Outline

1. Previous lecture recap: backpropagation, activations, intuition.
2. Optimizers.
3. Data normalization.
4. Regularization.

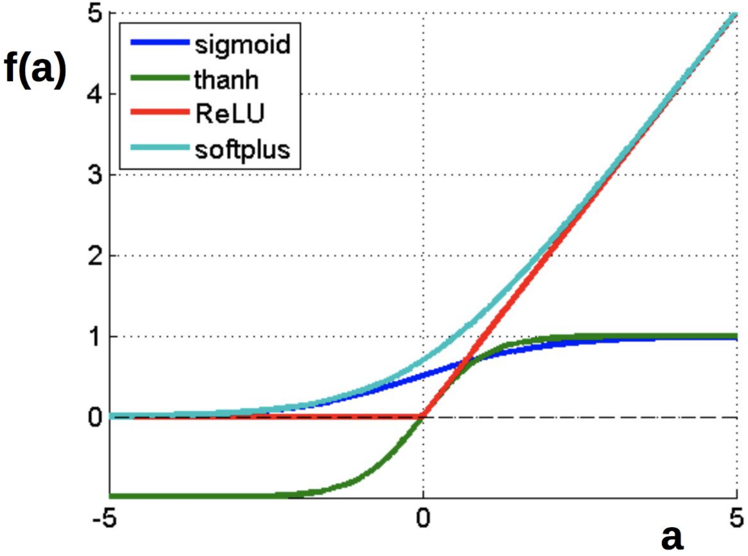
Once again: nonlinearities

$$f(a) = \frac{1}{1 + e^{-a}}$$

$$f(a) = \tanh(a)$$

$$f(a) = \max(0, a)$$

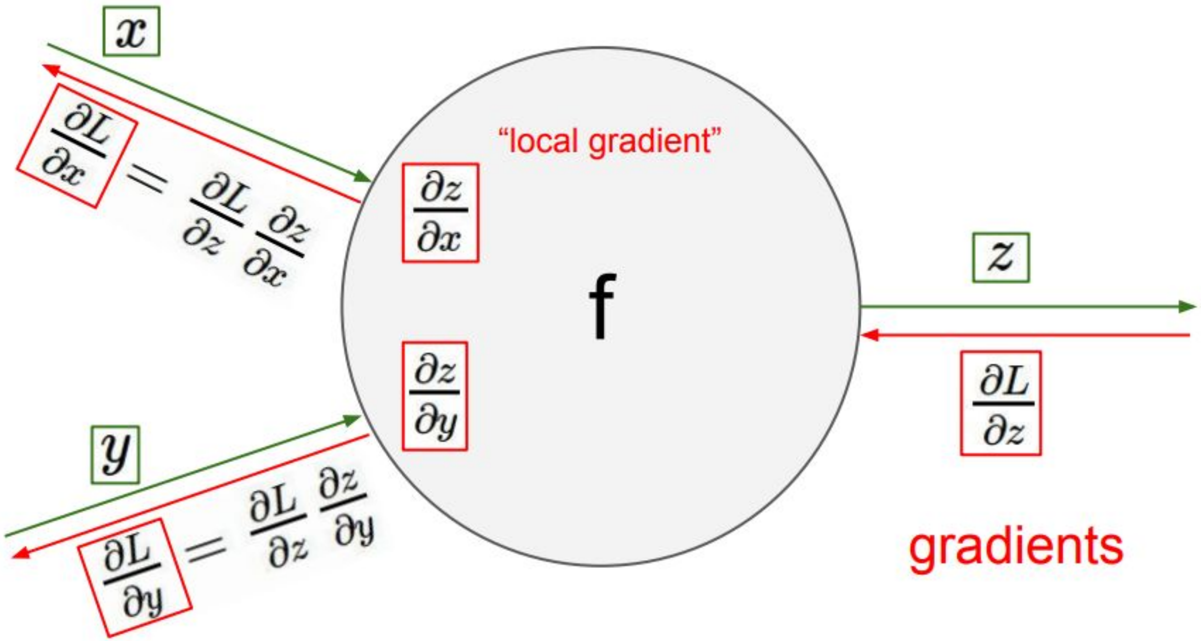
$$f(a) = \log(1 + e^a)$$



Backpropagation and chain rule

Chain rule is just simple math: $$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

Backprop is just way to use it in NN training.

