Optimization and Regularization for NNs

Vladislav Goncharenko
ML Teamlead, DZEN

girafe ai

MSU, spring 2024

Outline

1. Previous lecture recap
   a. activations
   b. backpropagation
2. Optimizers
   a. SGD
   b. Momentum
   c. RMSProp
   d. Adam
3. Data normalization
   a. Batch Norm
   b. Layer Norm
4. Regularization
   a. Dropout
5. Augmentation
   a. Images
   b. Texts

Recap

girafe
ai

01

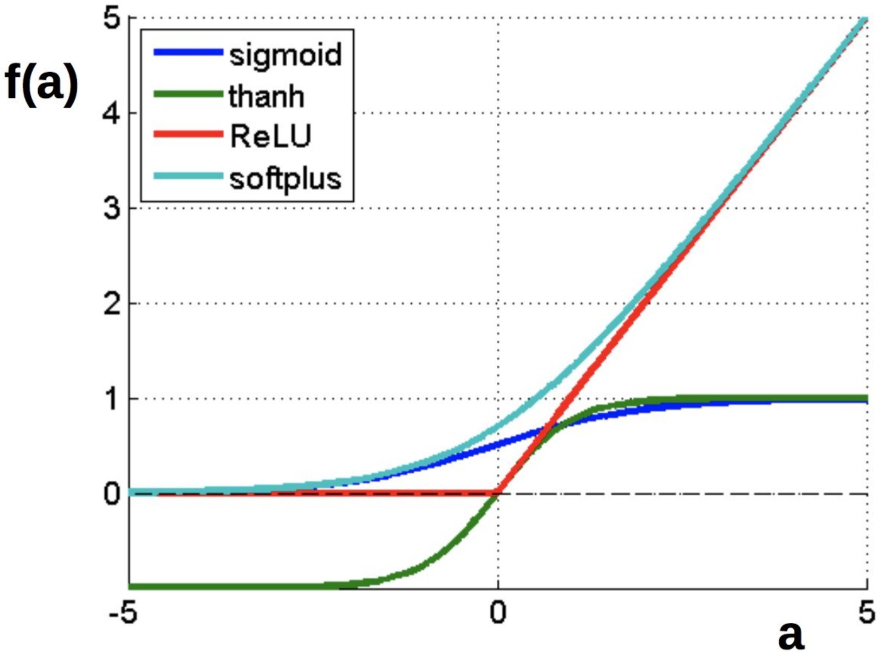
Once again: nonlinearities

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

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

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

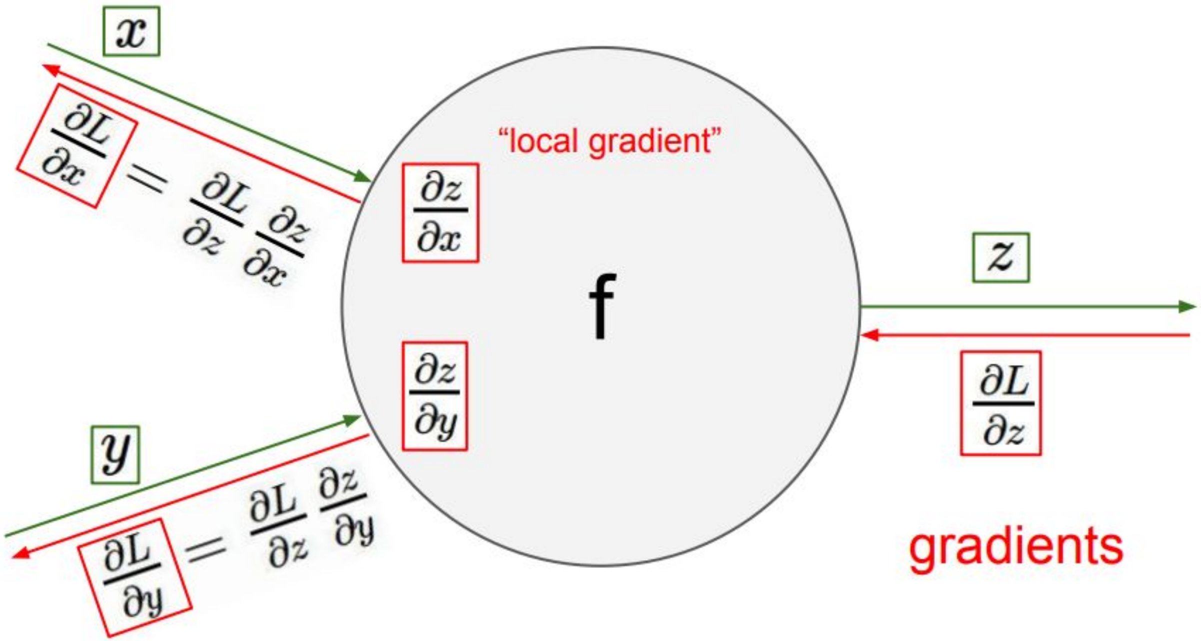
$$f(a) = \log(1+e^{a})$$



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.

