$\Phi \mathrel{-}= \eta_t \nabla_\Phi \, \mathcal{L}(\Phi, x_t, y_t)$$

For "sufficiently smooth" non-negative loss with

$$\eta_t \geq 0 \qquad \lim_{t \to \infty} \eta_t = 0 \qquad \sum_t \eta_t = \infty \qquad \sum_t \eta_t^2 < \infty$$

we have that the training loss $E_{(x,y)\sim\text{Train}} \, \mathcal{L}(\Phi, x, t)$ converges to a limit and any limit point of the sequence Φ_t is a stationary point in the sense that $\nabla_\Phi \, E_{(x,y)\sim\text{Train}} \, \mathcal{L}(\Phi, x, t) = 0$.

Rigor Police: One can construct cases where Φ diverges to infinity, converges to a saddle point, or even converges to a limit cycle.

Physicist's Proof of the Convergence Theorem

Since $\lim_{t \rightarrow 0}\ \eta_t = 0$ we will eventually get to arbitrarilly small learning rates.

For sufficiently small learning rates any meaningful update of the parameters will be based on an arbitrarily large sample of gradients at essentially the same parameter value.

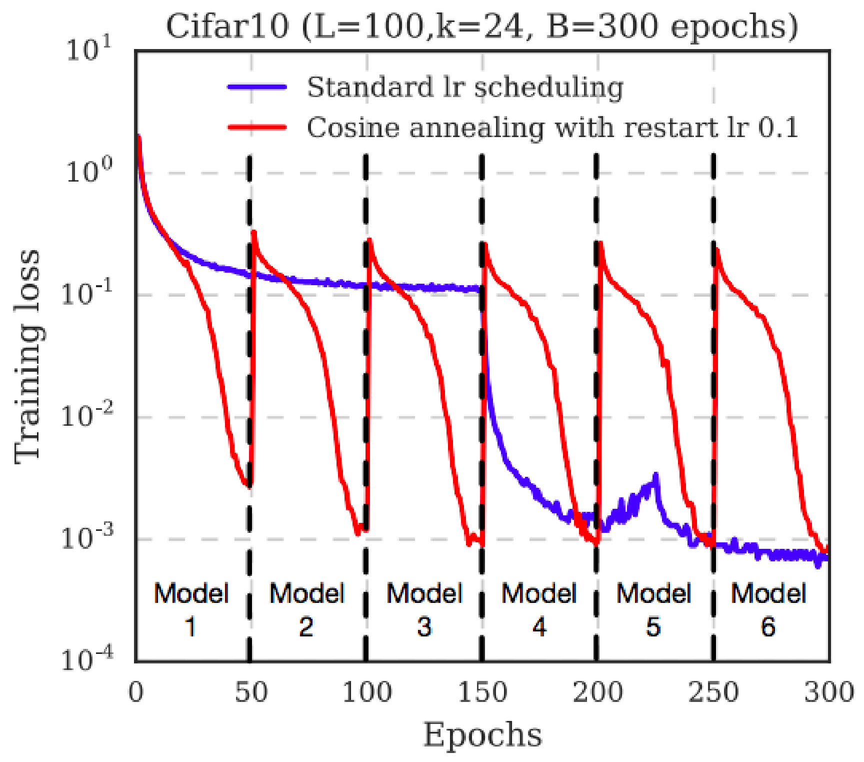
An arbitrarily large sample will become arbitrarily accurate as an estimate of the full gradient.

But since $\sum_t \eta_t = \infty$, no matter how small the learning rate gets, we still can make arbitrarily large motions in parameter space.

For a rigorous proof see Neuro-Dynamic Programming, Bertsekas and Tsitsiklis, 1996.

SGD as a form of MCMC

Learning Rate as a Temperature Parameter



Gao Huang et. al., ICLR 2017

Temperature

Physical temperature is a relationship between the energy and probability.

$$P(x) = \frac{1}{Z}\, e^{\frac{-E(x)}{kT}} \qquad Z = \sum_{x} e^{\frac{-E(x)}{kT}}$$

This is called the Gibbs or Boltzman distribution.