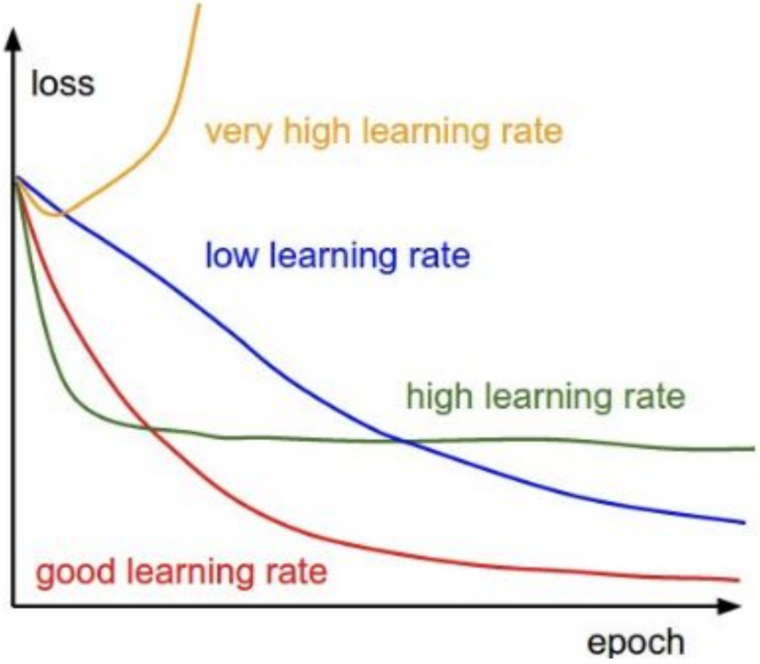


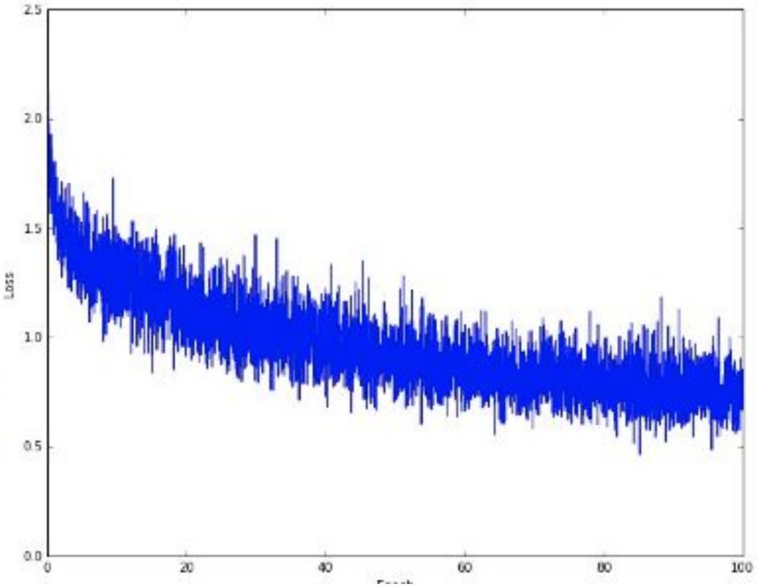
source: http://cs231n.github.

Optimizers

Stochastic gradient descent is used to optimize NN parameters.

$$x_{t+1} = x_t - \text{learning rate} \cdot dx$$



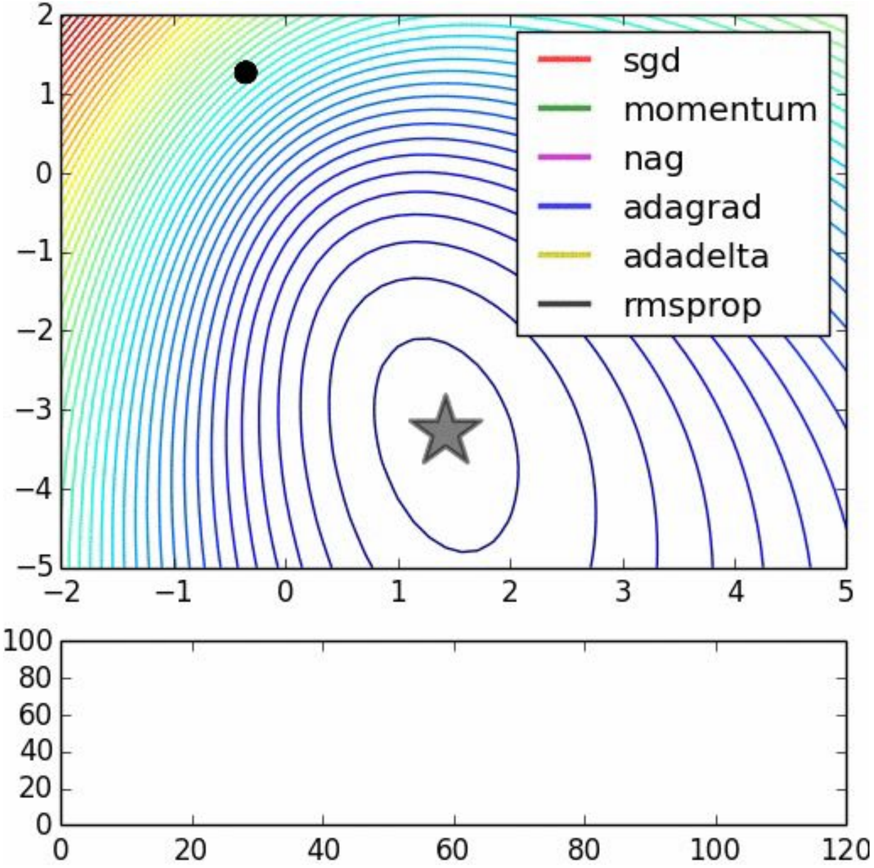


source: http://cs231n.github.io/neural-networks-3/

Optimizers

There are much more optimizers:

- Momentum
- Adagrad
- Adadelta
- RMSprop
- Adam
- ...
- even other NNs

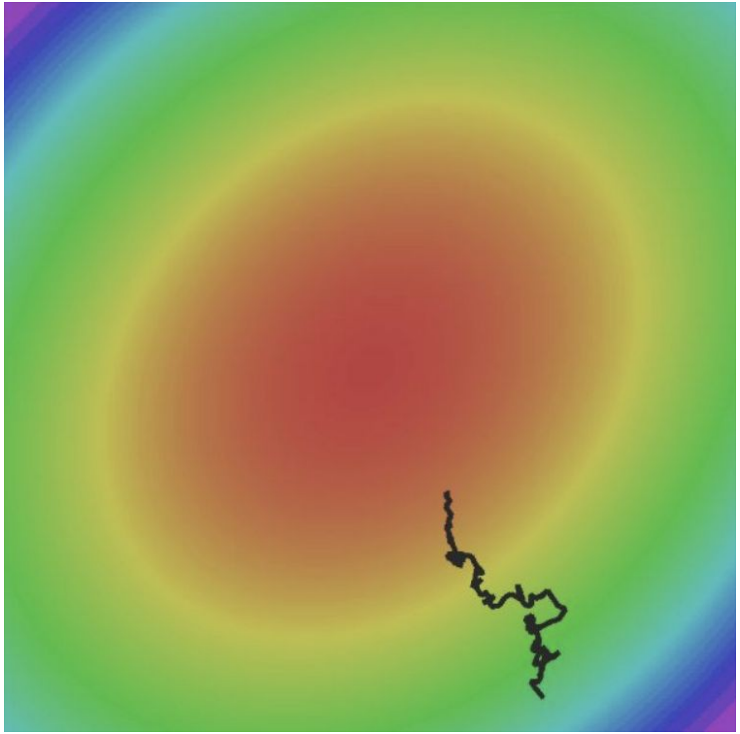


source: link

Optimization: SGD

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



Averaging over mini batches => noisy gradient

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

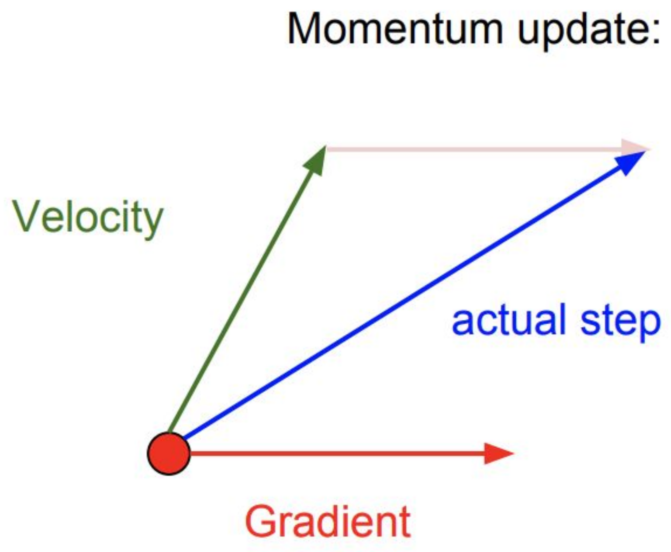
First idea: momentum

Simple SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD with momentum

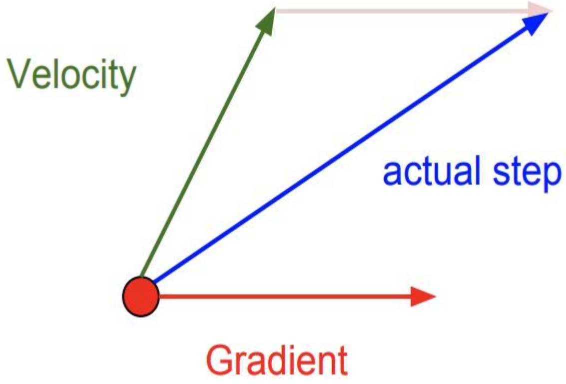
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$



source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

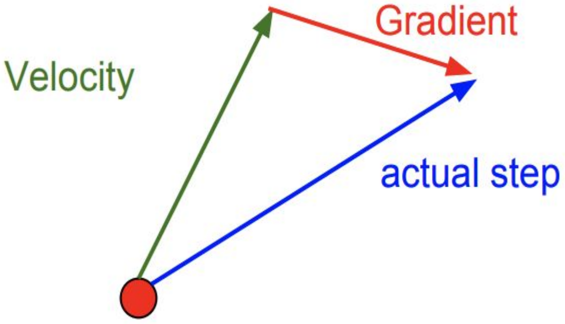
Nesterov momentum

Momentum update:



$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

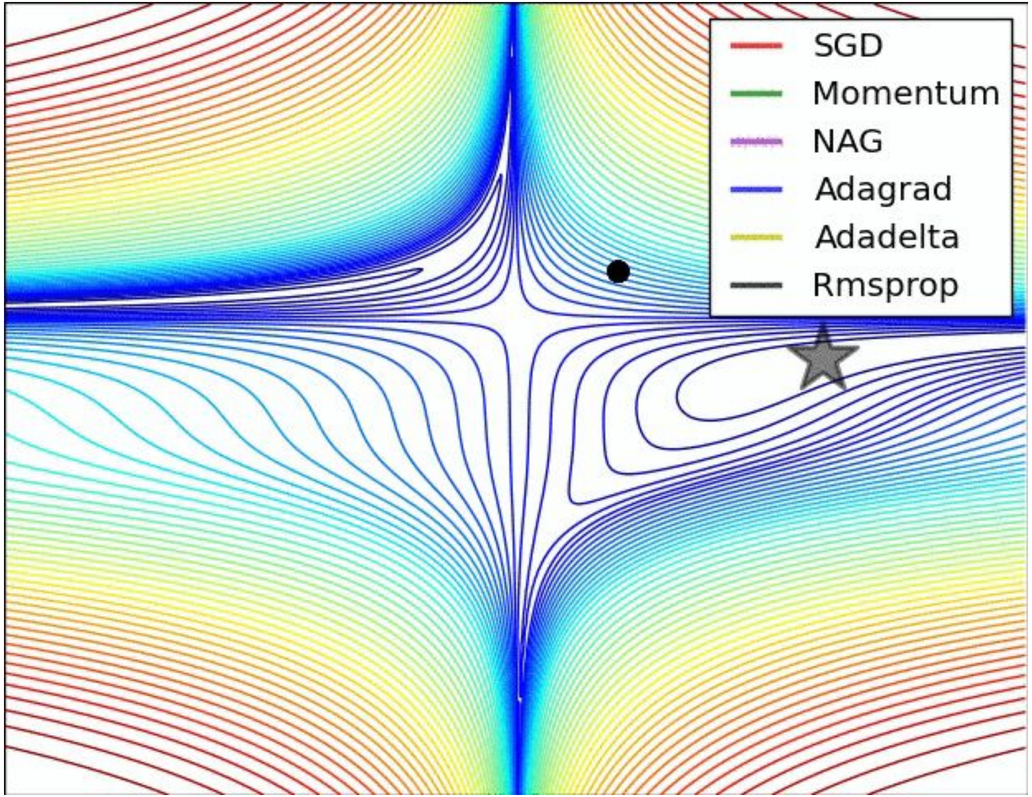
Nesterov Momentum



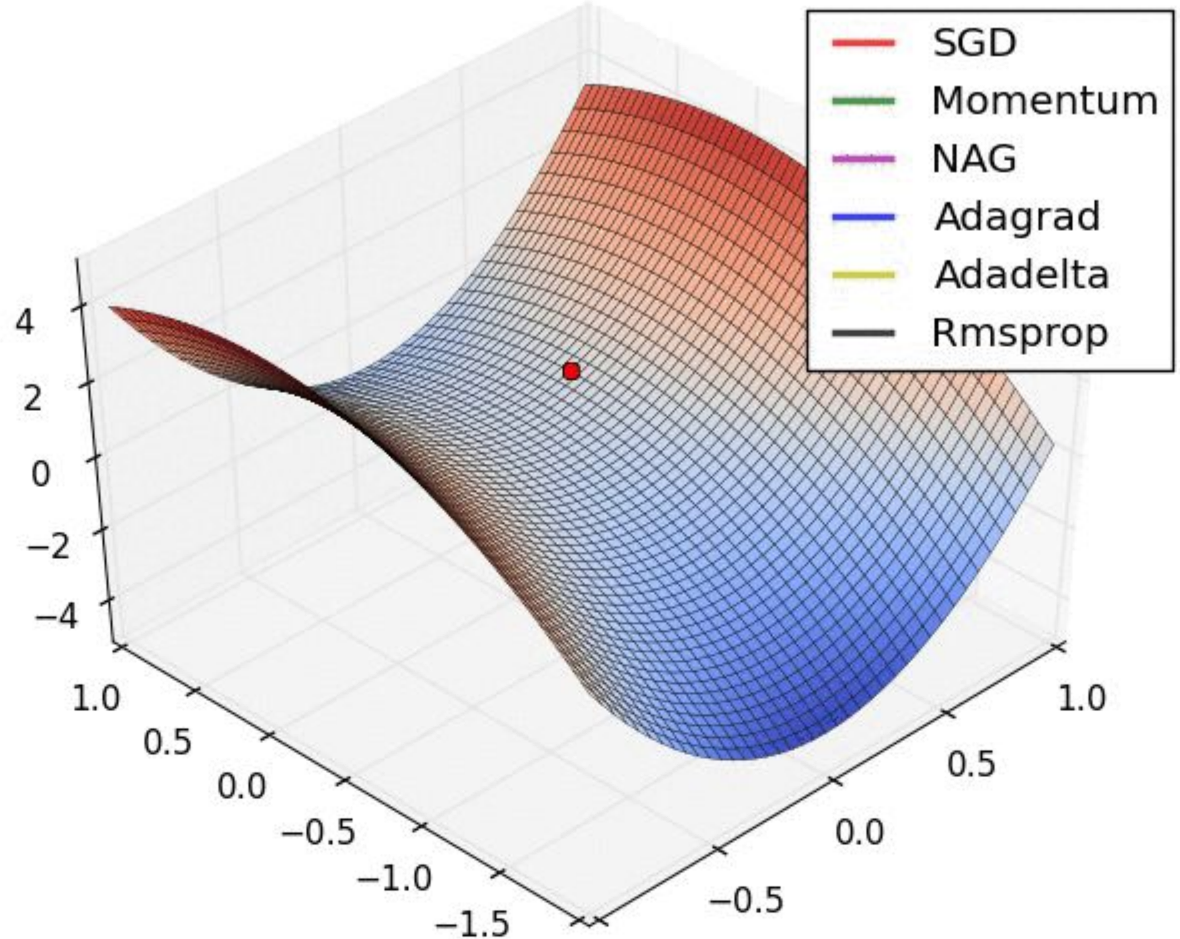
$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Comparing momentums



source:



source:
https://ruder.io/content/images/2016/09/saddle_point_evaluation_optimizers.gif