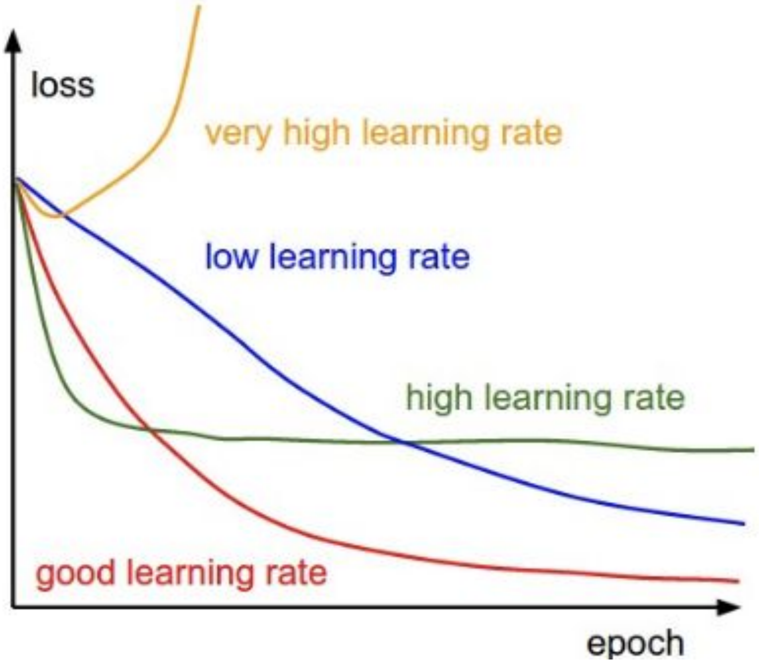
Optimizers

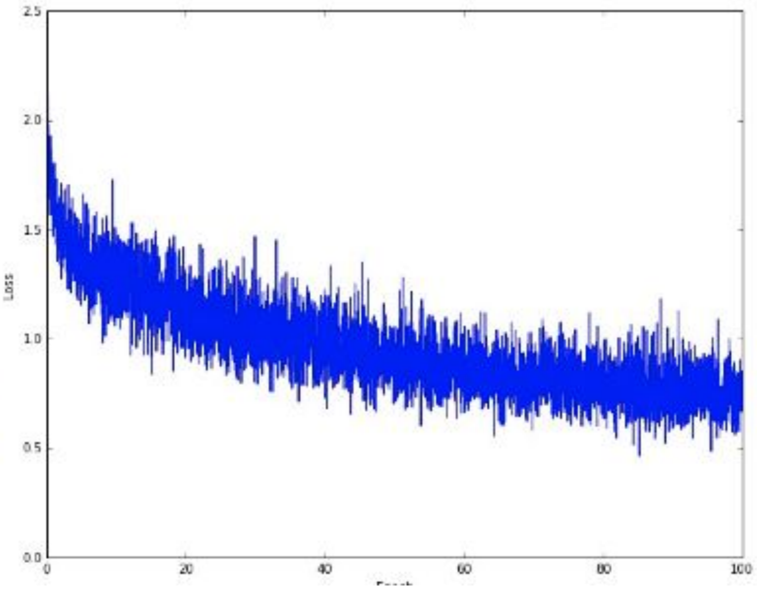
girafe
ai

02

Stochastic gradient descent

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

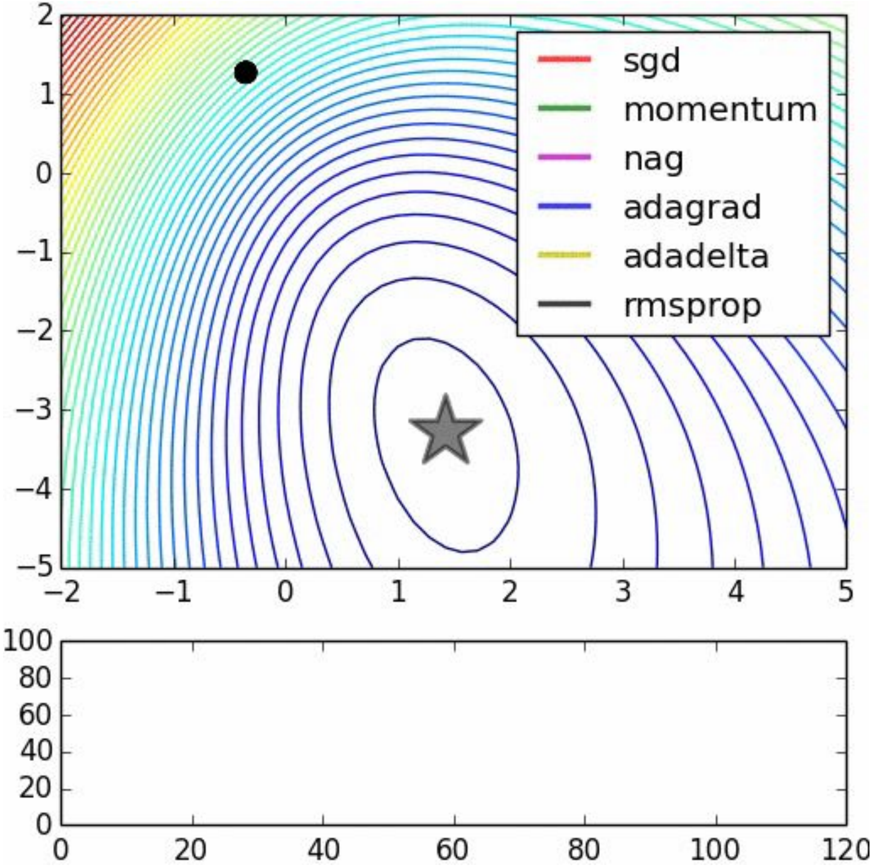




Optimizers

There are lots of optimizers:

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

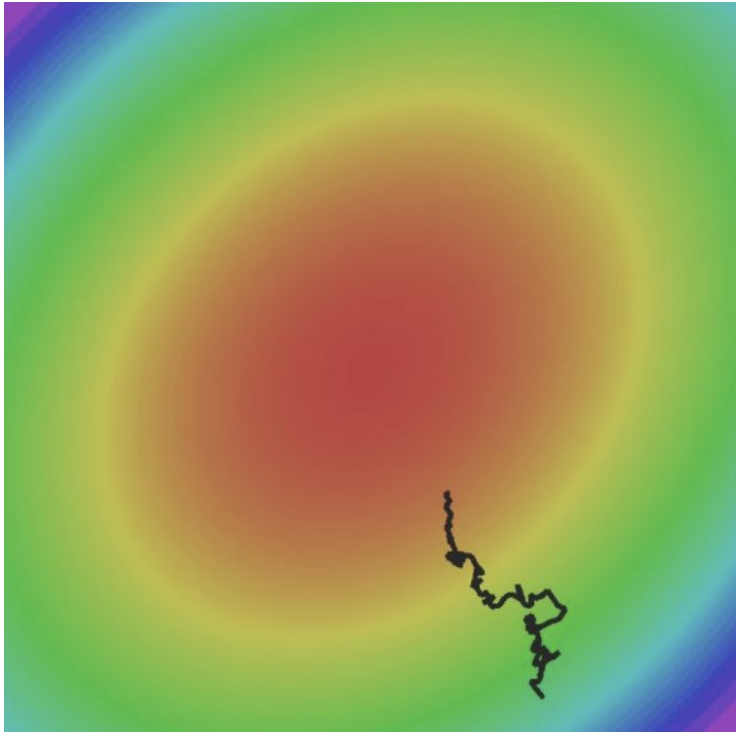


Optimization: SGD

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

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

Averaging over mini batches
=> noisy gradient



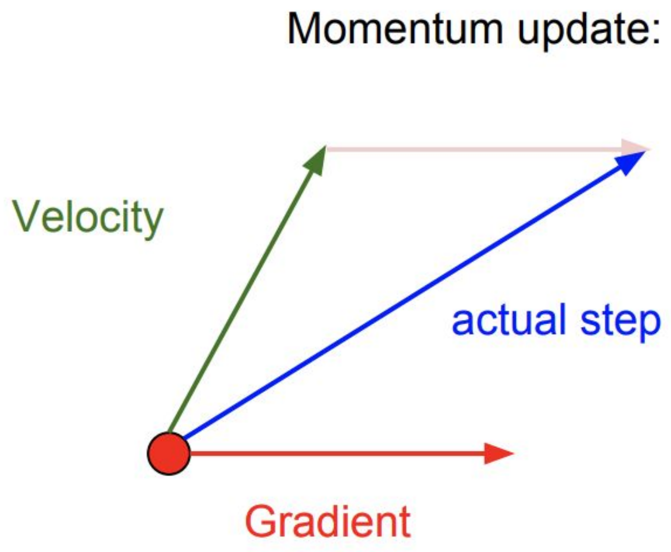
First idea: momentum

Simple SGD

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

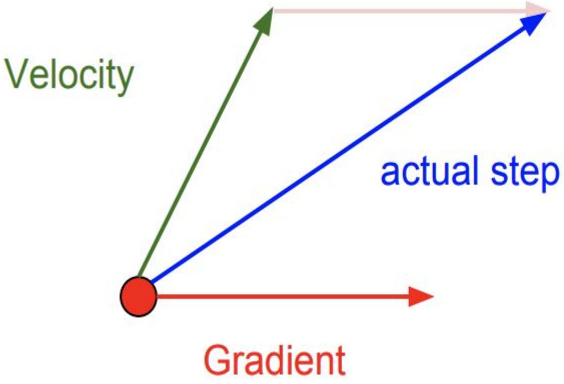
SGD with momentum

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



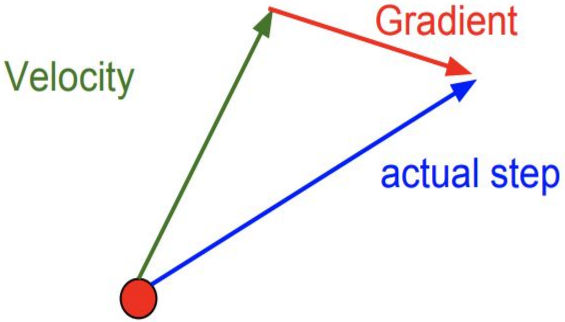
Nesterov momentum

Momentum update:



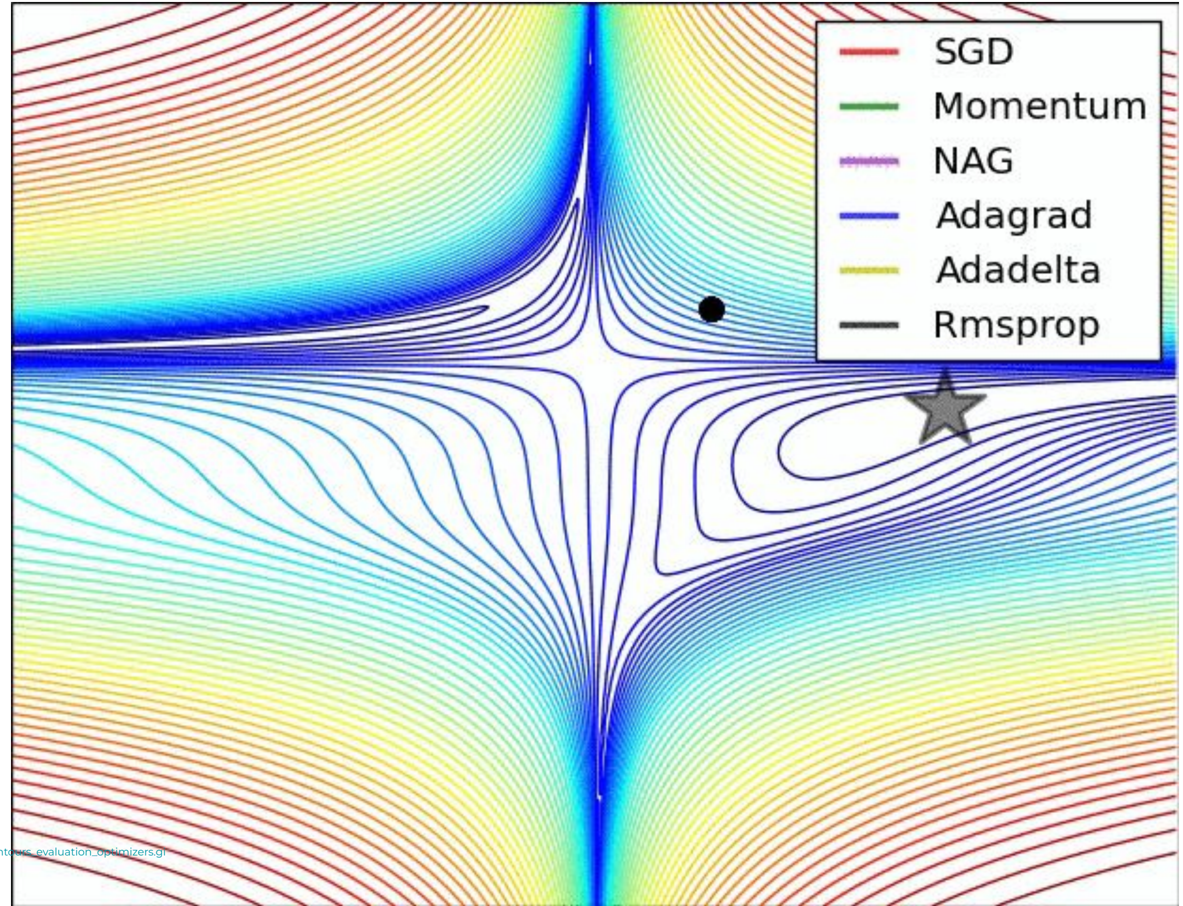
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Nesterov Momentum