$E(x)$ is the energy of physical microstate state x.

k is Boltzman's constant.

Z is called the partition function.

Temperature

Boltzman's constant can be measured using the ideal gas law.

$$pV = NkT$$

$$\begin{aligned} p &= \text{pressure} \\ V &= \text{volume} \\ N &= \text{the number of molecules} \\ T &= \text{temperature} \\ k &= \text{Boltzman's constant} \end{aligned}$$

We can measure p, V, N and T and solve for k.

Temperature

The Gibbs distribution is typically written as

$$P(x) = \frac{1}{Z}\, e^{-\beta E(x)}$$

$\beta = \frac{1}{kT}$ is the (inverse) temperature parameter.

"Hot" is when β is small and "cold" is when β is large (confusing).

Loss as Energy
Learning Rate as Temperature

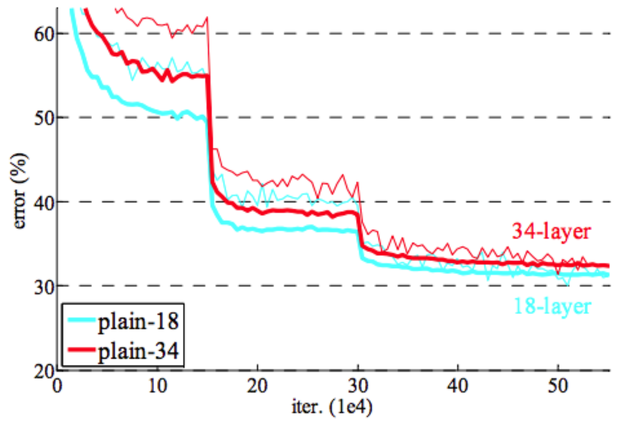
A finite learning rate defines an equalibrium probability distribution (or density) over the model parameters.

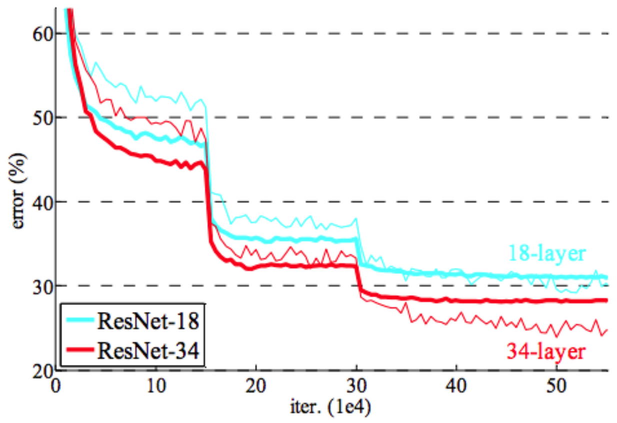
Each value of the model parameters has an associated loss.

The distribution over model parameters defines a distribution over loss.

Loss as Energy

Learning Rate as Temperature



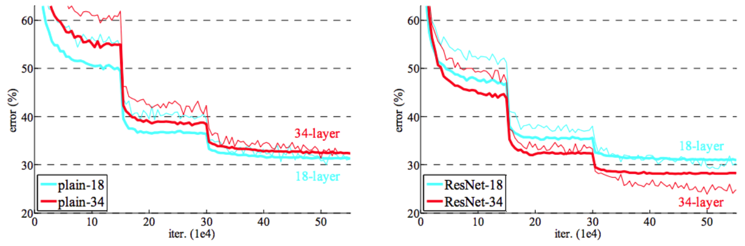


Equalibrium energy (loss) distributions at three different temperatures (learning rates).

Plots are from the ResNet paper. Left plot is for CNNs without residual skip connections, the right plot is ResNet. Thin lines are training error, thick lines are validation error. In all cases η is reduced twice, each time by a factor of 2.

