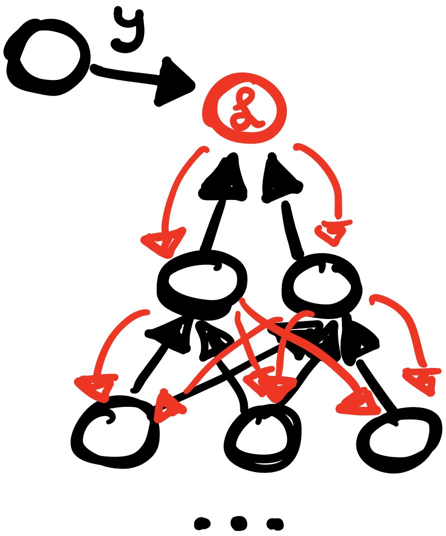
DS 4440

Optimizer Matters



Backprop yields gradients for all parameters efficiently ($\nabla_W \mathcal{L}$).

Cool! ...Why did we want these again? For Gradient Descent!

$$\hat{W}_{t+1} \leftarrow \hat{W}_t - \alpha \nabla_W \mathcal{L}(\hat{W}_t, X, y)$$

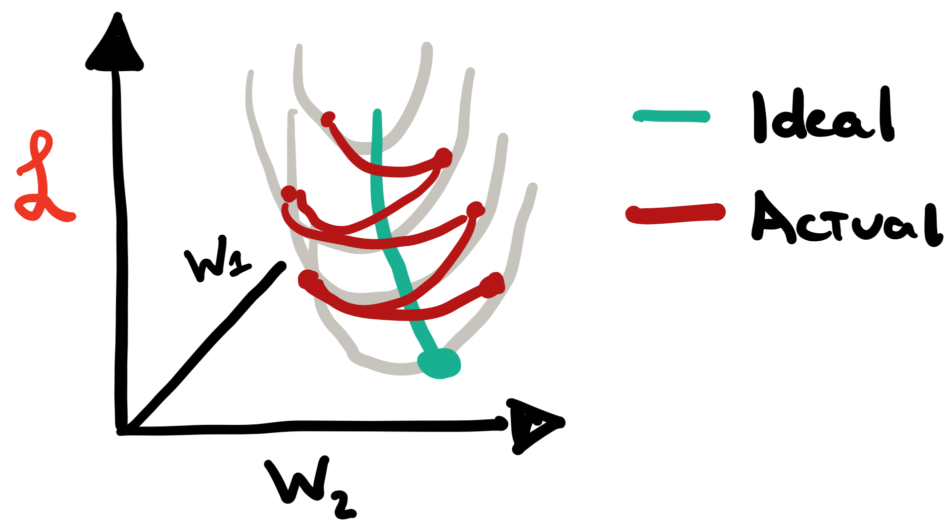
learning rate

Variants

Stochastic 1 instance (x, y) per step

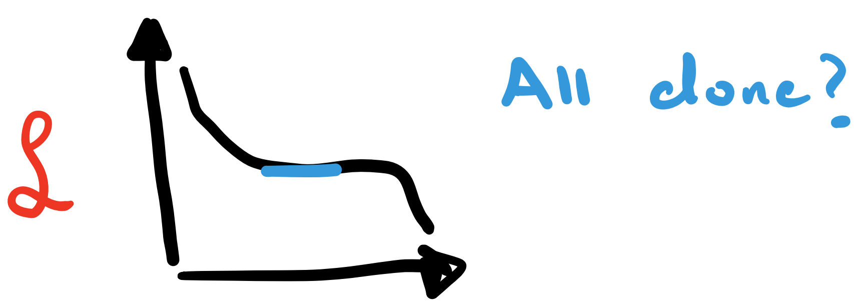
Batch Sets of (x, y); $\frac{1}{|B|} \sum \nabla_W \mathcal{L}$