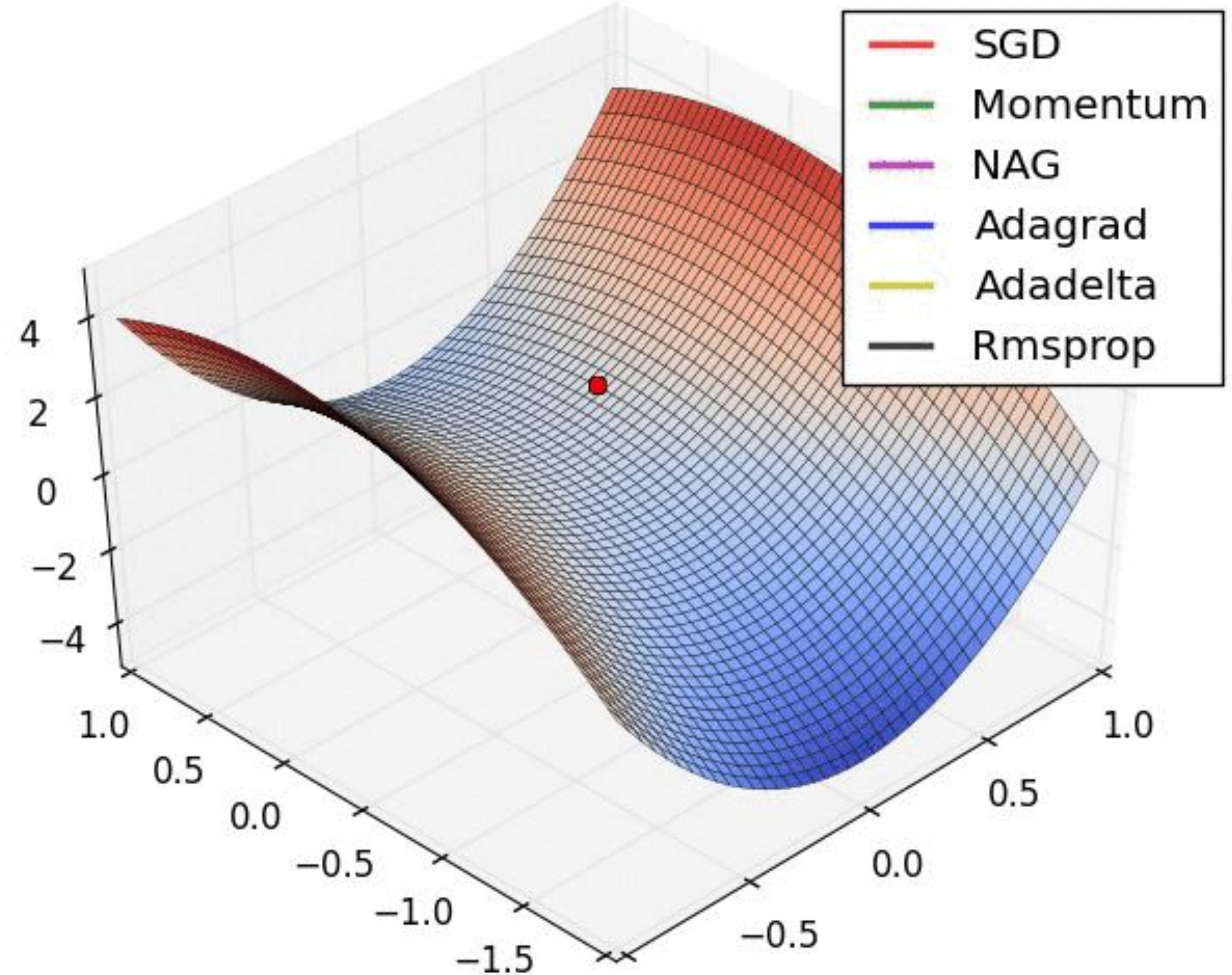


$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$

Comparing momentums



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


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

Second idea: different dimensions are different

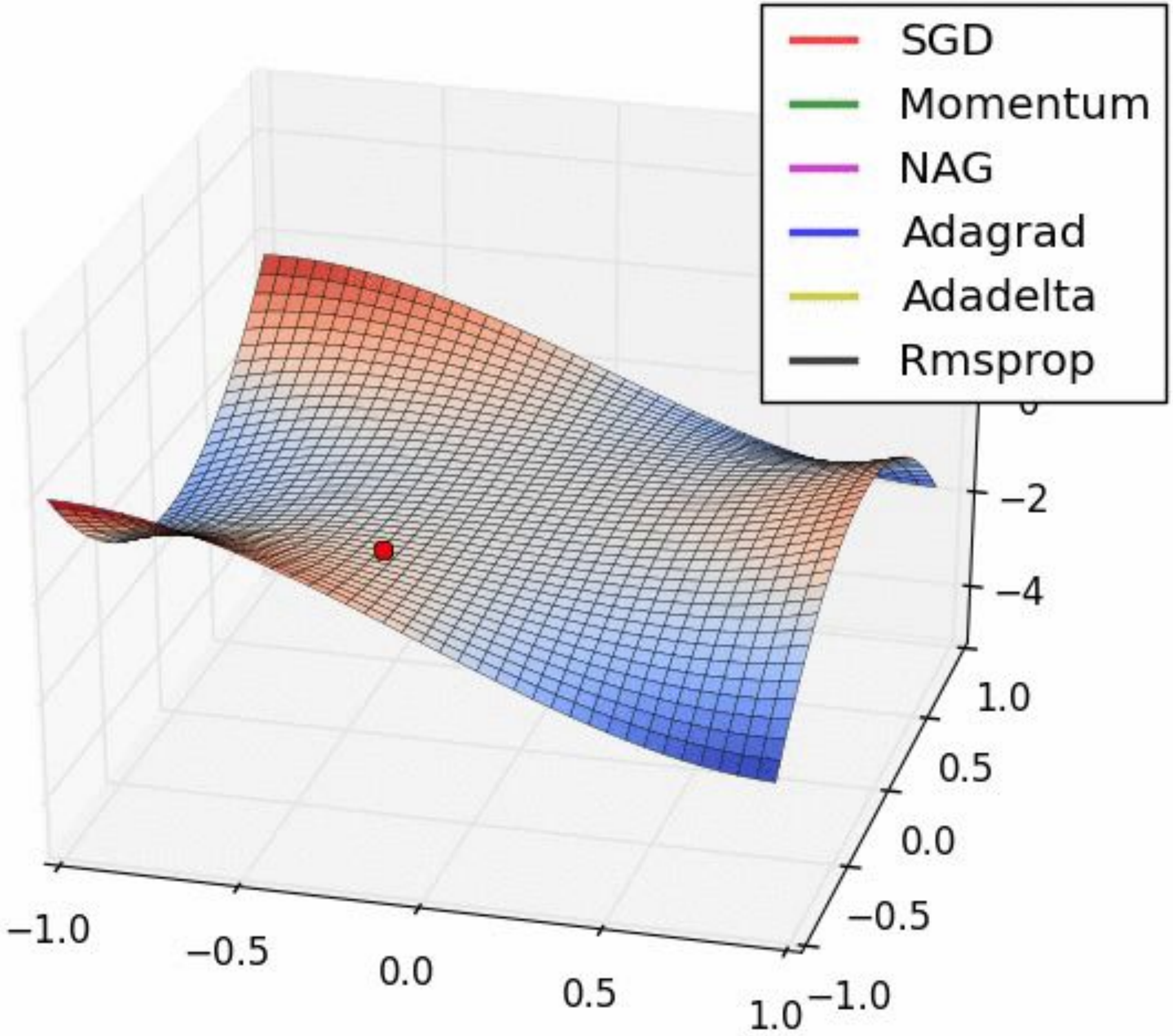
RMSProp - SGD with exponential cache

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

$$x_{t+1} = x_t - \alpha\frac{\nabla f(x_t)}{\text{cache}_{t+1}^{1/2} + \varepsilon}$$

Slide 29 Lecture 6 of Geoff Hinton's Coursera class
http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Simpler (historical) method:
Adagrad - SGD with cache


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

Adam

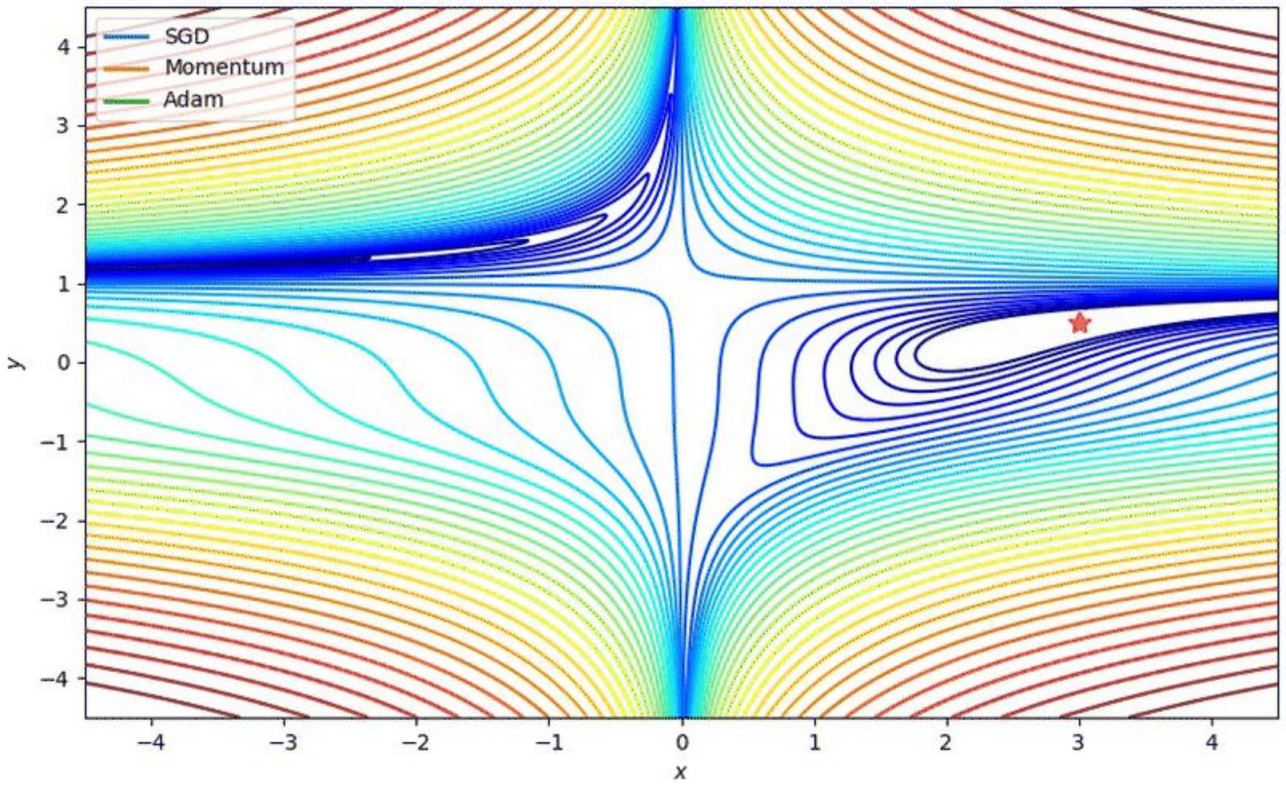
Let's combine the momentum idea and RMSProp normalization:

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

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




Andrej Karpathy
@karpathy

3e-4 is the best learning rate for Adam, hands down.

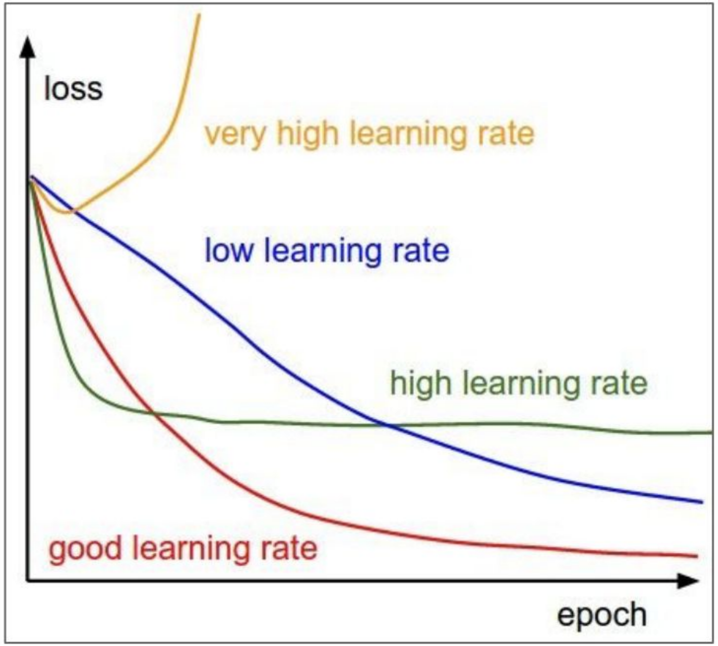
6:01 AM · Nov 24, 2016 · Twitter Web Client