Loss as Energy

Learning Rate as Temperature



The learnng rate is always reduced over time. The profile of learning rate reduction is called **the learning rate schedule**. An older convention (shown here) is to reduce the learning rate by a factor of 2 in steps.

Modern transformer training first quickly and smoothly ramps up the learning rate (the warm-up phase) up and then slowly and smoothly ramps it down.

Batch Size and Temperature

Vanilla SGD with minibatching typically uses the following update which defines the meaning of η.

$$\begin{aligned} \Phi_{t+1} & \mathrel{-}= \eta \hat{g}_t \\ \hat{g}_t & = \frac{1}{B} \sum_b \hat{g}_{t,b} \end{aligned}$$

Here $\hat{g}_b$ is the average gradient over the batch.

Under this update **increasing the batch size (while holding η fixed) reduces the temperature.**

Making Temperature Independent of B

For batch size 1 with learning rate η_0 we have

$$\Phi_{t+1} = \Phi_t - \eta_0 \nabla_\Phi \mathcal{L}(t, \Phi_t)$$

$$\begin{aligned}\Phi_{t+B} &= \Phi_t - \sum_{b=0}^{B-1} \eta_0 \nabla_\Phi \mathcal{L}(t+b, \Phi_{t+b-1}) \\ &\approx \Phi_t - \eta_0 \sum_b \nabla_\Phi \mathcal{L}(t+b, \Phi_t) \\ &= \Phi_t - B\eta_0 \, \hat{g}_t\end{aligned}$$

For batch updates $\Phi_{t+1} = \Phi_t - B\eta_0 \, \hat{g}_t$ the temperature is essentially determined by η_0 independent of B.

Making Temperature Independent of B

In 2017 it was discovered that setting $\eta = B\eta_0$ allows very large (highly parallel) batches.

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

Momentum

Moment is equivalent to using an exponential moving average (EMA) of the gradient.

Exponential Moving Average (EMA)

Consider a sequence x_1, x_2, x_3, ...

For $t \geq N$, consider the average of the N most recent values.

$$\overline{x}_t = \frac{1}{N} \sum_{k=0}^{N-1} x_{t-k}$$

This can be approximated efficiently with

$$\begin{aligned} \tilde{x}_0 &= 0 \\ \tilde{x}_t &= \left(1 - \frac{1}{N}\right) \tilde{x}_{t-1} + \left(\frac{1}{N}\right) x_t \end{aligned}$$

Exponential Moving Average (EMA)

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

$$= \frac{1}{N} \sum_{s=1}^{t} \left[\left(1 - \frac{1}{N}\right)^{t-s} x_s \right] \approx \frac{1}{N} \sum_{s=1}^{t} e^{-(s-t)/N} x_s$$

$$\sum_{i=0}^{\infty} \left(1 - \frac{1}{N}\right)^{i} = N$$

Deep Learning Convention for EMAs

In deep learning an exponential moving average

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

is written as

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

where

$$\beta = 1 - 1/N$$

Typical values for β are .9, .99 or .999 corresponding to N being 10, 100 or 1000.

Bias Correction

Consider a standard moving average.

$$\begin{aligned}\tilde{x}_0 &= 0\\ \tilde{x}_t &= \left(1-\frac{1}{N}\right)\tilde{x}_{t-1}+\left(\frac{1}{N}\right)x_t\end{aligned}$$

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 subtle form of bias correction which yields the same effect.

EMA Momentum

The adaptive momentum (Adam) optimizer uses EMA momentum.

Here we focus on the EMA momentum and discuss adaptation later.

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

$$\Phi_{t+1} = \Phi_t - \eta\tilde{g}_t$$

EMA Momentum Is Decoupled From Temperature

We will adopt the rule of thumb that the temperature is determined by the total effect of a single training gradient $g_{t,b}$.

Also that "temperature" corresponds to the converged loss at fixed learning rate.