Basic Gradient Descent is Myopic in that it zig zags → Slow Convergence



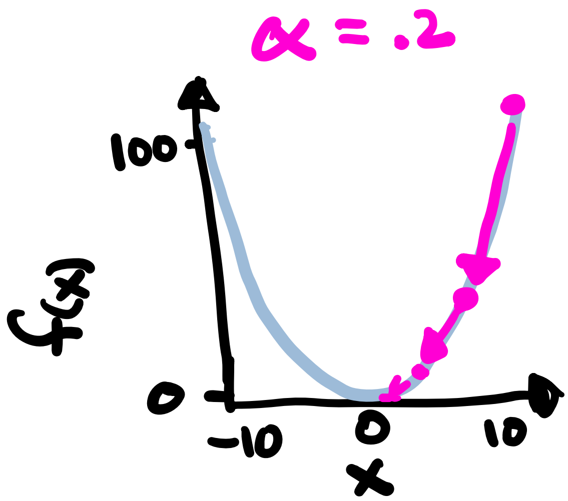
Normalizing inputs can help a bit.

A general Challenge: Saddle points



The Importance of learning rate α

$f(x) = x^2$ $\quad \frac{d}{dx} f(x) = 2x$ $\quad x_{init} = 10$



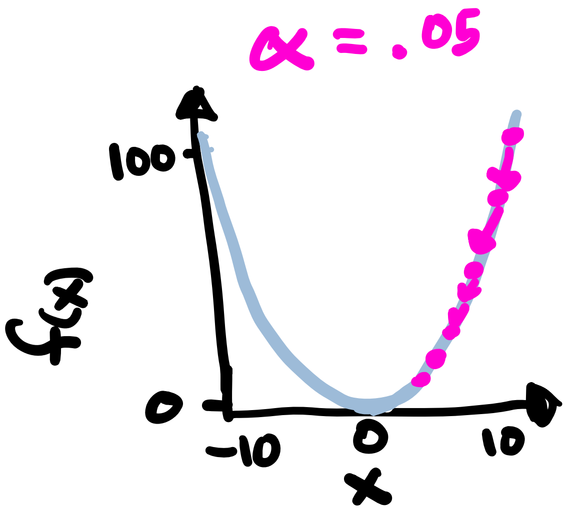
$.2 \cdot (2 \cdot 10) = 4$ $\quad x = 6$

$.2 \cdot (2 \cdot 6) = 2.4$ $\quad x = 3.6$

$.2 \cdot (2 \cdot 2.4) = .96$ $\quad x \approx 2.6$

...

But $\alpha = .05$



$.05 \cdot (2 \cdot 10) = 1$ $\quad x = 9$

$.05 \cdot (2 \cdot 9) = .9$ $\quad x = 8.1$

$.05 \cdot (2 \cdot 8.1) = .81$ $\quad x \cong 7.3$

... Slooooow

Momentum

But adjust locally (e.g. .1)

$$v_t \leftarrow \gamma v_{t-1} - \alpha \nabla_W \mathcal{L}(\hat{W}_t, x, y)$$

Keep going in direction you've been going ($\gamma = .9$, e.g.)

$$\hat{W}_{\tau+1} \leftarrow \hat{W}_\tau + V_\tau$$

We can even anticipate where we are going:

$$V_\tau \leftarrow \gamma V_{\tau-1} - \alpha \mathcal{L}(\underbrace{\hat{W}_\tau + \gamma V_{\tau-1}}_{\text{Move toward accumulated } \nabla_W}, x, y)$$

This is Nesterov Momentum.

(See examples in CoLab.)

So far we have assumed a single learning rate (α) for all parameters.

Let $g_{\tau,j} \doteq \nabla_W \mathcal{L}(\hat{W}_\tau)_j$

$$\hat{W}_{(\tau+1),j} \leftarrow \hat{W}_{\tau,j} - \alpha_{\boxed{\tau,j}}\, g_{\tau,j}$$ | "AdaGrad"

But how to set $\alpha_{\tau,j}$?

Intuition: Move slower for params we have updated a bunch

$$\alpha_{\tau,j} \doteq \frac{\alpha}{\sqrt{G_{\tau,j}} + \epsilon}$$ — Avoids div. by $\emptyset$.