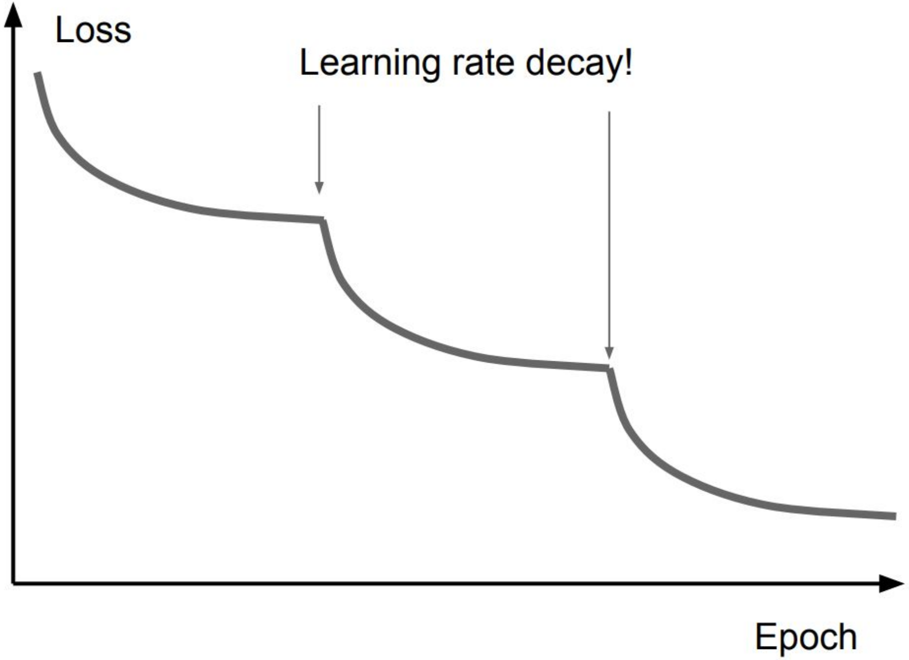
108 Retweets 461 Likes

Andrej Karpathy @karpathy · Nov 24, 2016
Replying to @karpathy
(i just wanted to make sure that people understand that this is a joke...)
9 3 119

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

Once more: learning rate





Weights initialization

- All zero initialization
  - pitfall
- Small random numbers
- Calibrated random numbers

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

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

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

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

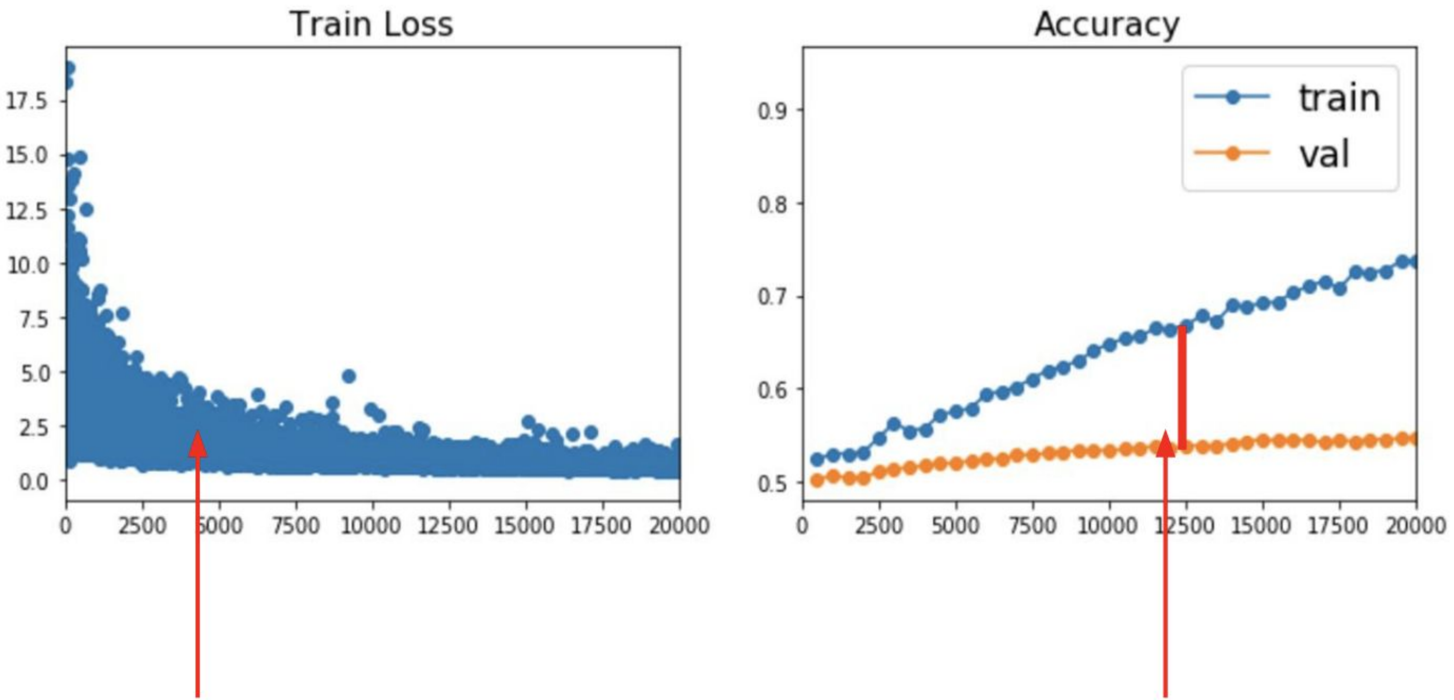
$$= (n\mathrm{Var}(w))\ \mathrm{Var}(x)$$

Sum up: optimization

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

Normalization

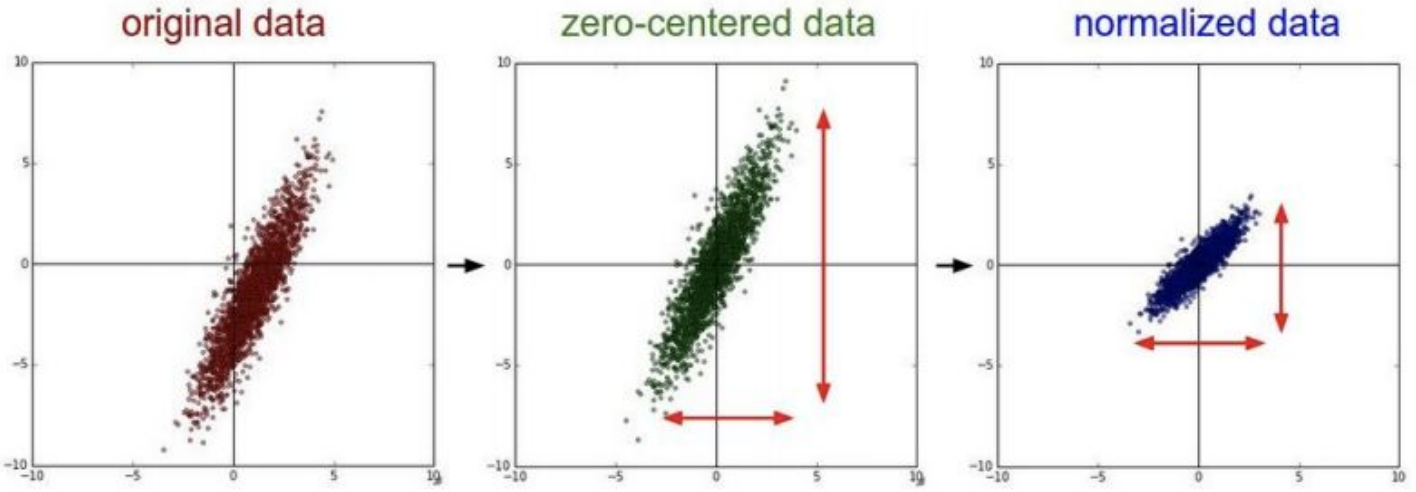
03



Better optimization algorithms
help reduce training loss

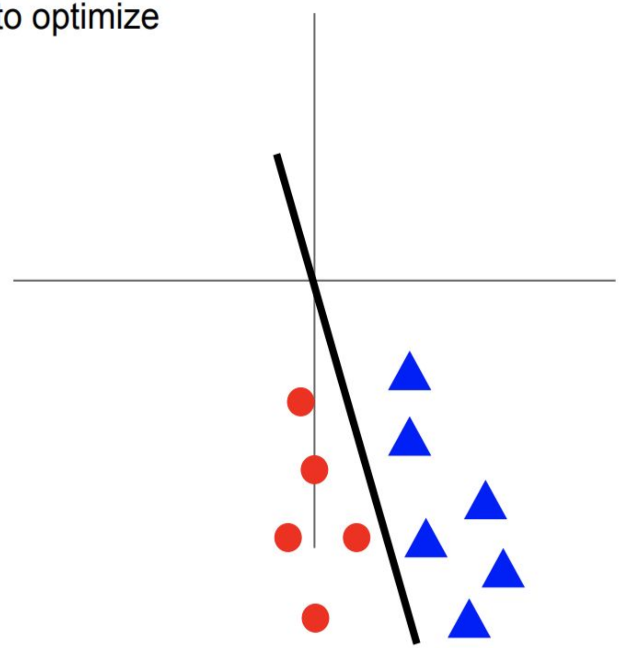
But we really care about error on new
data - how to reduce the gap?

