Optimization and regularization in Deep Learning

Moscow, Fall 2020

Radoslav Neychev

Outline

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

Recap: Deep Learning basics

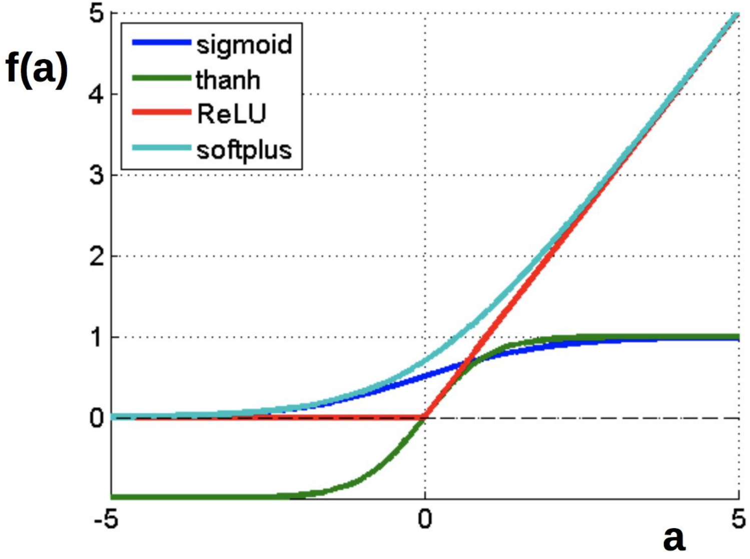
Once more: nonlinearities

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

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

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

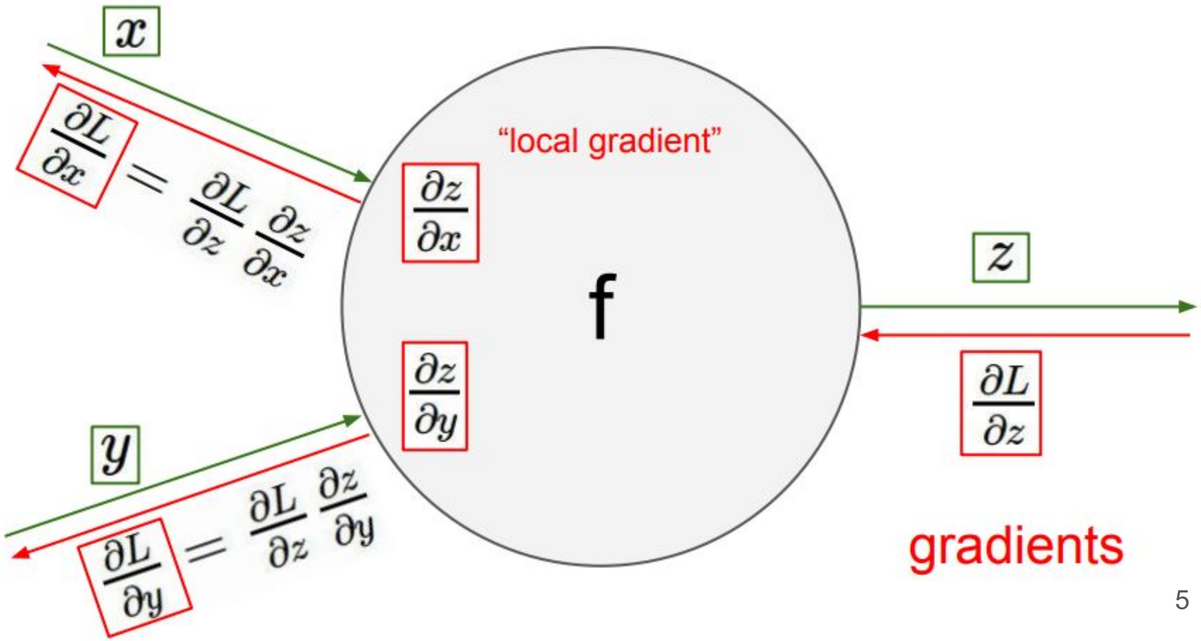
$$f(a) = \log(1 + e^{a})$$



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.

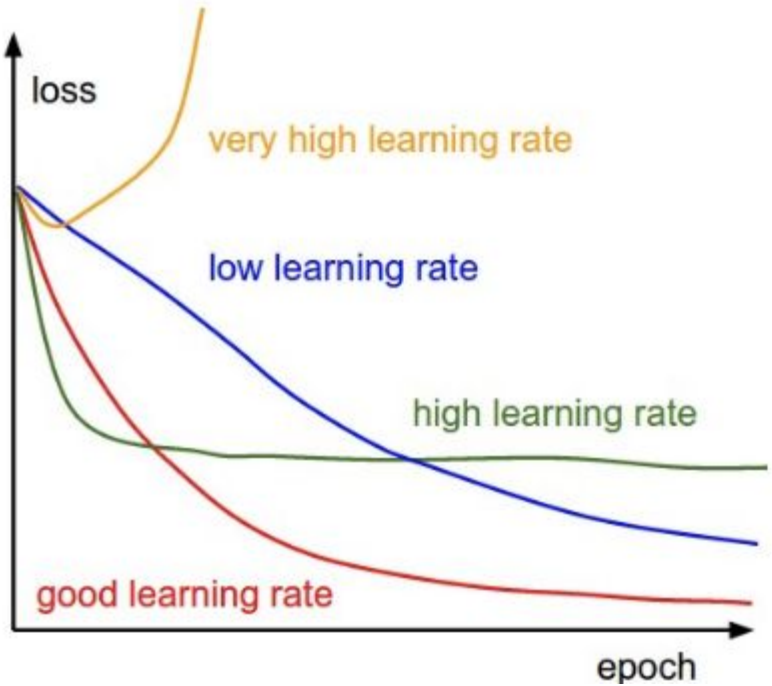


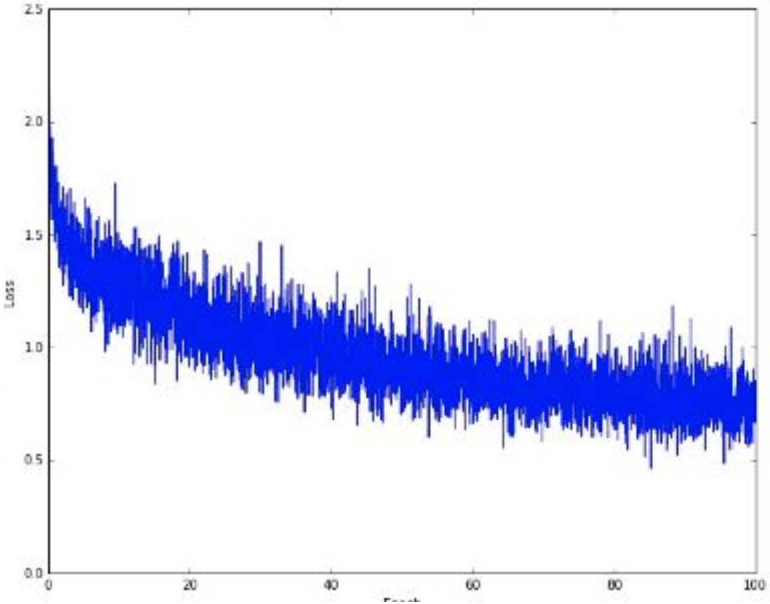
source: http://cs231n.github.io

Optimizers

Stochastic gradient descent is used to optimize NN parameters.