Second idea: different dimensions are different

Adagrad: SGD with cache

$$\text{cache}_{t+1} = \text{cache}_t + (\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\text{cache}_{t+1} + \varepsilon}$$

Second idea: different dimensions are different

Adagrad: SGD with cache

$$\text{cache}_{t+1} = \text{cache}_t + (\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\text{cache}_{t+1} + \varepsilon}$$

Problem: gradient fades with time

Second idea: different dimensions are different

Adagrad: SGD with cache

$$\text{cache}_{t+1} = \text{cache}_t + (\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\text{cache}_{t+1} + \varepsilon}$$

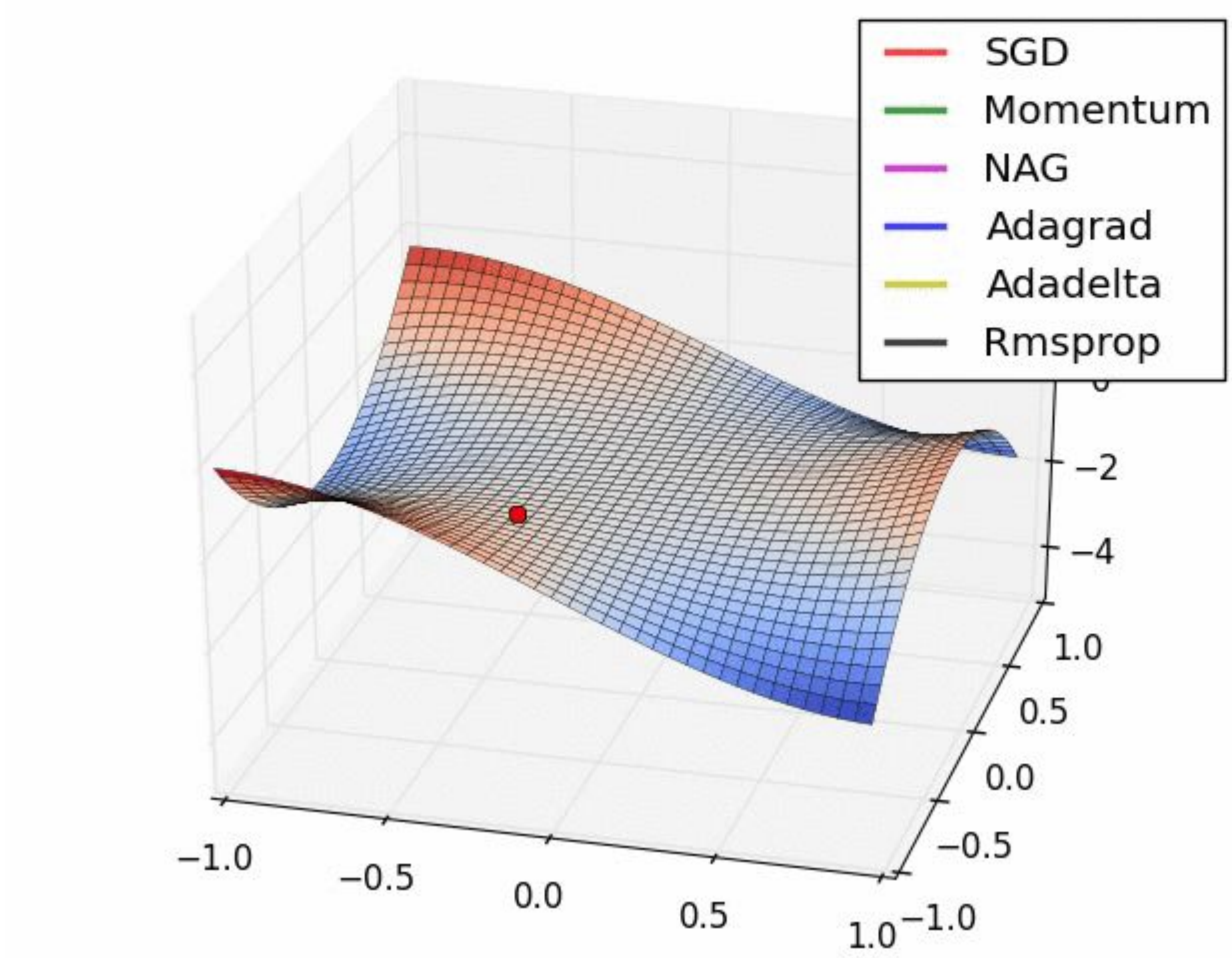
RMSProp: SGD with cache with exp. Smoothing

$$\text{cache}_{t+1} = \beta \text{cache}_t + (1 - \beta)(\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\text{cache}_{t+1} + \varepsilon}$$

