Wednesday, February 3, 2021 10:01 AM

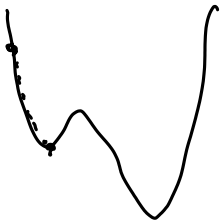
Lecture 5

Stochastic Gradient Descent

- Vanilla GD is not popular b/c
  - full gradient is expensive to compute $\mathcal{L} = \sum_{i \in Data} L_i(f(x_i; w), y_i)$

compute: $\frac{\partial \mathcal{L}}{\partial w} = \sum_{i \in Data} \frac{\partial L_i}{\partial w}$

- have to sum over full dataset for every weight update.

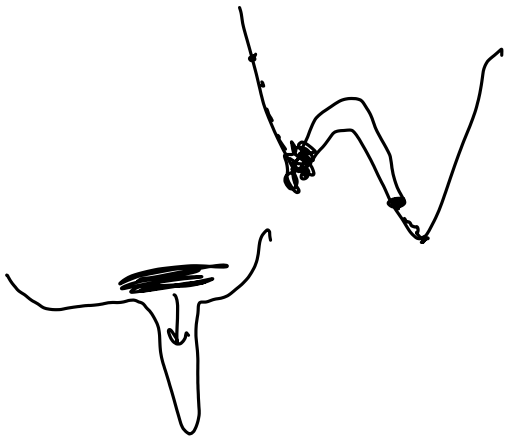


- can easily get stuck in a local minimum.

Wednesday, February 3, 2021 10:31 AM

SGD solves both problems in same way.

IDEA: allow gradients to be noisy



How?

Compute gradients over subset of data.

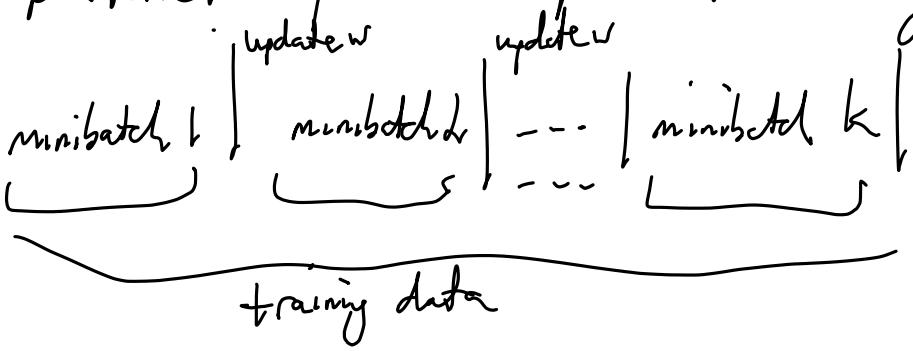
- Pure SGD: one datapoint
- minibatch SGD: chunk of data.

$\nabla_w L_i$

$\delta w = -\eta \frac{\partial L_i}{\partial w}$

$\delta w = -\eta \sum_{i=i_{start}}^{i_{start}+N_{batch}} \frac{\partial L_i}{\partial w}$

preferred in part b/c computational efficiency (parallelize over minibatch)



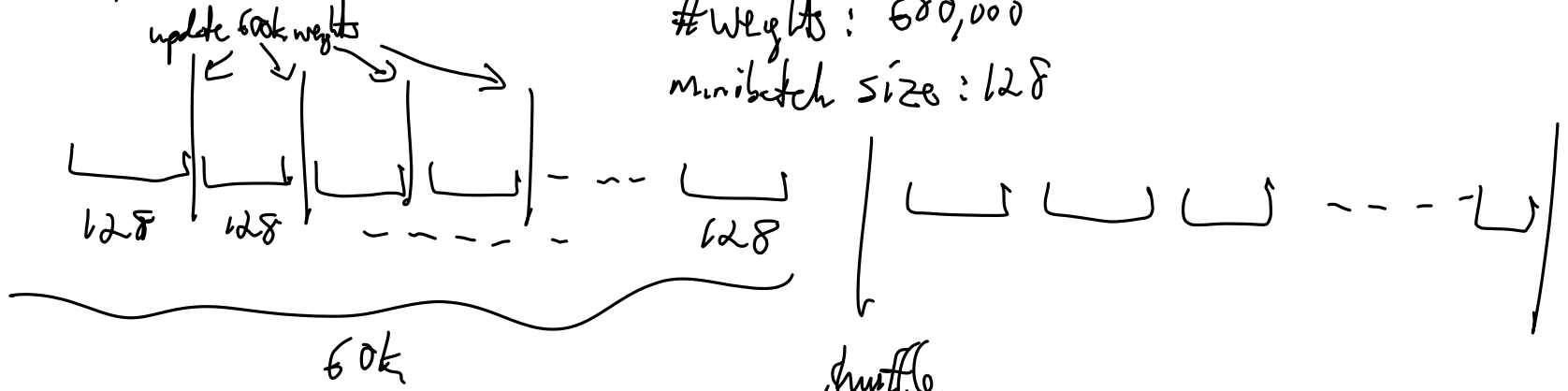
"1 epoch"
after each epoch, shuffle data!