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





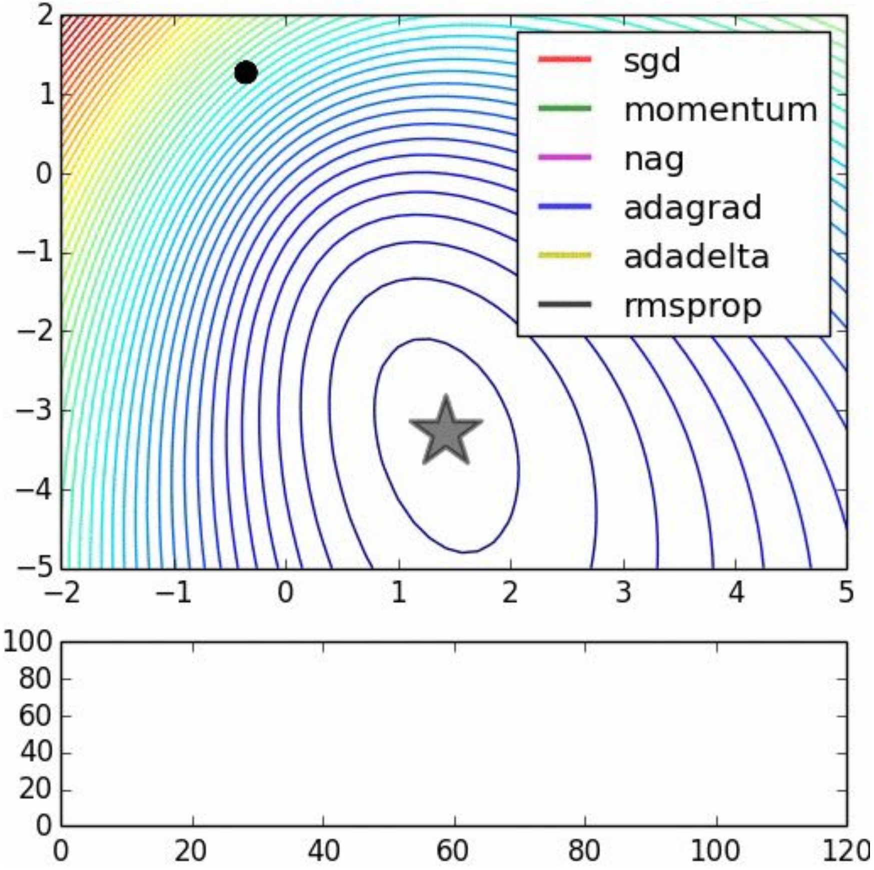
source: http://cs231n.github.io/neural-networks-3/

Optimization: SGD upgrades

Optimizers

There are much more optimizers:

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



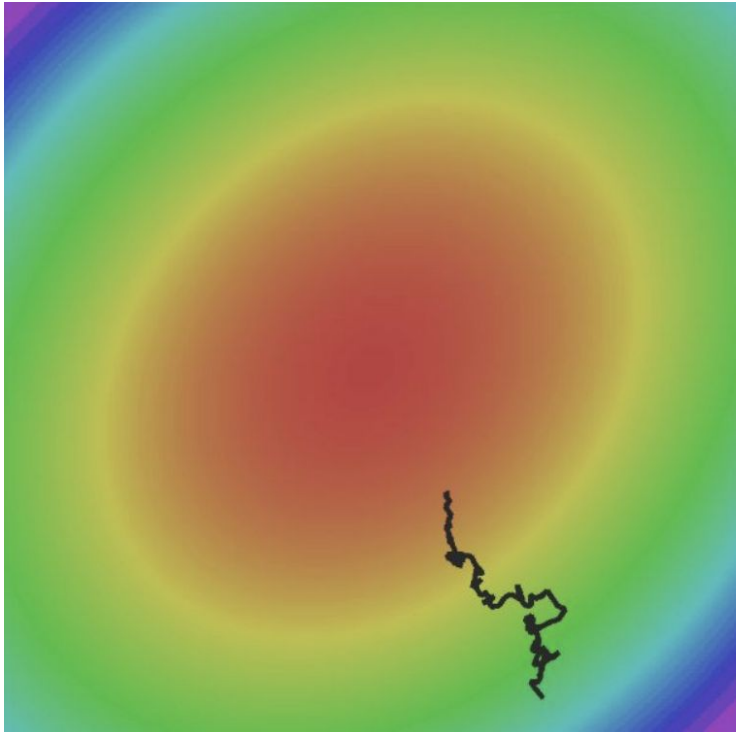
source: [link]

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 too small batches leads to noisy gradient



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

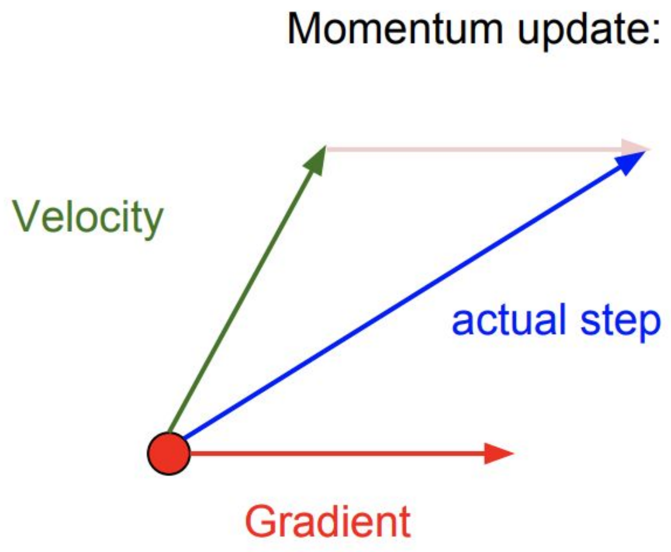
First idea: momentum

Simple SGD

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

SGD with momentum

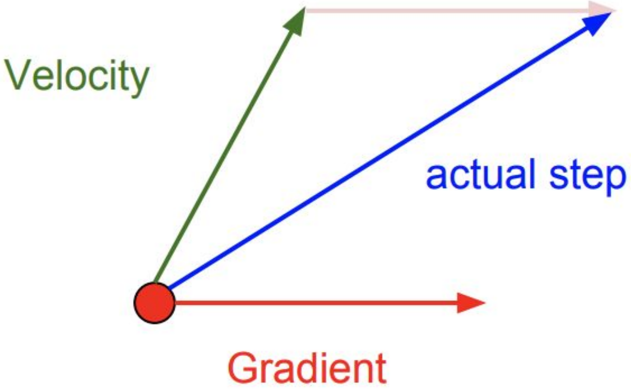
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$



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

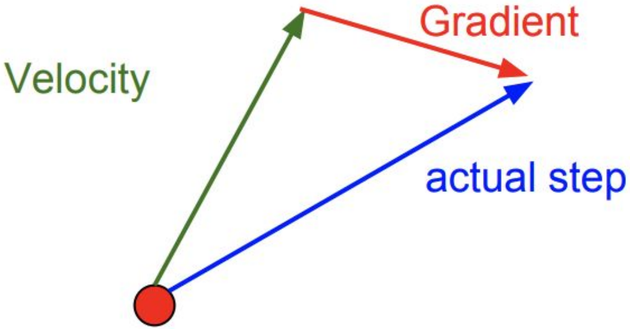
Nesterov momentum

Momentum update:



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

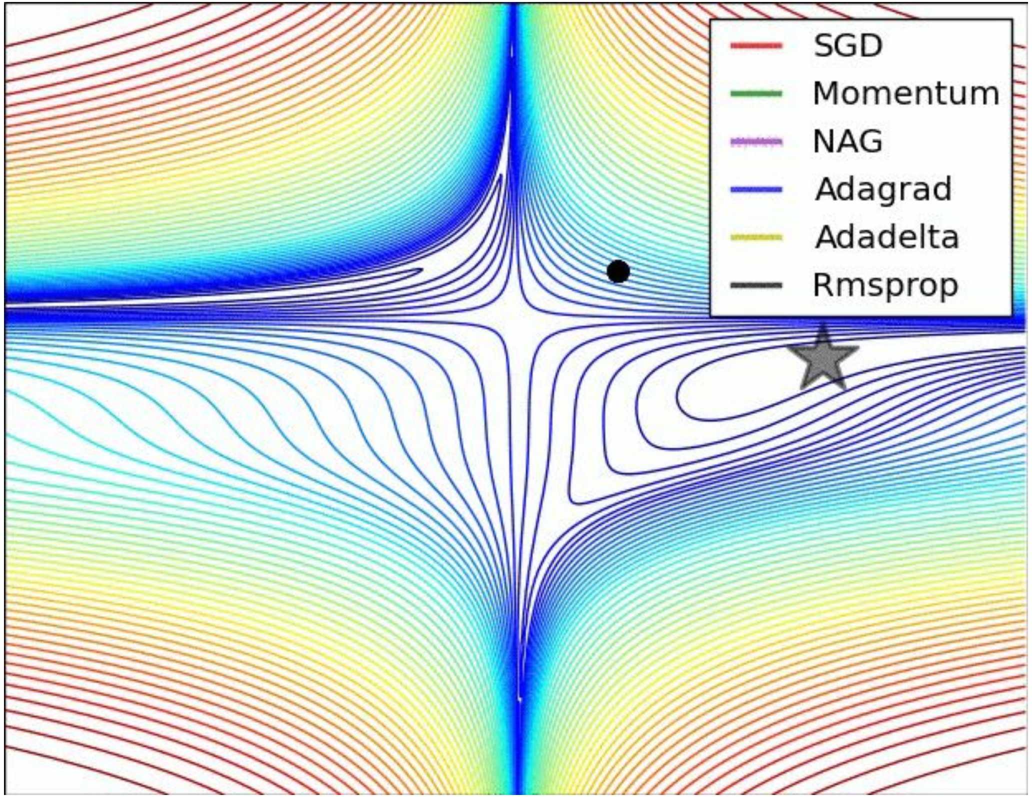
Nesterov Momentum



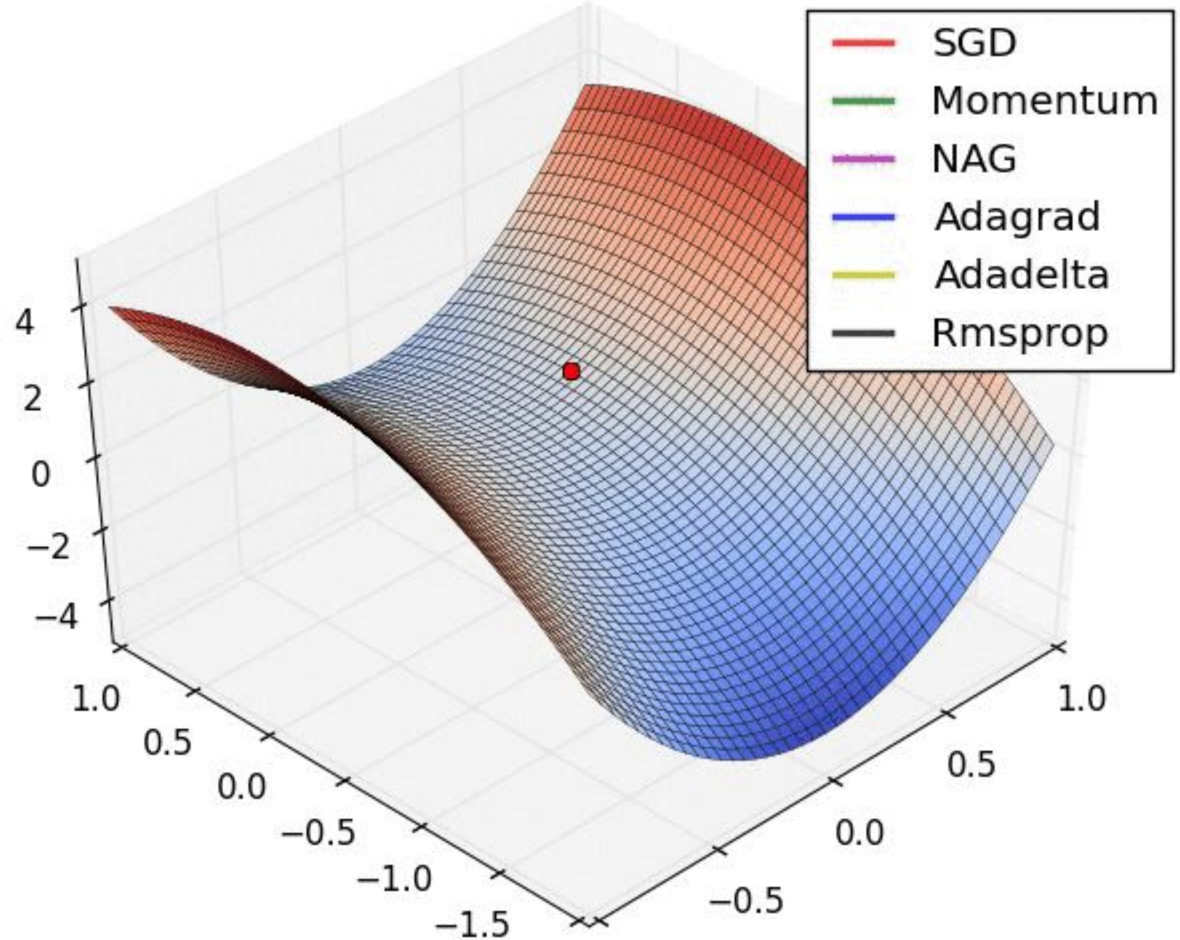
$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$

