Gradient descent

In the multivariate case:

$$\nabla_x f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \dots\right]^T$$

$$f(x+\varepsilon) = f(x) + \varepsilon^T \nabla f(x) + O(\|\varepsilon\|^2)$$

$$\varepsilon = -\eta \nabla f(x)$$

$\hookrightarrow$ small scalar > 0

$$f(x - \eta \nabla f(x)) = f(x) - \underbrace{\eta \nabla f(x)^T \nabla f(x)}_{> 0} + O(\dots)$$

$$f(x - \eta \nabla f(x)) \lesssim f(x)$$

In ML, we have $L(y^{(i)}, x^{(i)}, \theta) = L_i(\theta)$

We want to minimize L by changing θ

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} L_i(\theta)$$

$$\nabla_\theta L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta L_i(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

"Batch" gradient descent

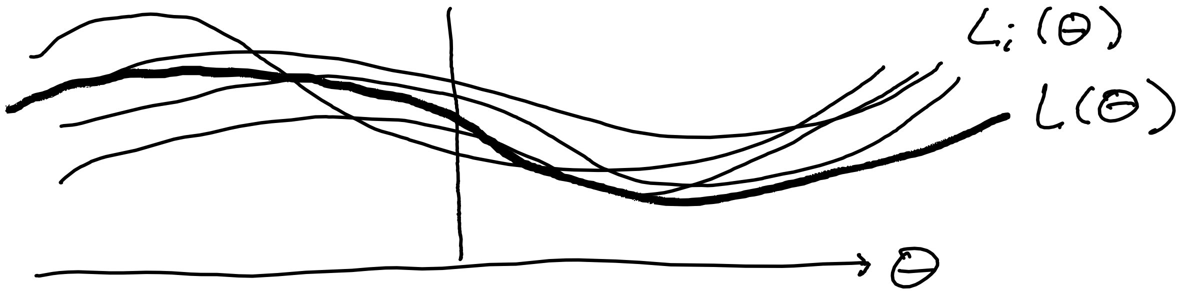
$$\theta \leftarrow \theta - \eta \nabla_\theta L_i(\theta)$$

$i \sim \text{uniform categorical}(n)$

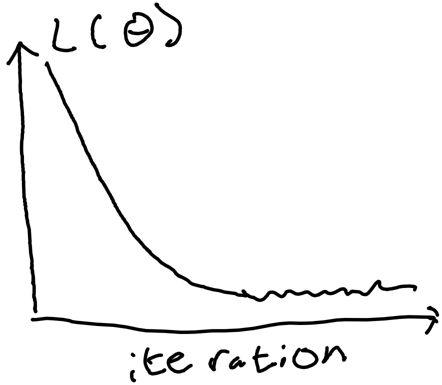
$$\mathbb{E}_i[\nabla_\theta L_i(\theta)] = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta L_i(\theta)$$

Stochastic gradient descent (SGD)

In SGD, we are optimizing a different function at each iteration.



progress can stall as we approach a minimum of $L(\theta)$



One option: Use a "learning rate schedule" → different η at each iteration t.

$$\eta(t) = \eta \quad \text{(constant)}$$

$$\eta(t) = \eta_i \quad \text{if } t_i \leq t < t_{i+1} \quad \text{(piecewise)}$$

$$\eta(t) = \eta_0 e^{-\lambda t} \quad \text{(exponential)}$$

Another option: Use a minibatch.

Sample B examples randomly and use their average gradient.

$$\Theta \leftarrow \Theta - \eta \frac{1}{B} \sum_{i=1}^{B} \nabla_\Theta L_i(\Theta)$$

Note: SGD can be more efficient than BGD.

(if $\nabla_\theta L_i$ is generally "aligned" $\nabla_\theta L(\theta)$)

Note: SGD can be noisy

Note: SGD is less parallelizable.

Adaptive Gradient Methods

Why might it be a bad idea to use the same η for every parameter?