Wednesday, February 3, 2021 10:49 AM

Example: MNIST DNN. Data: 60,000

#weights: 600,000

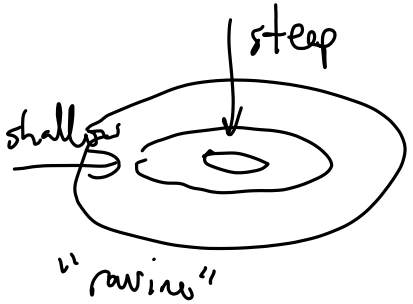
minibatch size: 128



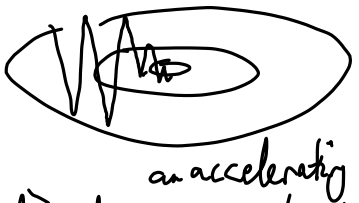
epoch 1 epoch 2

Variations on (S)GD

- Momentum



GD can get in trouble here



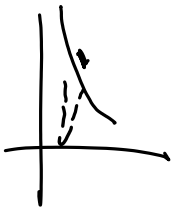
idea of momentum: give gradients a push based on their previous direction

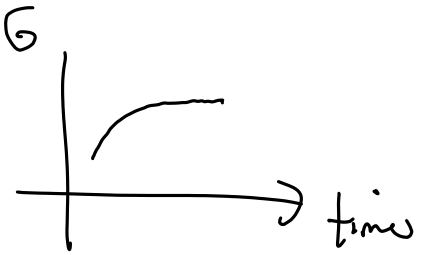
GD w/ momentum: $v_t = \gamma v_{t-1} + \eta \nabla_w L$

$w \to w - v_t$

- Adaptive learning rates!

Adagrad: $w \to w - \frac{\eta}{\sqrt{G}} \nabla_w L$ $\quad G = \sum_{\text{time steps}} (\nabla_w L)^2$





large gradients → smaller learning rate

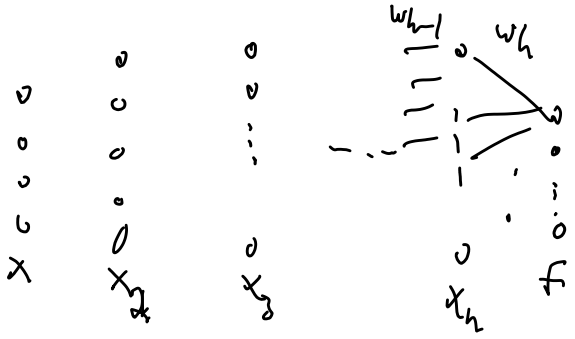
accumulating gradients → decreasing l.r.

☆ Adadelta: Adagrad w/ finite time window

$G = \sum_{\text{past time window}} (\nabla_w L)^2$

☆ Adam (Adaptive Moment estimation): "Adadelta w/ adaptive moments"

Wednesday, February 3, 2021 11:27 AM



Backpropation: how gradients are computed efficiently.

FF NN's have a recursive structure.

nested

$$f(x) = A\Big(w_h \cdot \underbrace{A\big(w_{h-1} \cdot A\big(\cdots A(w_1 x)\big)\cdots\big)}_{x_{h-1}}\Big) \leftarrow$$

x_h

$$x_h = A(w_{h-1} x_{h-1})$$
$$x_{h-1} = A(w_{h-2} x_{h-2})$$
$$\vdots$$
$$x_2 = A(w x)$$
$$x_1 = x.$$

$$\mathcal{L} = \sum_{i \in Data} L(f(x_i), y)$$

$$\frac{\partial \mathcal{L}}{\partial w_k} \longrightarrow \frac{\partial L}{\partial w_k} \longrightarrow \frac{\partial f(x_i; w)}{\partial w_k}$$

Let's work out a few examples

$$\frac{\partial f}{\partial w_h} = A'(w_h x_h) \cdot x_h \equiv G_h x_h$$

$$\frac{\partial f}{\partial w_{h-1}} = A'(w_h x_h) \cdot w_h A'(w_{h-1} x_{h-1}) x_{h-1} \equiv G_{h-1} x_{h-1}$$

$$\frac{\partial f}{\partial w_{h-2}} = A'(w_h x_h) \cdot w_h A'(w_{h-1} x_{h-1}) w_{h-1} A'(w_{h-2} x_{h-2}) x_{h-2} \equiv G_{h-2} x_{h-2}$$

$\vdots$

- compute gradients recursively

Backpropagation: $G_{k-1} = G_k W_k \sigma'(W_{k-1} X_{k-1})$

- matrix mult. — expensive
- enabled "end-to-end" training of NNs.

GPU's are key for deep learning.