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

Comparing momentums



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



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

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

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

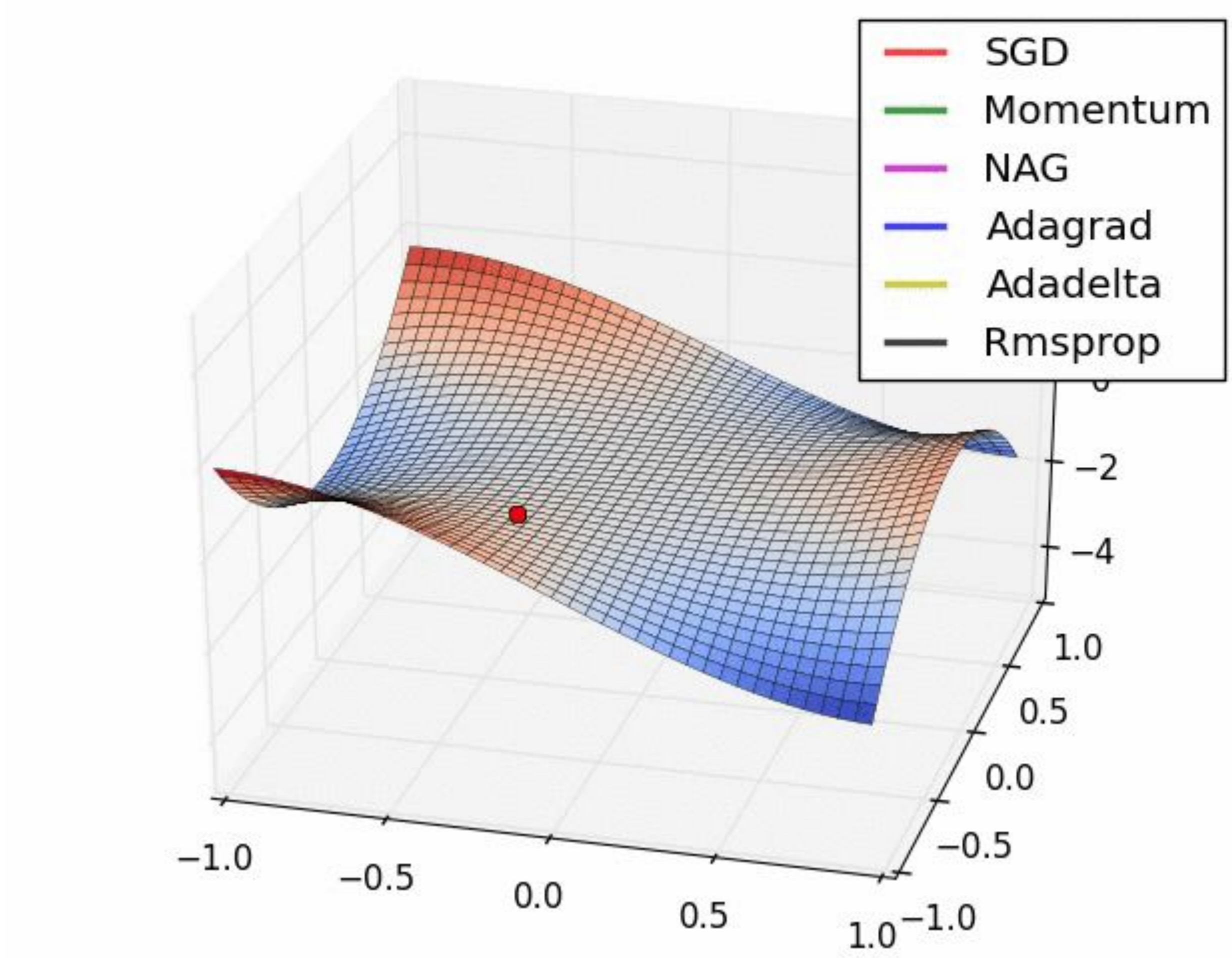
RMSProp: SGD with cache with exp. Smoothing

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

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

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:

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

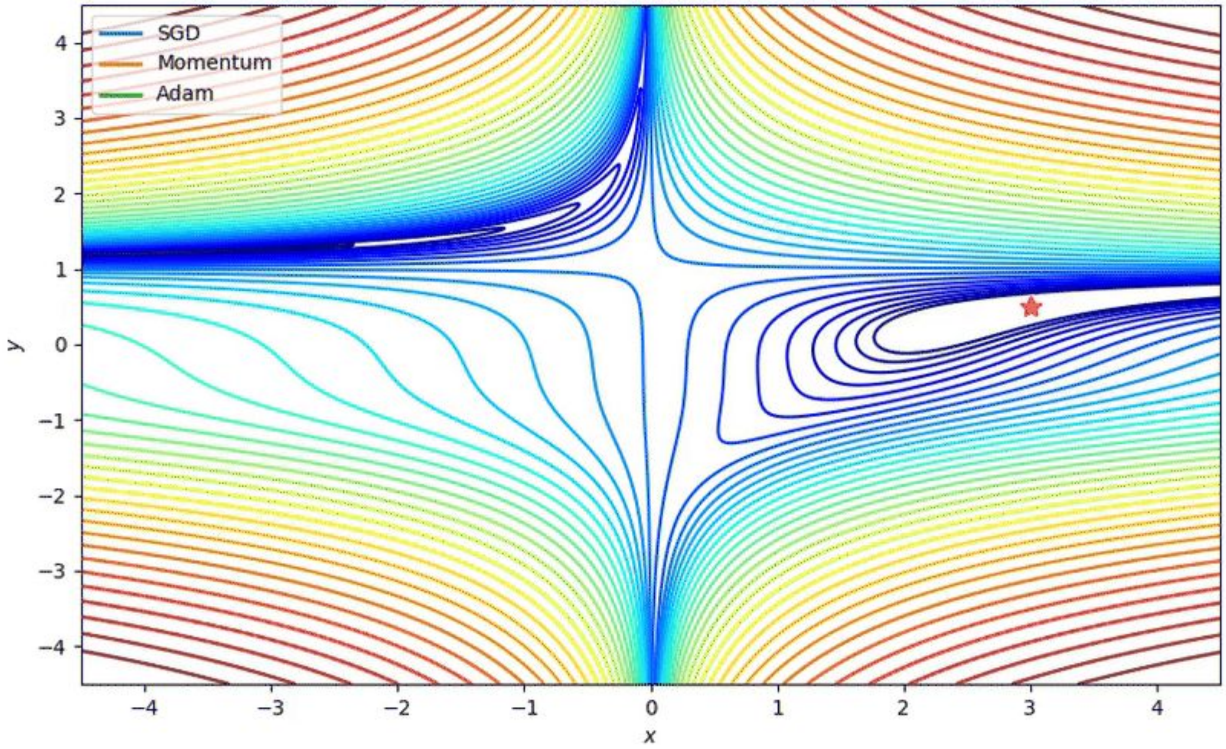
$$\text{cache}_{t+1} = \beta \text{cache}_t + (1-\beta)(\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{v_{t+1}}{\text{cache}_{t+1} + \varepsilon}$$

Actually, that's not quite Adam.