~~ex~~ $\hat{y} = w^T x$

$$L(y, \hat{y}) = \tfrac{1}{2}\|y - \hat{y}\|^2$$

$$\frac{dL}{d\hat{y}} = y - \hat{y}$$

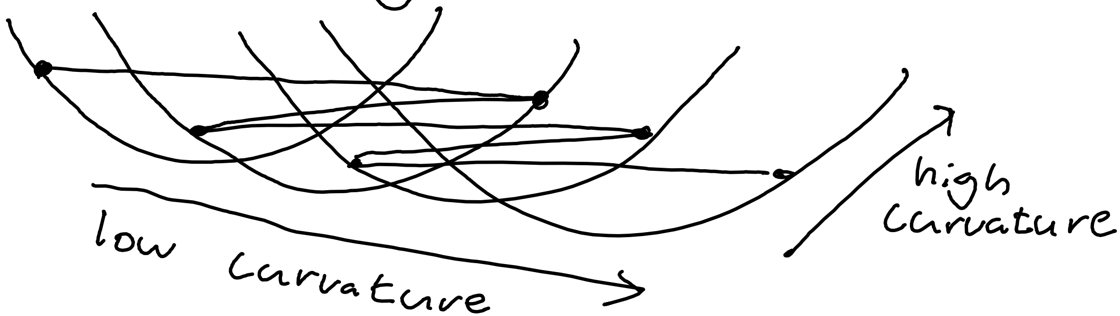
$$\frac{dL}{dw} = (y - \hat{y})\,x$$

Imagine that $|x_i| \gg |x_j|$

e.g. $x_i \sim \mathcal{N}(0, \sigma_i^2)$, $x_j \sim \mathcal{N}(0, \sigma_j^2)$, $\sigma_i \gg \sigma_j$

Then we will often have $\left|\frac{dL}{dw_i}\right| \gg \left|\frac{dL}{dw_j}\right|$

Descending in a valley:



To avoid divergence in "steep" dimensions, use a small η. But then progress is slow in "gradual" dimensions.

Idea: Use a different effective learning rate for each parameter based on the history of gradients for that parameters.

Simplest: Momentum

$v_t \leftarrow \beta v_{t-1} + g_t$ → gradient vector at timestept $\nabla_\theta L(\theta)$ → $\mathbb{R}^d$

$\hookrightarrow \in \mathbb{R}^d$

$\hookrightarrow$ "momentum" hyperparameter, $0 < \beta < 1$

$\theta_t \leftarrow \theta_{t-1} - \eta v_t$

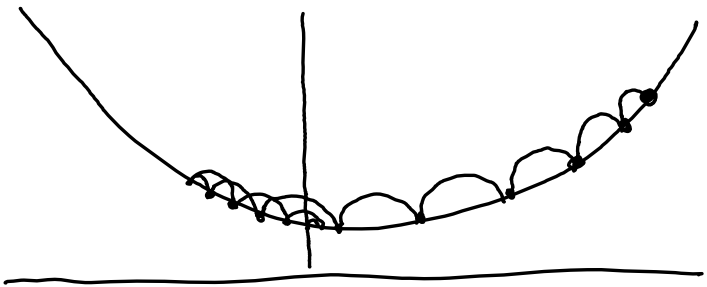
$\hookrightarrow$ learning rate

v_t will change more slowly than g_t. If g_t is consistent, v_t will grow in that direction.