Slide 29 Lecture 6 of Geoff Hinton's Coursera class

http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf



source: https://imgur.com/a/Hqolp#NKsFHJb

Adam

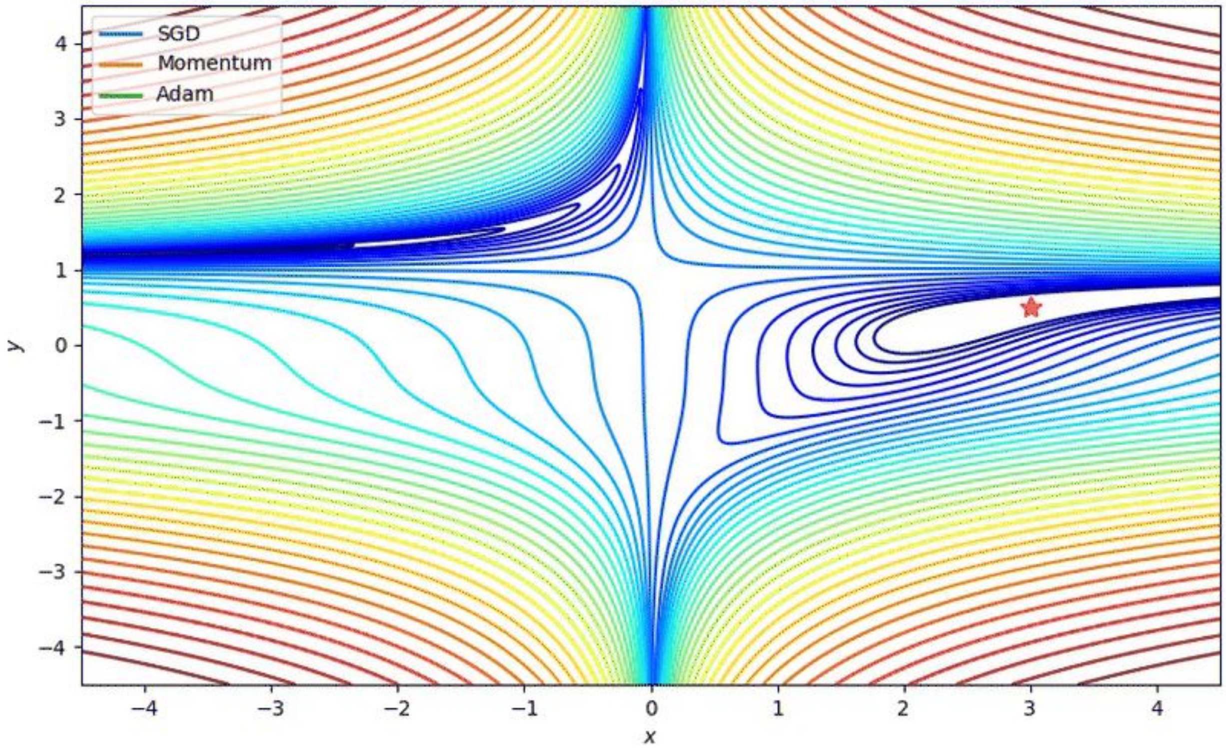
Let’s combine the momentum idea and RMSProp normalization:

$$v_{t+1} = \gamma v_t + (1 - \gamma)\nabla f(x_t)$$
$$\text{cache}_{t+1} = \beta \text{cache}_t + (1 - \beta)(\nabla f(x_t))^2$$
$$x_{t+1} = x_t - \alpha \frac{v_{t+1}}{\text{cache}_{t+1} + \varepsilon}$$

Actually, that’s not quite Adam.

Adam full form involves bias correction term. See http://cs231n.github.io/neural-networks-3/ for more info.