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

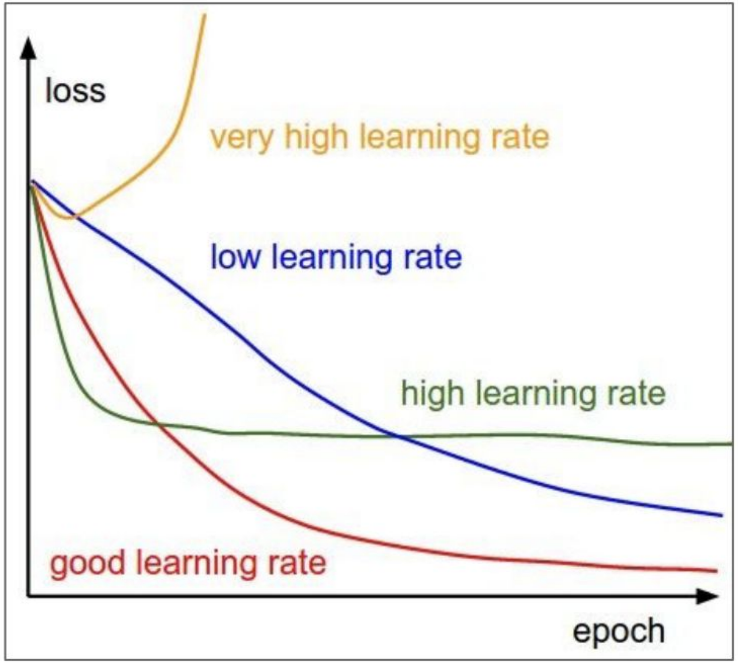
Comparing optimizers

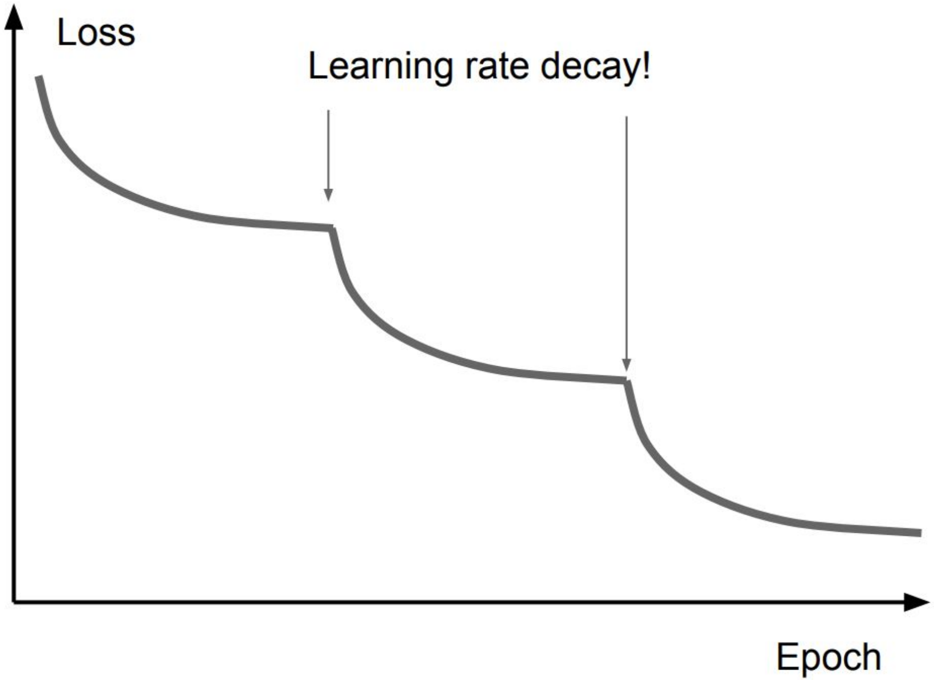


source: https://joshvarty.com/2018/02/27/ltfn-7-a-quick-look-at-tensorflow-optimizers/