Total Effect Rule

The effect of $g_{t,b}$ on the batch average $\hat{g}_t$ is $\left(\frac{1}{B}\right) g_{t,b}$.

$$\text{Using} \qquad \sum_{i=0}^{\infty} \frac{1}{N}\left(1-\frac{1}{N}\right)^i = 1$$

we get that the effect of $\hat{g}_t$ on $\sum_{t=0}^{\infty} \tilde{g}_t$ equals $\hat{g}_t$.

So for $\Phi_{t+1} = \Phi_t - N\eta\tilde{g}_t$ the total effect of $g_{t,b}$ is $(N/B)\eta\; g_{t,b}$.

Momentum

The theory of momentum is generally given in terms of second order structure and total gradients (GD rather than SGD).

But second order analyses are controversial for SDG in very large dimension.

Still, momentum is widely used in practice.

Momentum in PyTorch Vanilla SGD

The standard (PyTorch) momentum SGD equations are

$$v_t = \mu v_{t-1} + \eta * \hat{g}_t \quad \mu \text{ is typically .9 or .99}$$

$$\Phi_{t+1} = \Phi_t - v_t$$

Here v is velocity, $0 \leq \mu < 1$ represents friction drag and $\eta\hat{g}$ is the acceleration generated by the gradient force.

Momentum and Temperature

$$v_t = \mu v_{t-1} + \eta * \hat{g}_t \quad \mu \text{ is typically .9 or .99}$$

$$\Phi_{t+1} = \Phi_t - v_t$$

We will use a first order analysis to argue that by setting

$$\eta = (1-\mu)B\eta_0$$

the temperature will be essentially determined by η_0 independent of the choice of the momentum parameter μ or the batch size B.

Momentum and Temperature

$$\eta = (1 - \mu)B\eta_0$$

Emprical evidence for this setting of η is given in

Don't Decay the Learning Rate, Increase the Batch Size, Smith et al., 2018

Momentum as an EMA

$$
\begin{aligned}
v_t &= \mu v_{t-1} + \eta \hat{g}_t \\
&= \left(1 - \frac{1}{N}\right) v_{t-1} + \frac{1}{N}(N\eta\hat{g}_t)
\end{aligned}
$$

We see that v_t is an EMA of $N\eta\hat{g}$.

Momentum as an EMA

v_t is an EMA of $N\eta\hat{g}$.

Alternatively, we can consider an EMA of the gradient.

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

The moving average of $N\eta\hat{g}$ is the same as $N\eta$ times the moving average of $\hat{g}$. Hence

$$v_t = N\eta\tilde{g}_t$$

Momentum as an EMA

We have now shown that the standard formulation of momentum can be written as

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

$$\Phi_{t+1} = \Phi_t - N\eta\tilde{g}_t$$

Total Effect Rule

We will adopt the rule of thumb that the temperature is determined by the total effect of a single training gradient $g_{t,b}$.

Also that "temperature" corresponds to the converged loss at fixed learning rate.

Total Effect Rule

The effect of $g_{t,b}$ on the batch average $\hat{g}_t$ is $\left(\frac{1}{B}\right) g_{t,b}$.

$$\text{Using} \quad \sum_{i=0}^{\infty} \frac{1}{N}\left(1-\frac{1}{N}\right)^i = 1$$

we get that the effect of $\hat{g}_t$ on $\sum_{t=0}^{\infty} \tilde{g}_t$ equals $\hat{g}_t$.

So for $\Phi_{t+1} = \Phi_t - N\eta\tilde{g}_t$ the total effect of $g_{t,b}$ is $(N/B)\eta\ g_{t,b}$.

Total Effect Rule

For $\Phi_{t+1} = \Phi_t - N\eta\tilde{g}_t$ the total effect of $g_{t,b}$ is $(N/B)\eta\, g_{t,b}$.

By taking $\eta = \frac{B}{N}\eta_0$ we get that the total effect, and hence the temperature, is determined by η_0 independent of the choice of N and B.

For the standard momentum paramenter $\mu = (1 - 1/N)$ this becomes

$$\eta = (1-\mu)B\eta_0$$

where η_0 determines temperature independent of μ and B.

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

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

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.

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