Data normalization

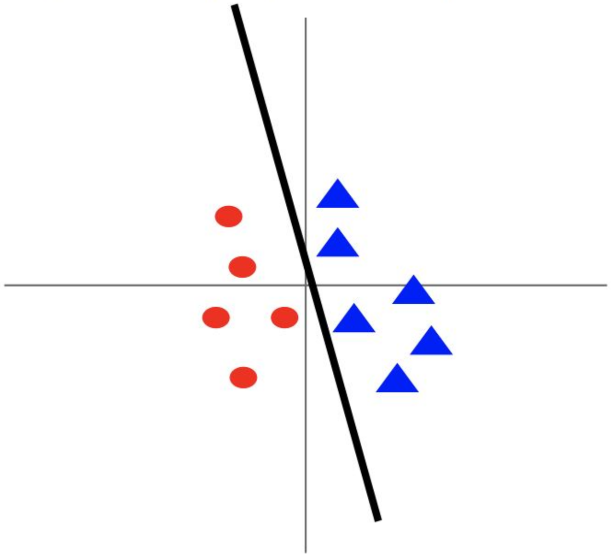


Data normalization

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



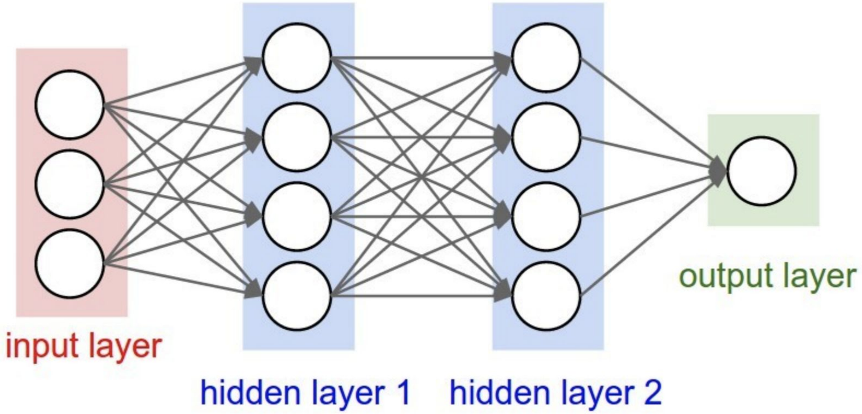
After normalization: less sensitive to small changes in weights; easier to optimize



Batch normalization

Problem (**internal covariate shift**):

- Consider a neuron in any layer beyond first
- At each iteration we tune it's weights towards better loss function
- But we also tune it's inputs. 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 layer

  (zero mean unit variance)

$$h_i = \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$$

- 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?

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

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

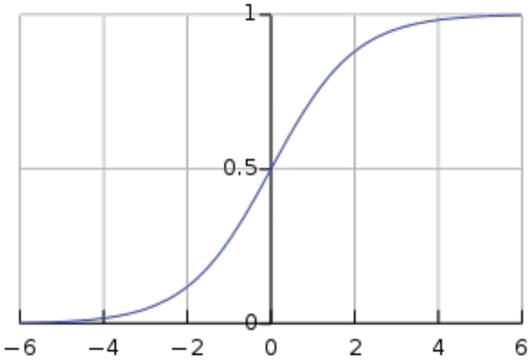
$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

Batch normalization



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

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

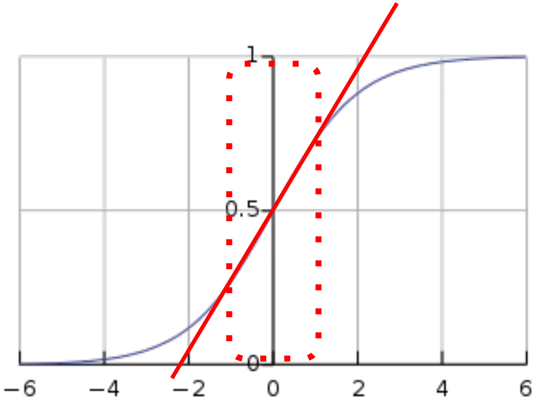
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$ // mini-batch mean

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

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

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

Batch normalization



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

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

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

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

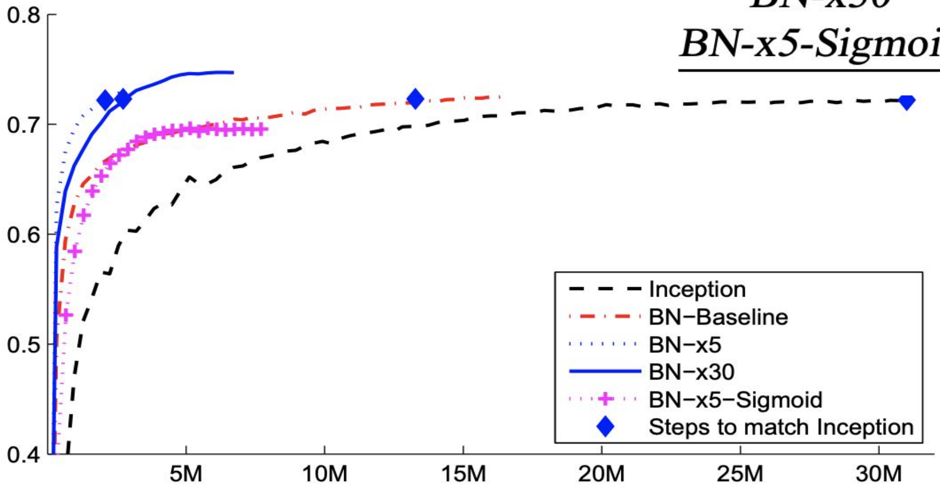
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