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

Once more: learning rate



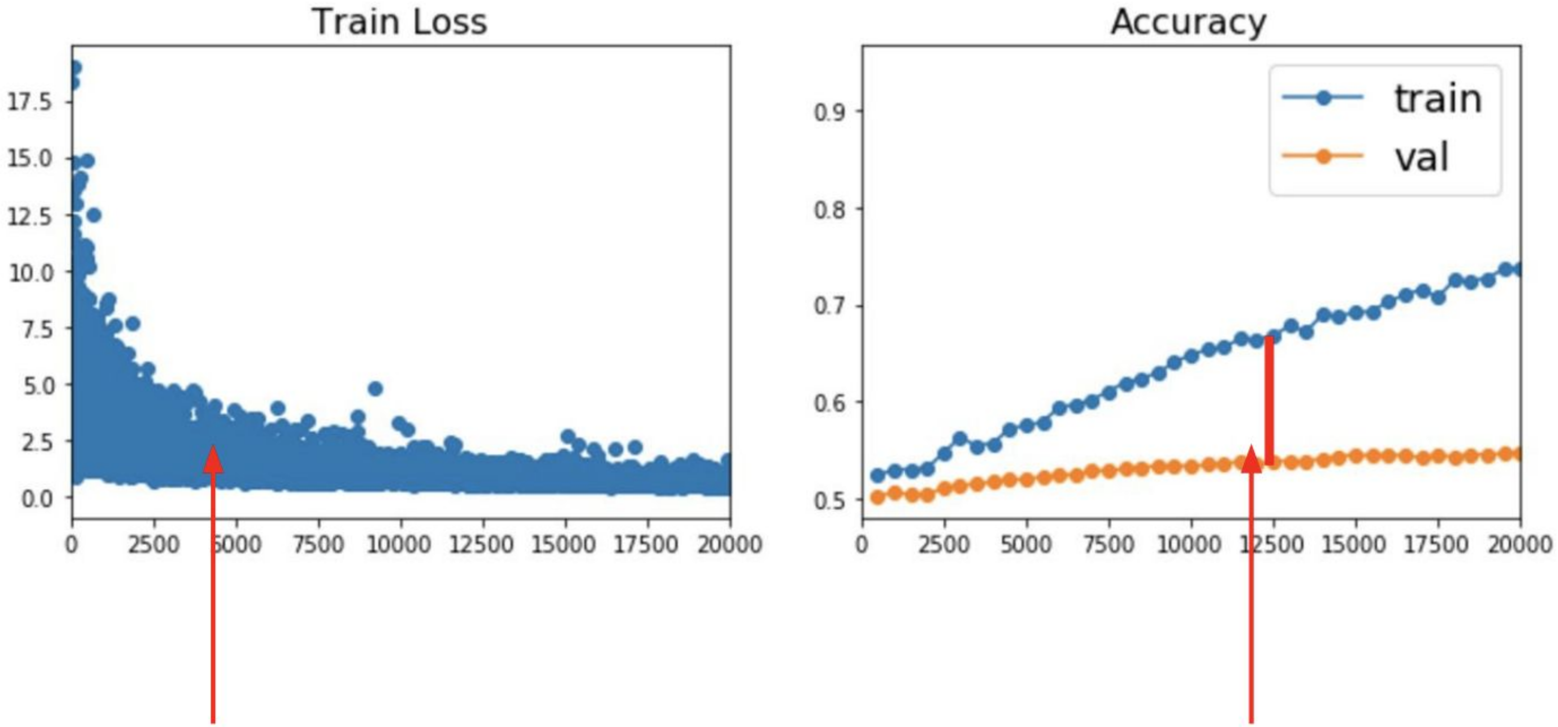


source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Sum up: optimization

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

Regularization in DL

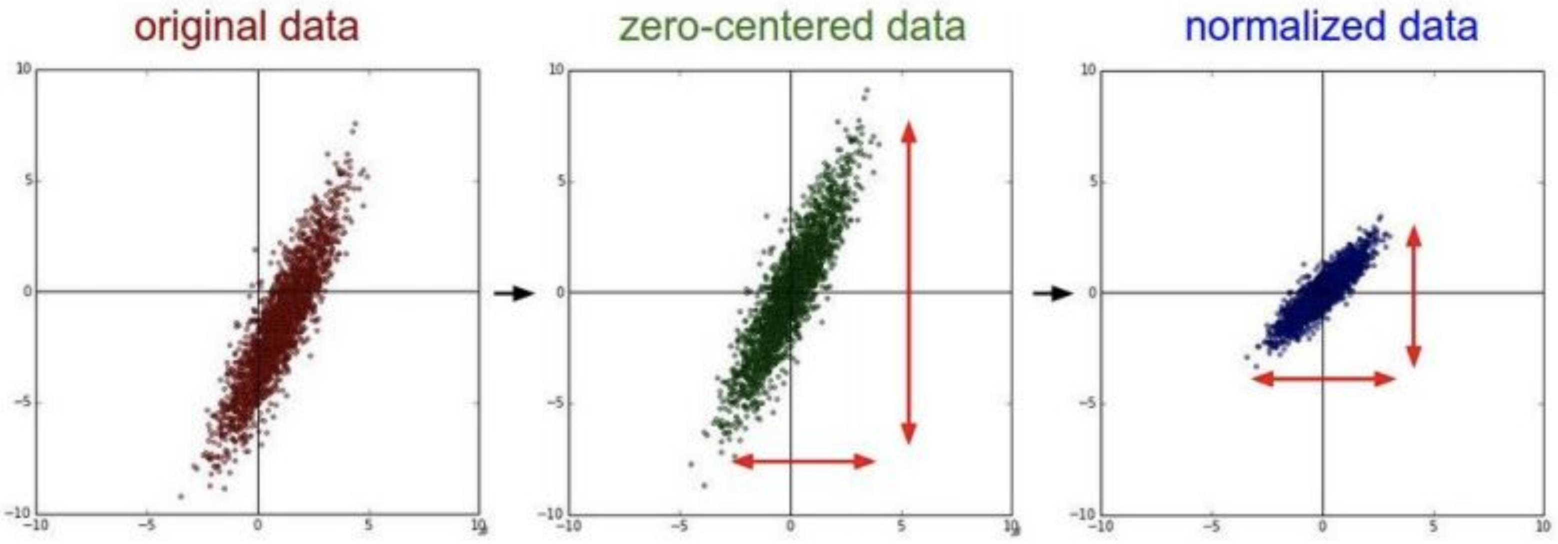


Better optimization algorithms help reduce training loss

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

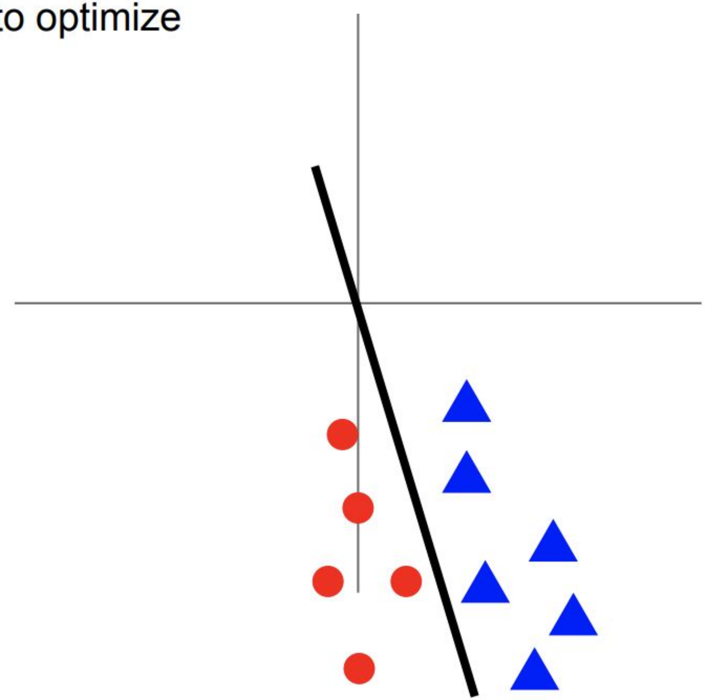
Data normalization



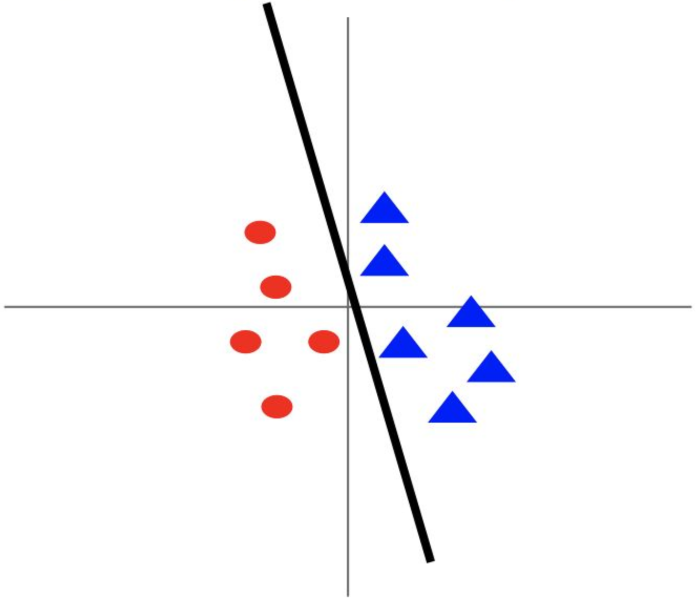
source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

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

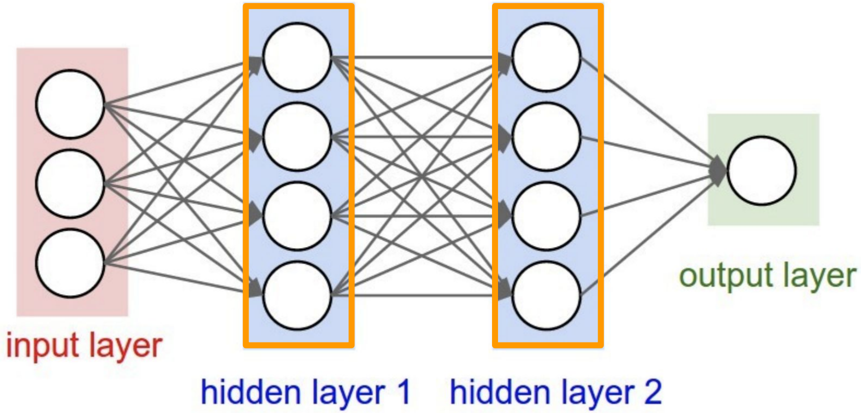


source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Batch normalization

Problem:

- Consider a neuron in any layer beyond first
- At each iteration its weights are tuned to reduce loss
- Its inputs are tuned as well. Some of them become larger, some – smaller
- Now the neuron needs to be re-tuned for it's new inputs



Batch normalization

TL; DR:

- *It's usually a good idea to normalize linear model inputs*

(c) Every machine learning lecturer, ever

Batch normalization

- Normalize activation of a hidden la $h_i = \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$

  (zero mean unit variance)

- Update μ_i, σ_i^2 with moving average while training

$$\mu_i := \alpha \cdot mean_{batch} + (1-\alpha) \cdot \mu_i$$
$$\sigma_i^2 := \alpha \cdot variance_{batch} + (1-\alpha) \cdot \sigma_i^2$$

Batch normalization

Original algorithm (2015)

What is this?