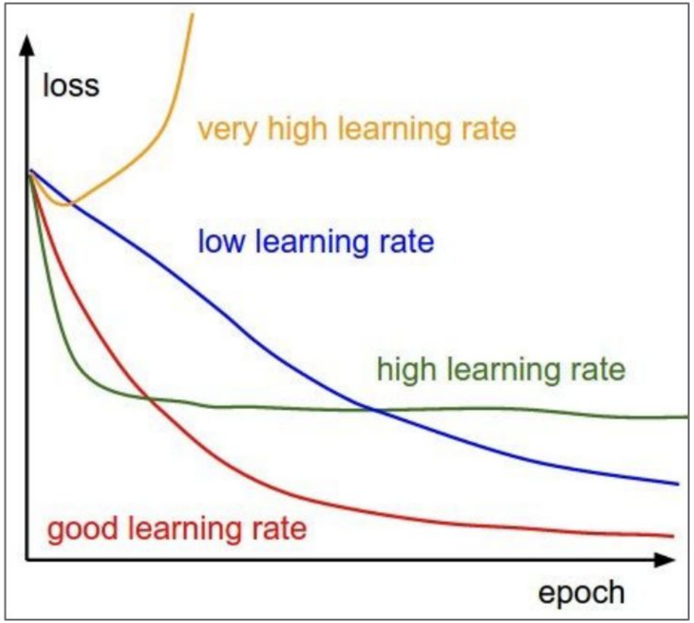
Comparing optimizers

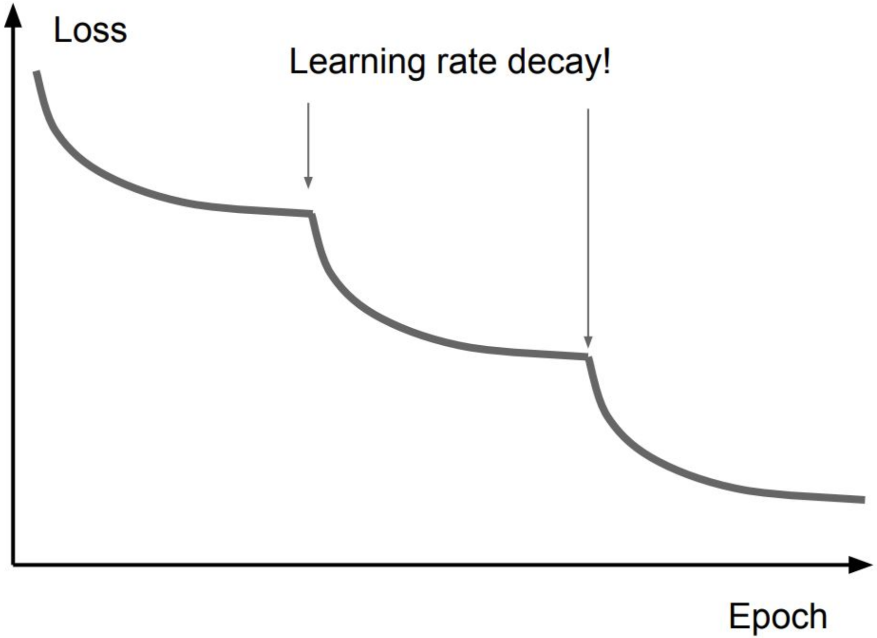


source:
https://joshvarty.com/2018/02/27/ltfn-7-a-quick-look-at-tensorflow-optimizers/

source: https://twitter.com/karpathy/status/801621764144971776

Once more: learning rate

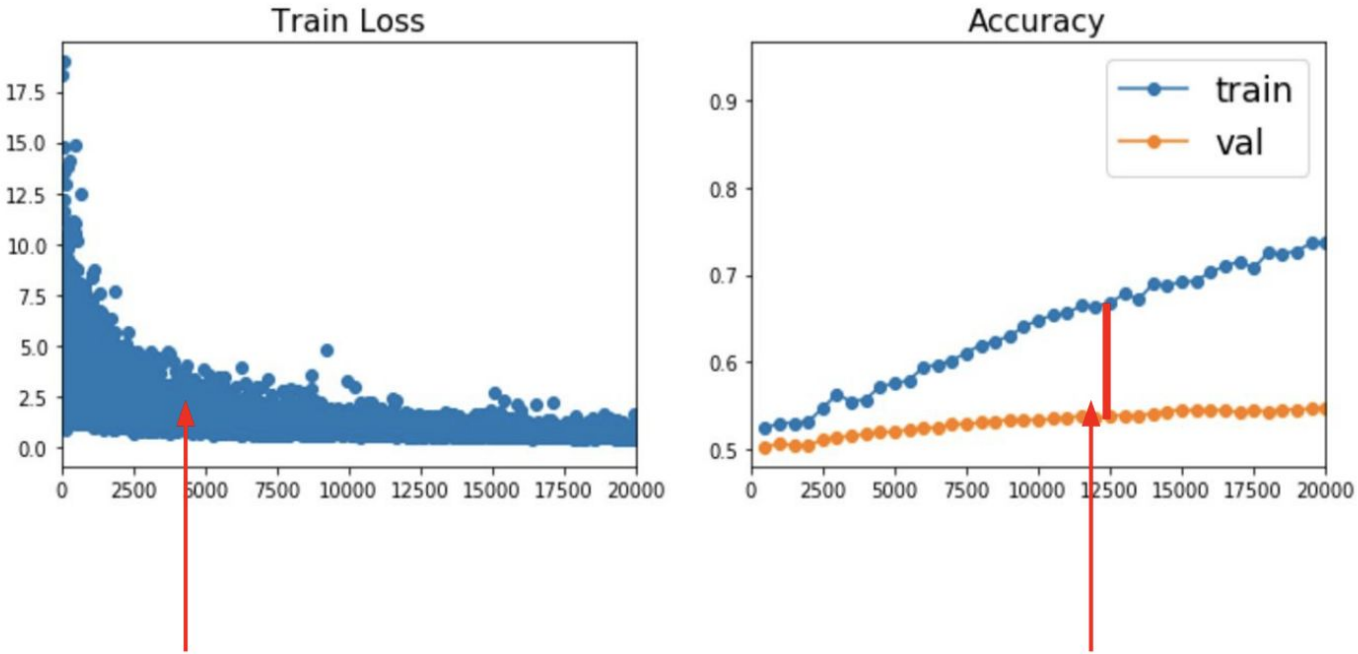




source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Sum up: optimization

- Adam is great basic choice
- Even for Adam/RMSProp learning rate matters
- Use learning rate decay
- Monitor your model quality