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

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%

accuracy



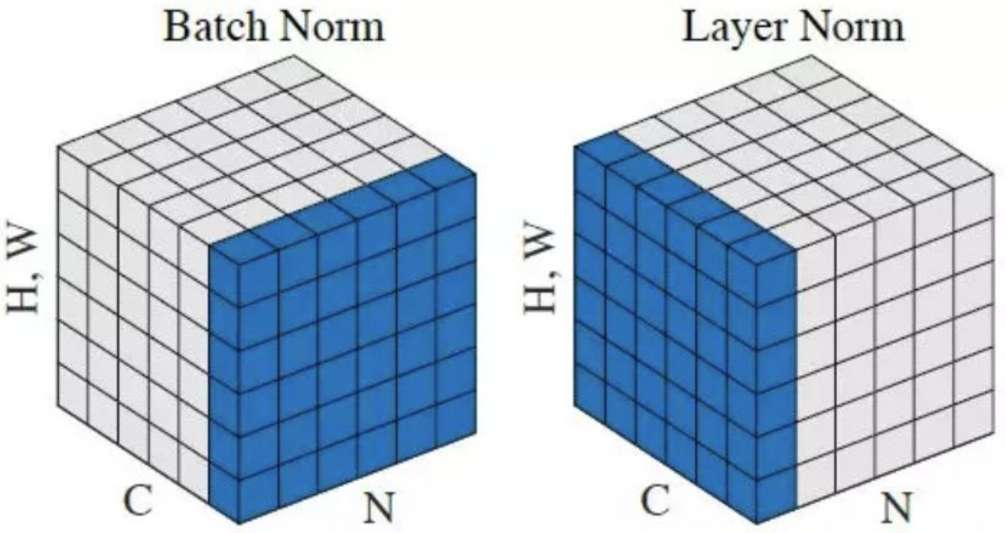
number of training steps

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

Layer normalization

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$



Regularization

girafe
ai

04

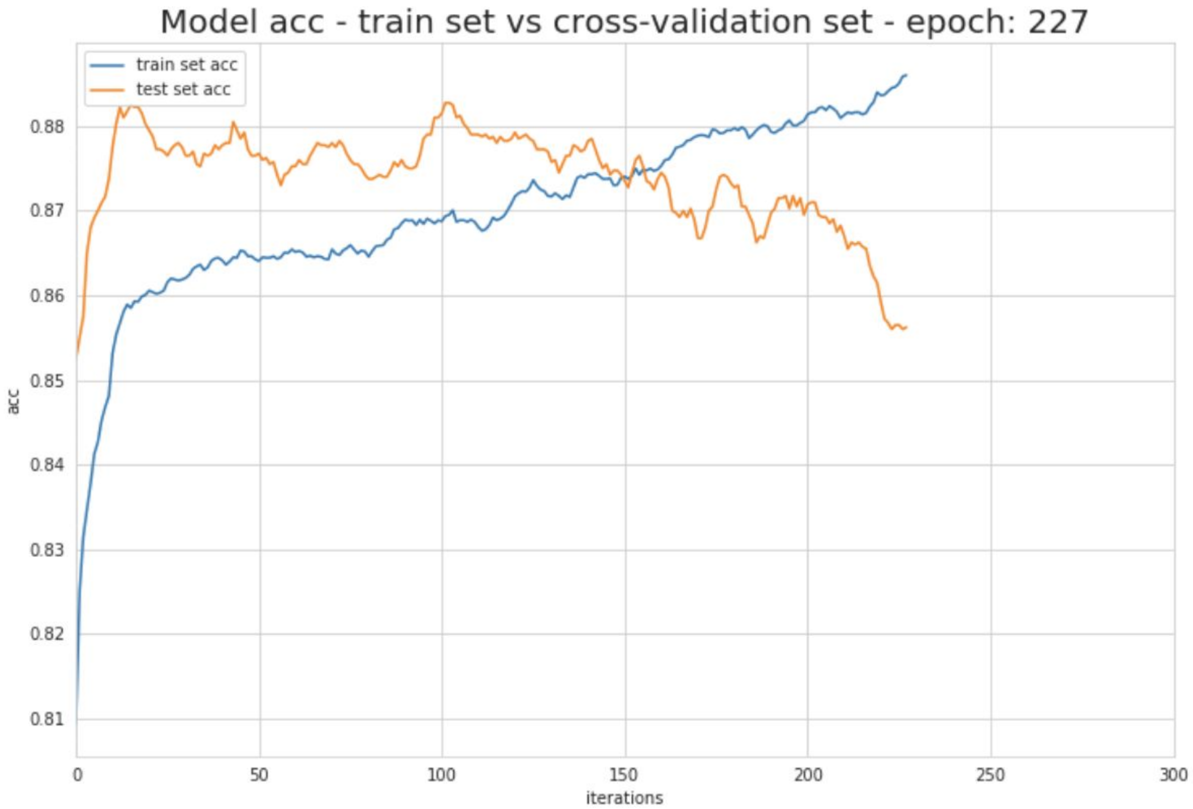
What is regularization anyway?

Regularization is a process that changes the result answer to be "simpler".
It is often used to obtain results for ill-posed problems or to prevent overfitting.

(c) Common knowledge site

- **Explicit** is regularization whenever one explicitly adds a term to the optimization problem. These terms could be priors, penalties, or constraints. Explicit regularization is commonly employed with ill-posed optimization problems. The regularization term, or penalty, imposes a cost on the optimization function to make the optimal solution unique.
- **Implicit** is all other forms of regularization. This includes, for example, early stopping, using a robust loss function, and discarding outliers

Problem: overfitting



Weights norm 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.

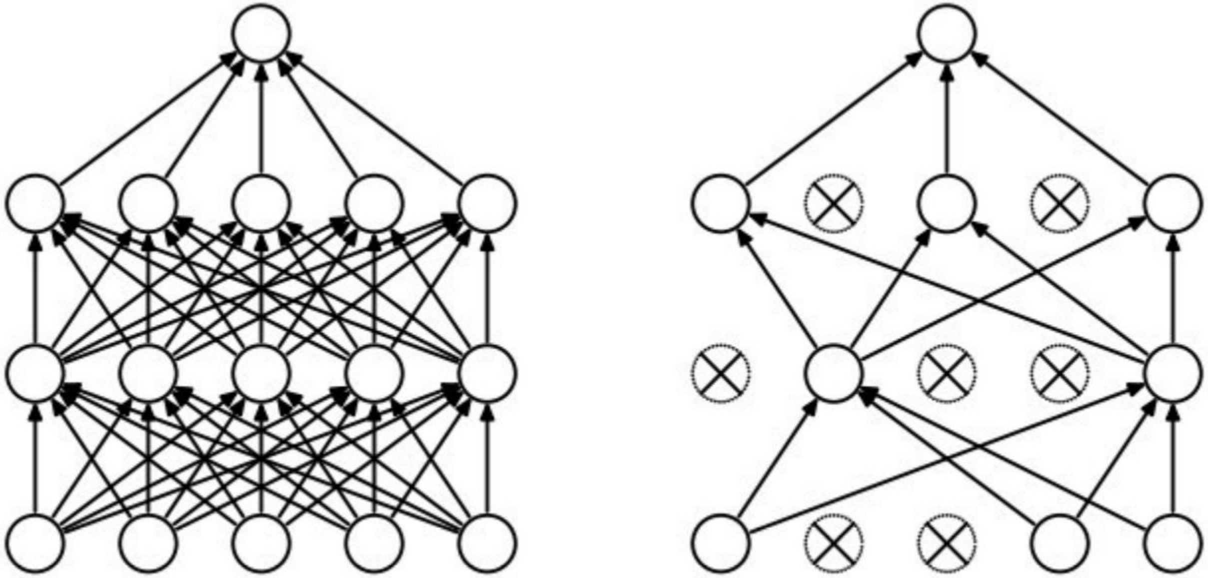
Common cases:

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

Dropout

Some neurons are "dropped" during training.

Prevents overfitting.



(a) Standard Neural Net

(b) After applying dropout.

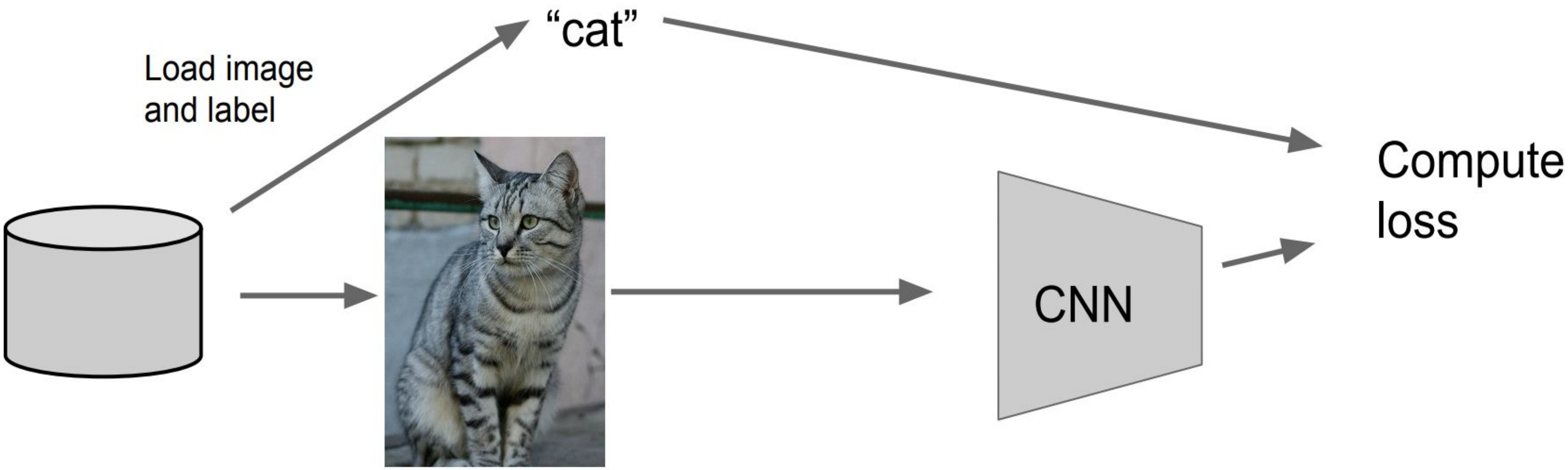
Actually, on test case output should be normalized. See sources for more info.