This transformation should be able to represent the identity transform.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

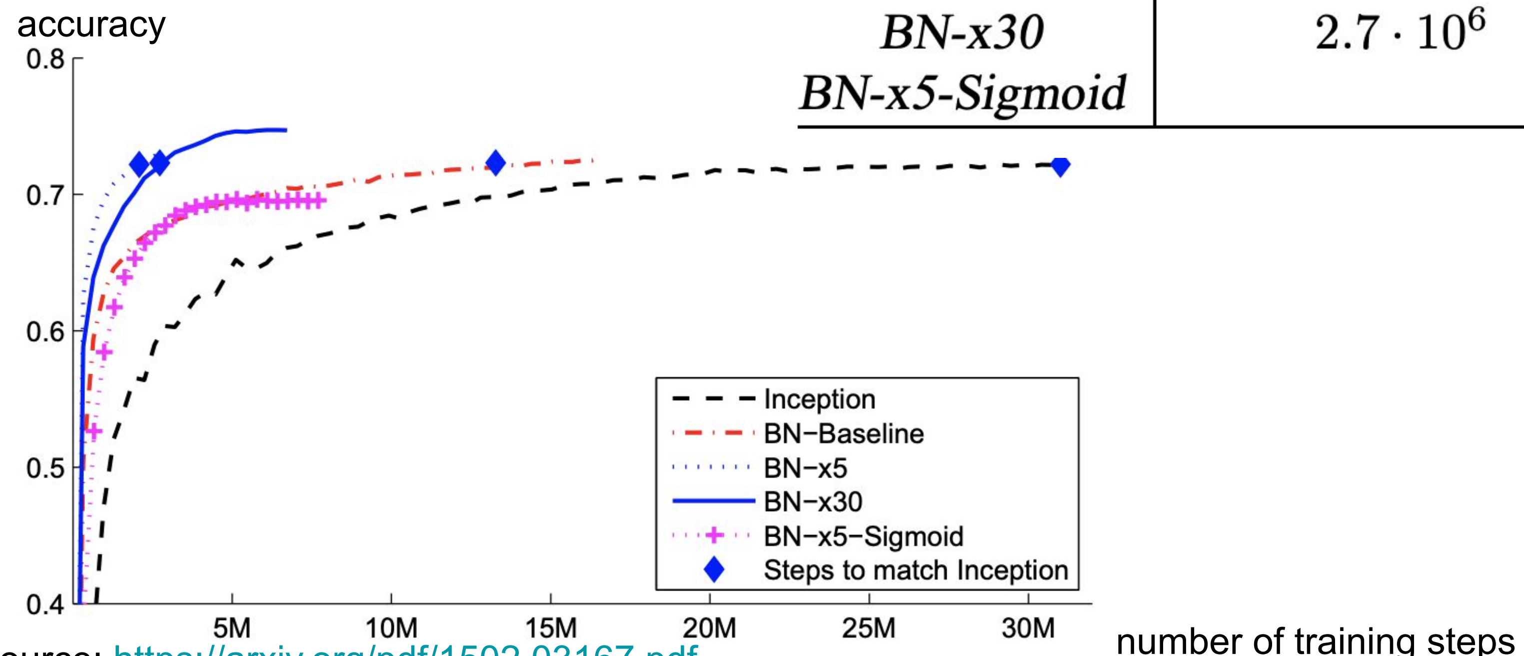
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$\boxed{y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)} \qquad \text{// scale and shift}$$

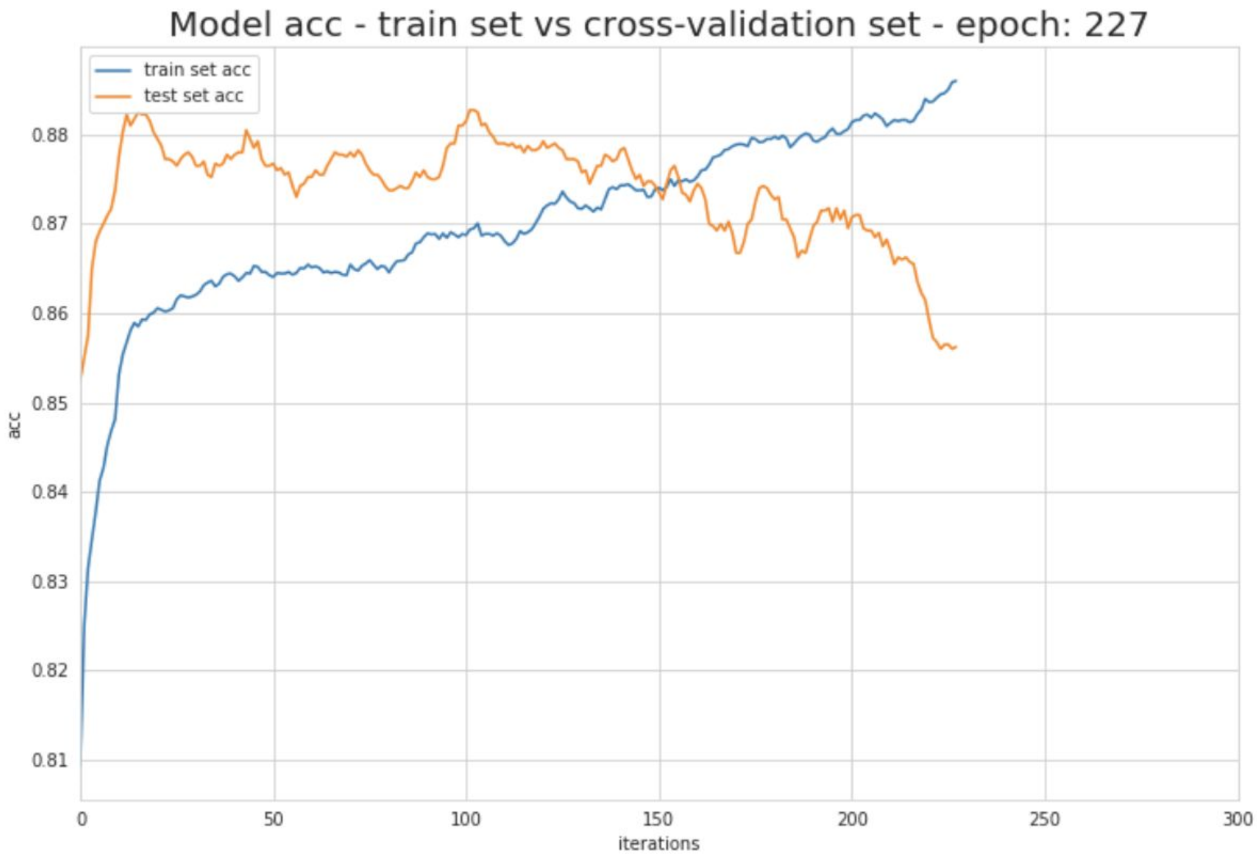
Batch normalization

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%



source: https://arxiv.org/pdf/1502.03167.pdf

Problem: overfitting



Regularization

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

Adding some extra term to the loss function.