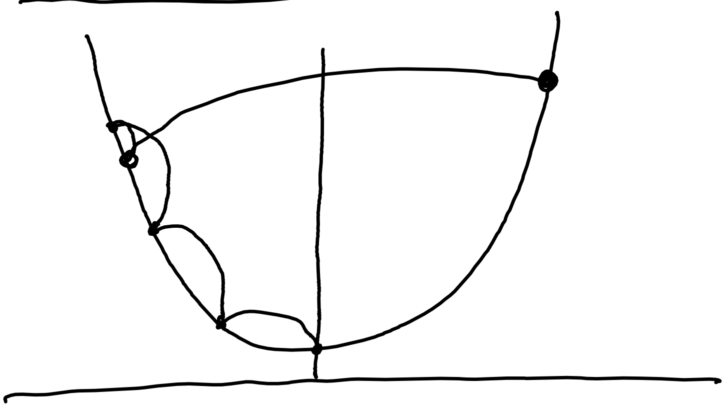
If g_t is inconsistent, it might "cancel out" changes.

$v_t \leftarrow \beta v_{t-1} + g_t$

(assume β is close to 1, 0.9 typical)



gradient points in the same direction → take bigger steps.



gradient direction is inconsistent → take smaller steps.

Adam:

$$v_t \leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t$$ ("first moment" estimate)

$$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2) g_t^2$$ ("second moment" estimate)

$$g_t' = \cancel{\frac{\eta v_t}{\sqrt{s_t} + \epsilon}}$$ small constant

$$\theta_t \leftarrow \theta_{t-1} - g_t'$$

$$v_1 = \beta_1 \underbrace{v_0}_{\to 0} + (1-\beta_1) g_1 = (1-\beta_1) g_1$$

$$v_2 = \beta_1 v_1 + (1-\beta_1) g_2 = \beta_1 (1-\beta_1) g_1 + (1-\beta_1) g_2$$

total rescaling: $\beta_1(1-\beta_1) + (1-\beta_1) = 1-\beta_1^2$

$$v_3 = \beta_1 (\beta_1 (1-\beta_1) g_1 + (1-\beta_1) g_2) + (1-\beta_1) g_3$$

total rescaling: $1-\beta_1^3$

Total gradient scale is $1-\beta_1^t$

Typically, β_1 and β_2 are close to 1

$1-\beta_1^t$ and $1-\beta_2^t$ will start near $\emptyset$ and grow slowly. So v_t and s_t are "biased" towards $\emptyset$.

To correct: Divide out $1-\beta_1^t$ and $1-\beta_2^t$

$$v_t \leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t \qquad \hat{v}_t = \frac{v_t}{1-\beta_1^t}$$

$$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2) g_t^2 \qquad \hat{s}_t = \frac{s_t}{1-\beta_2^t}$$

$$g_t' = \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t} + \varepsilon}$$

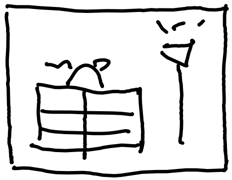
$$\theta_t \leftarrow \theta_{t-1} - g_t'$$

Adam rescales gradients so that you use (approximately) the average gradient divided by the average magnitude. Hopefully helps us use the learning rate for all parameters and all problems!

$$\beta_1 = 0.9$$
$$\beta_2 = 0.999$$
$$\eta = 0.001$$
$$\varepsilon = 10^{-8}$$

<u>Note</u>: Have to store v_t and s_t ...

Convolutional Neural Networks



Does this image contain a cat?

1. Translation invariance
   (respond similarly everywhere)

2. Locality
   (only consider a "small" region)

$$h = u + Wx$$

$$h_m = u_m + \sum_n W_{m,n} x_n$$

$$H_{i,j} = U_{i,j} + \sum_k \sum_l W_{i,j,k,l} X_{k,l}$$

$i \cdot j = m,\ k \cdot l = n$

$$H_{i,j} = U_{i,j} + \sum_a \sum_b V_{i,j,a,b} X_{i+a,j+b}$$

can be positive or negative to go over the image

Translation invariance:

$$H_{i,j} = U + \sum_a \sum_b V_{a,b} X_{i+a,j+b}$$

Locality:

"kernel"
"filter"

$$\underbrace{H_{i,j} = U + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V_{a,b} X_{i+a,j+b}}_{\text{Convolution!}}$$

How much V is present at i,j?