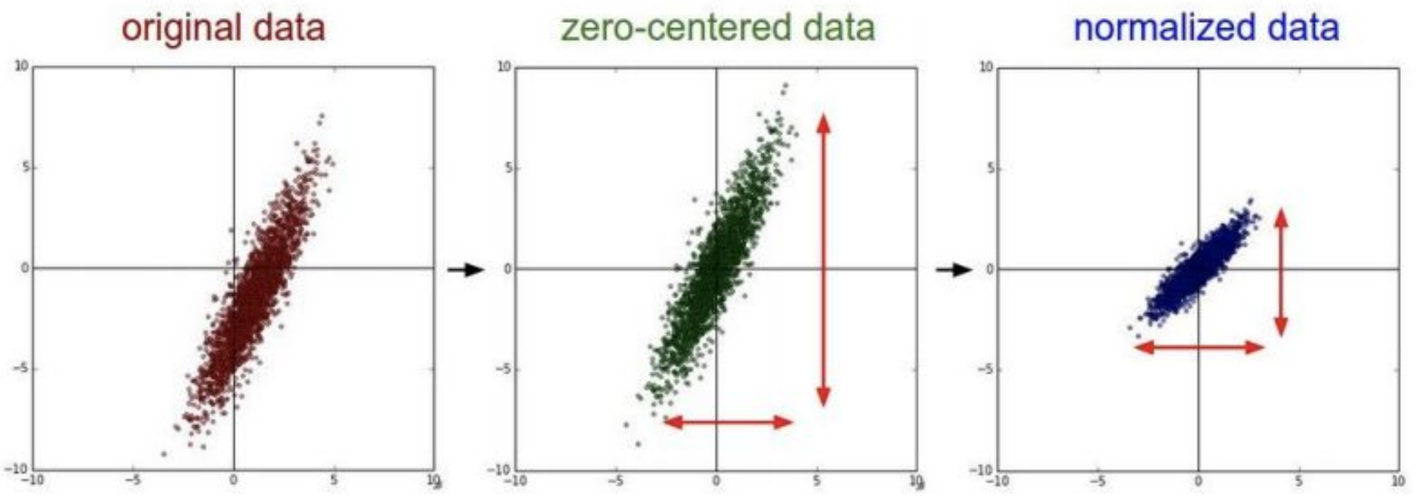


Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

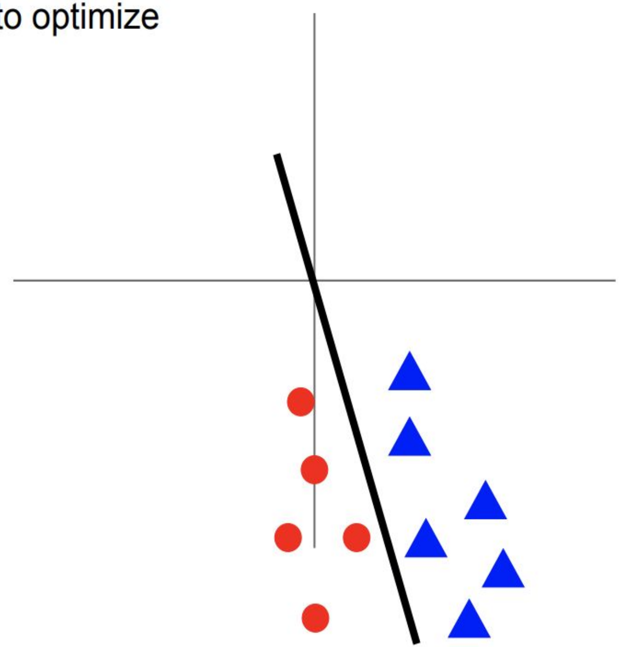
Data normalization



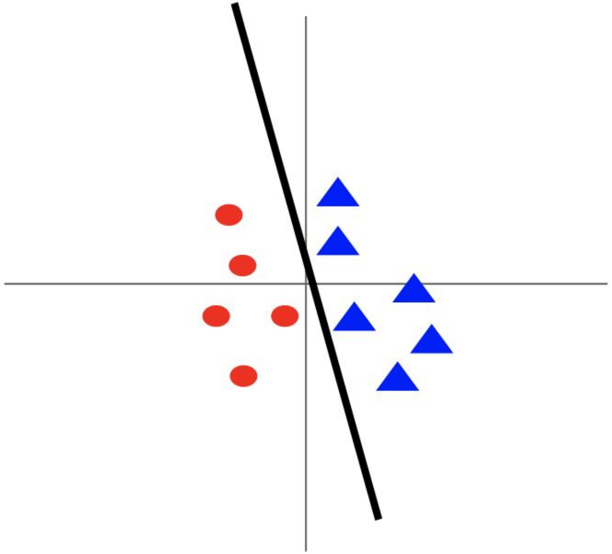
source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Data normalization

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After normalization: less sensitive to small changes in weights; easier to optimize



source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Weights initialization

- Pitfall: all zero initialization.

Weights initialization

- Pitfall: all zero initialization.
- Small random numbers.

Weights initialization

- Pitfall: all zero initialization.
- Small random numbers.
- Calibrated random numbers.

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_i^n w_i x_i)$$

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= (n\mathrm{Var}(w))\ \mathrm{Var}(x)$$

Thanks for attention!

Questions?

girafe
ai