Common cases:

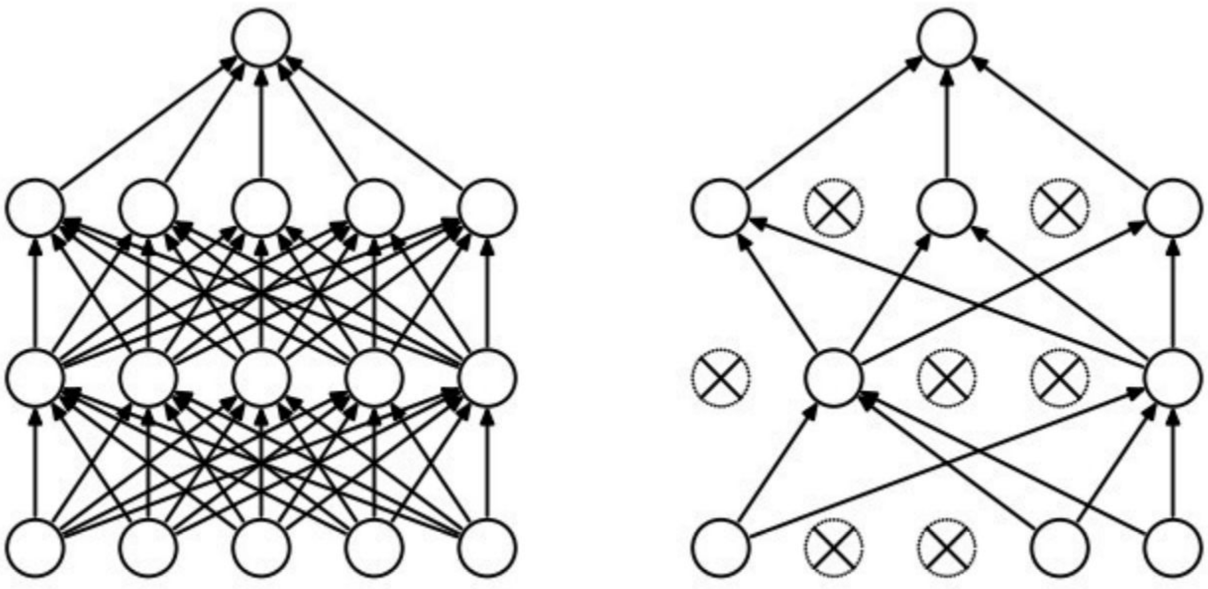
- L2 regularization: $R(W) = \|W\|_2^2$
- L1 regularization: $R(W) = \|W\|_1$
- Elastic Net (L1 + L2): $R(W) = \beta\|W\|_2^2 + \|W\|_1$

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

Regularization: Dropout

Some neurons are “dropped” during training.

Prevents overfitting.

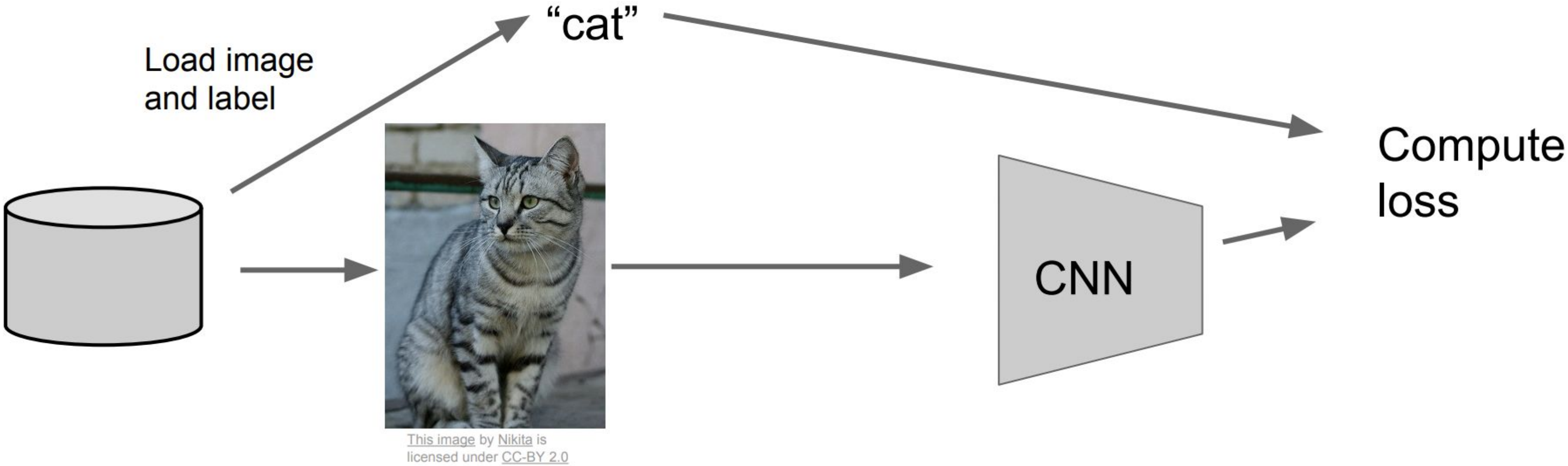


(a) Standard Neural Net (b) After applying dropout.

Actually, on test case output should be normalized. See sources for more info.

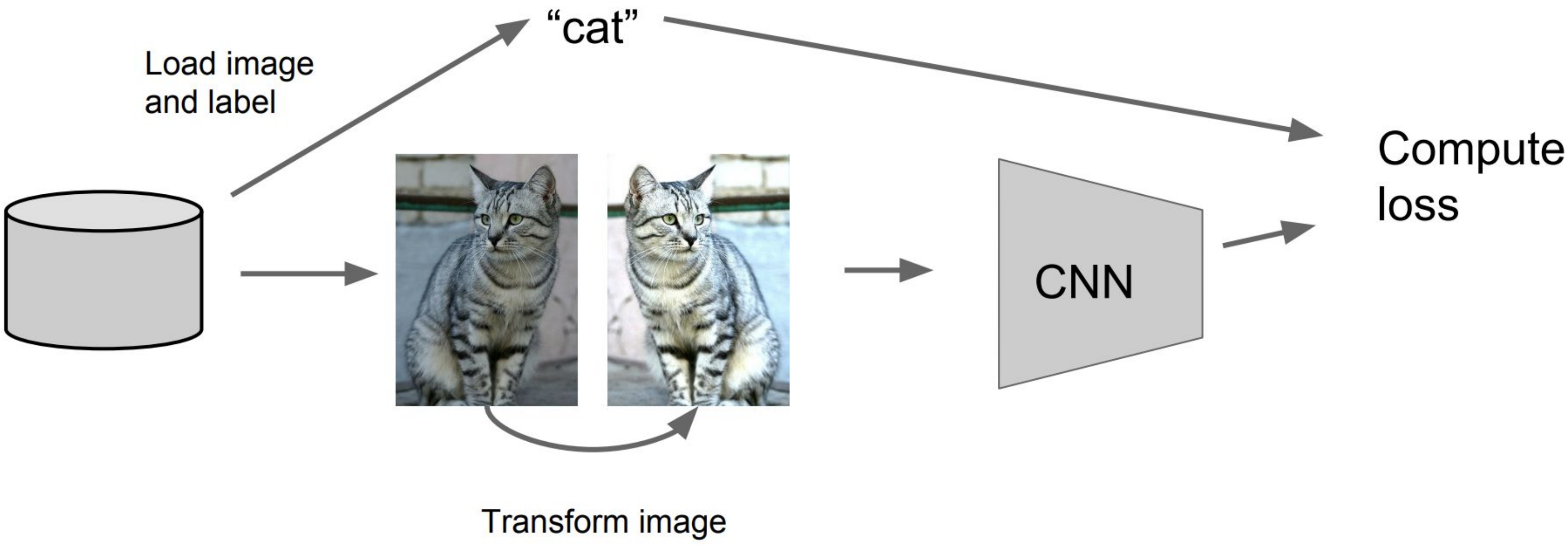
source: https://jmlr.org/papers/v15/srivastava14a.html

Regularization: data augmentation



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

Regularization: data augmentation



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

Optimization:

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

Regularization:

- Add some weight constraints
- Add some random noise during train and marginalize it during test
- Add some prior information in appropriate form

Further readings available here