Augmentation

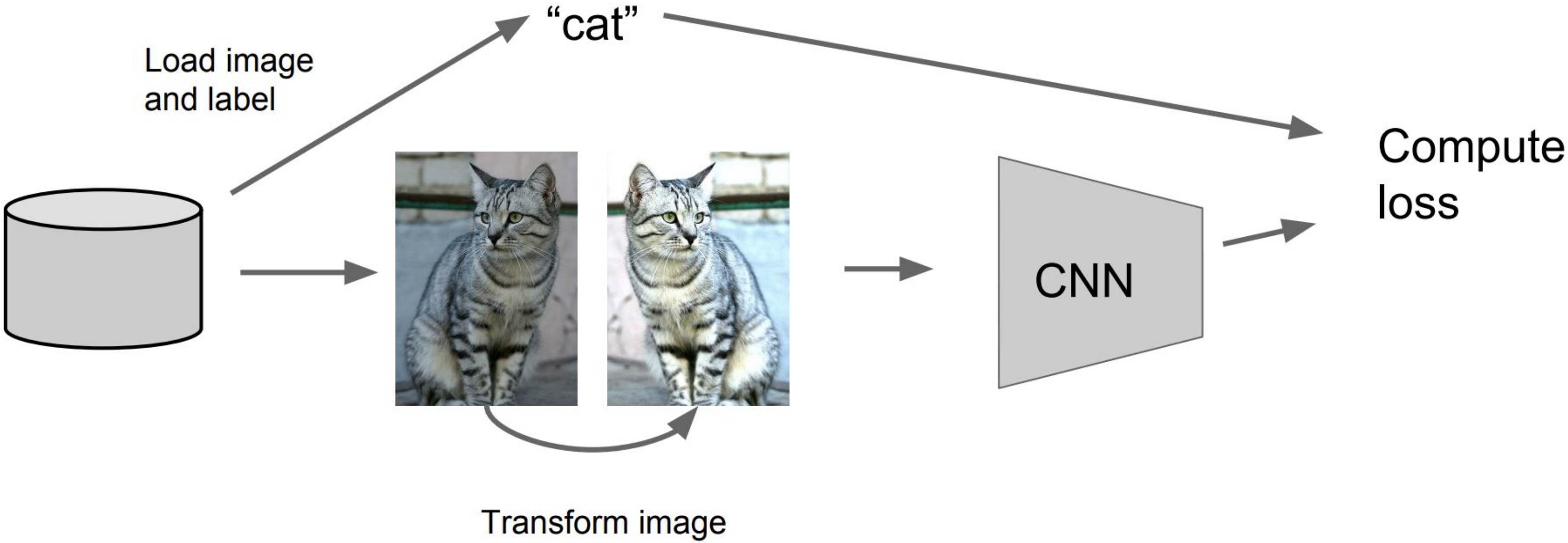
girafe
ai

05

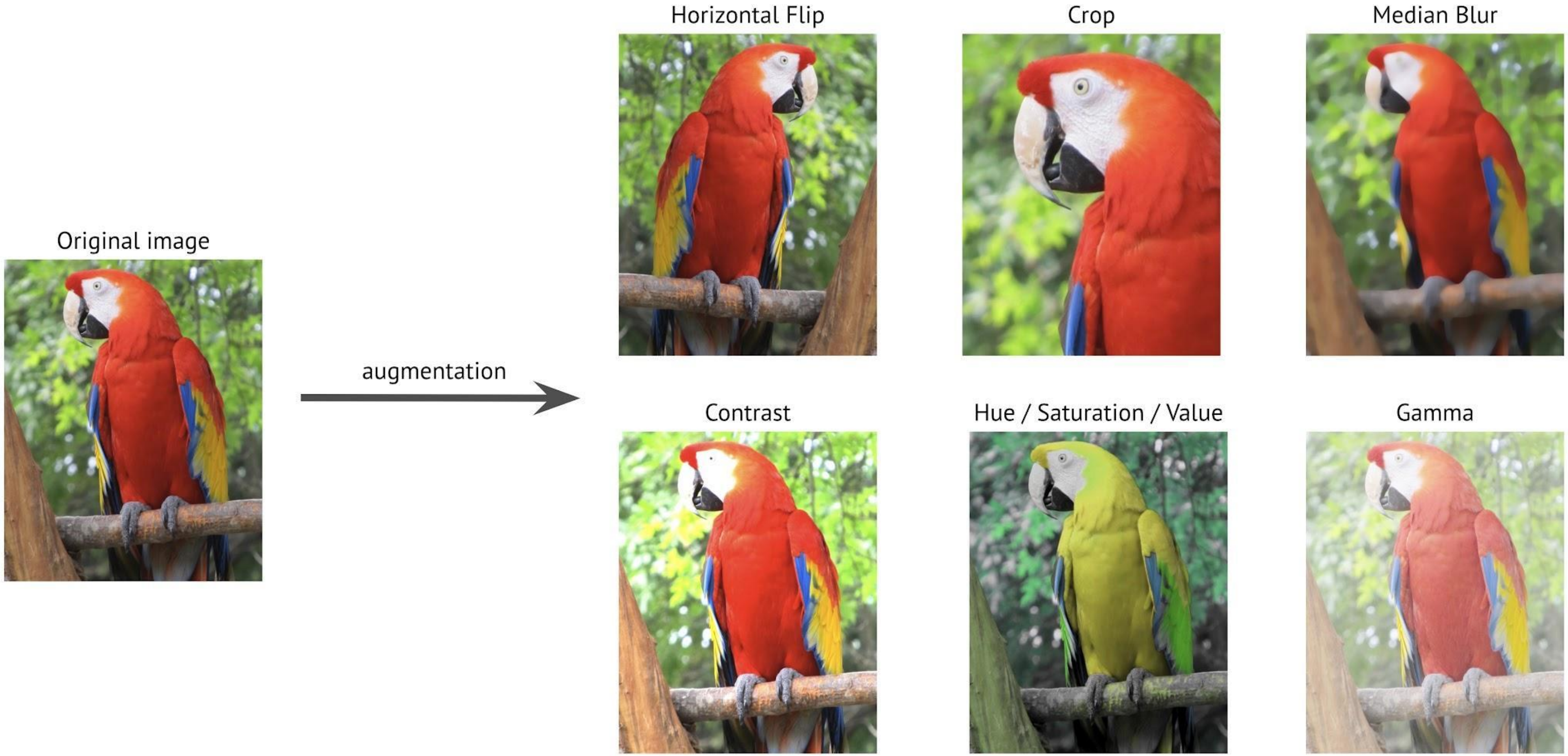
Data augmentation



Data augmentation



Many ways to augment



Albumentations for images

https://albumentations.ai/

Text augmentations

	Sentence
Original	The quick brown fox jumps over the lazy dog
Synonym (PPDB)	The quick brown fox climbs over the lazy dog
Word Embeddings (word2vec)	The easy brown fox jumps over the lazy dog
Contextual Word Embeddings (BERT)	Little quick brown fox jumps over the lazy dog
PPDB + word2vec + BERT	Little easy brown fox climbs over the lazy dog

- https://github.com/sloria/TextBlob
- https://github.com/facebookresearch/AugLy
- https://github.com/makcedward/nlpaug/
- https://github.com/QData/TextAttack

Revise

1. Previous lecture recap
    a. activations
    b. backpropagation
2. Optimizers
    a. SGD
    b. Momentum
    c. RMSProp
    d. Adam
3. Data normalization
    a. Batch Norm
    b. Layer Norm
4. Regularization
    a. Dropout
5. Augmentation
    a. Images
    b. Texts

Thanks for attention!

Questions?

girafe
ai