DS 4440 Optimizer Matters

Gradient Descent

$$\hat{\theta}_{t+1} \leftarrow \hat{\theta}_{\tau} - \overset{\text{learning rate}}{\alpha} \underbrace{\nabla_{\theta} \mathcal{L}(\hat{\theta}_{\tau}, x, y)}_{\text{either per instance or batch}}$$

Variants

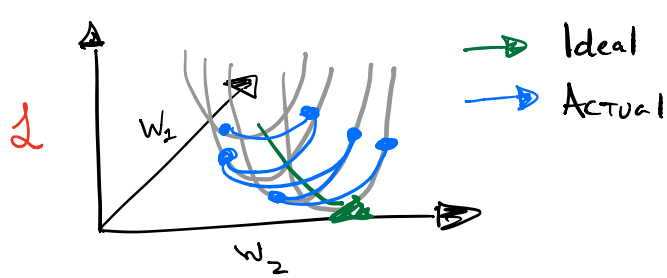
Stochastic One instance at a time

Batch all instances at once

Mini batch b samples at once

Basic GD is myopic which can cause zig-zagging → slow convergence

Sometimes loss curves are sharper in some dims than others.



Normalizing inputs can mitigate this a bit.

$\boxed{\text{Momentum}}$ is another approach

$$V_\tau \leftarrow \gamma V_{\tau-1} - \alpha \nabla_\theta \mathcal{L}(\hat{\theta}_{\tau-1}, x, y)$$

Keep going in direction you've been going

But adjust locally

$$\hat{\theta}_\tau \leftarrow \hat{\theta}_{\tau-1} + V_\tau$$

In fact, we already "know" where we are going at this point; why not look-ahead? This is the idea behind Nesterov gradient descent, a variant of momentum.

$$V_\tau \leftarrow \gamma V_{\tau-1} - \alpha \nabla_\theta \mathcal{L}(\boxed{\hat{\theta}_{\tau-1} + \gamma V_{\tau-1}}, x, y)$$

move toward accumulated ∇

(See examples in CoLab.)

So far we have assumed the learning rate is the same for all parameters.

But you can imagine optimal rates varying across params.

Let: $g_{\tau,j} = \nabla_\theta \mathcal{L}(\theta_\tau)_j$

That is, the partial for param j at iteration τ. So far we have said:

$$\hat{\theta}_{(\tau+1),j} \leftarrow \hat{\theta}_{\tau,j} + \alpha g_{\tau,j}^{\star}$$

$\star$ (Credit to Sebastian Ruder for this framing.)

But having a global α may be suboptimal.

$\boxed{\text{Adagrad}}$ is one approach that defines adaptive, per-parameter learning rates.

$$\hat{\theta}_{(\tau+1),j} \leftarrow \hat{\theta}_{\tau,j} + \alpha_{\tau,j} g_{\tau,j}$$

Intuition: Move slower for parameters we have updated a bunch already.

$$\alpha_{\tau,j} \stackrel{\text{def}}{=} \frac{\alpha}{G_{\tau,j} + \epsilon}$$

avoids div. by $\emptyset$

Where $G_{\tau,j}$ is the sum of the squares of grads for param j

$$\sum_\tau \nabla_\theta \left(\mathcal{L}(\hat{\theta}_\tau, x, y)\right)_j^2$$

$$\hat{\theta}_{(\tau+1),j} \leftarrow \hat{\theta}_{\tau,j} - \alpha_{\tau,j} g_{\tau,j}$$

If we have accumulated a lot of updates for j by time $t \rightarrow$ smaller steps

All entries in G grow over time; this means descent slows.

$\boxed{\text{RMSProp}}$ Extends Adagrad by reducing the effect of monotonically $\downarrow$ learning rates.

We keep a <u>decaying</u> <u>average</u> of past squared gradients.

$$\mathbb{E}[g^2]_t \overset{\text{def}}{=} \lambda \, \mathbb{E}[g^2]_{t-1} + (1-\lambda) g_t^2$$

Where λ is our decay term.

Can replace entries in G with this.

This is very similar to Adadelta.

$\boxed{\text{Adam}}$ (Adaptive Moment Estimation)

Also maintains parameter-wise learning rates

$$m_\tau \leftarrow \beta_1 m_{(\tau-1)} + (1-\beta_1) g_\tau$$

Estimate of mean of grad

$$v_\tau \leftarrow \beta_2 v_{(\tau-1)} + (1-\beta_2) g_\tau^2$$

Estimate of Variance

then:

$$\hat{\theta}_{(\tau+1),j} \leftarrow \hat{\theta}_{\tau,j} - \frac{\alpha}{\sqrt{v_\tau}+\epsilon} \cdot m_\tau$$

Simplified slightly; ignores bias-correction

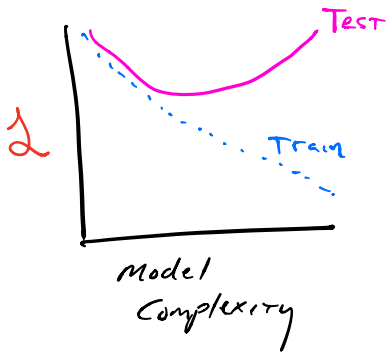
This is a bit like Momentum + RMSProp

Optimizers Recap

Lots of options; a few key ideas

- Exploiting history
- Per-parameter and adaptive rates

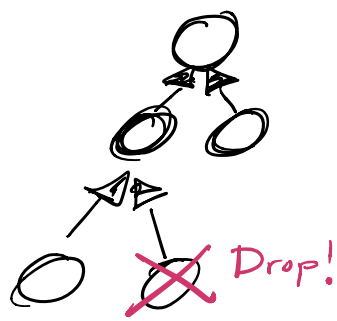
A few notes on regularization & overfitting



Weight regularization

Impose a norm loss to keep weights <u>small</u>.

Dropout



Randomly mask (drop) weights during training.

v = random sample $[0, 1] \rightarrow v_j = 0$ if $v_j < p$ else 1

$\tilde{h} = h \odot v$

At test time: $\tilde{h} = h \odot v$

Early Stopping

Monitor loss on a 'nested' set; keep $\hat{\theta}$ that gave best performance.