Sum of grads for j.

$$\sum_{\tau} \left[\nabla_W \mathcal{L}(\hat{W}_\tau, x, y) \right]_j^2$$

Cumulative updates slow rate.

RMSProp Extends AdaGrad by keeping a decaying average of the squares of ∇ terms.

$$\boxed{S_{\tau,j} \doteq \lambda S_{(\tau-1),j} + (1-\lambda) g_{\tau,j}^2}$$

Then we divide α by $\sqrt{S_{t,j}}$.

Adam is maybe the go-to optimizer now. We keep two vars per param:

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \approx \text{Momentum}$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \approx \text{RMS Prop}$$

Then

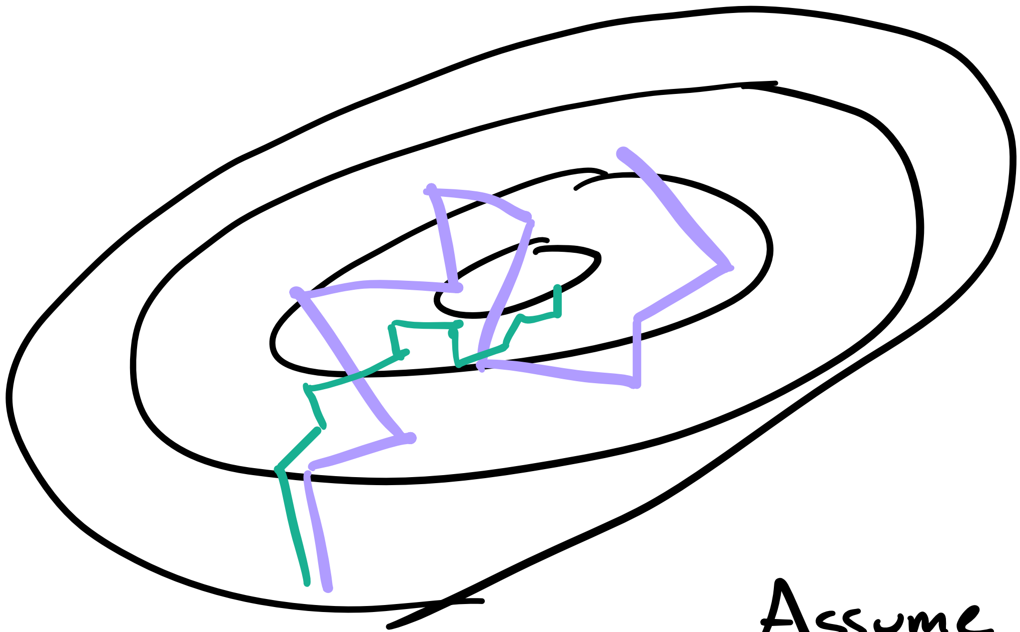
$$\underbrace{\hat{W}_{(t+1),j} \leftarrow \hat{W}_{t,j} - \frac{\alpha}{\sqrt{v_{t,j}} + \epsilon} \cdot m_{t,j}}$$

Slightly simplified; ignores bias-correction.

Learning rate decay

Loss Surface

Big steps to start → smaller over time



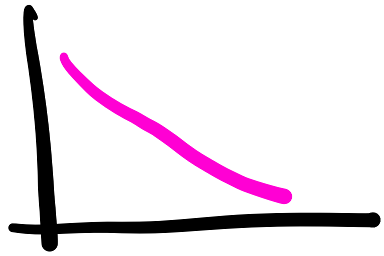
Assume noisey minibatches

Big α might jump too much

Shrinking α at each step might help

$$\alpha = \frac{1}{1 + d \cdot \text{epoch}} \cdot \alpha_0$$

decay rate

If $d = 1$ → 

Can also do exponential decay

$$\alpha = (.9)^{epoch} \cdot \alpha_0$$

Note on hyperparameter Tuning

- Choose random Values (not grid)