Optimization, Inductive bias

Radoslav Neychev

MIPT, fall 2024

Outline

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

Recap

girafe ai

01

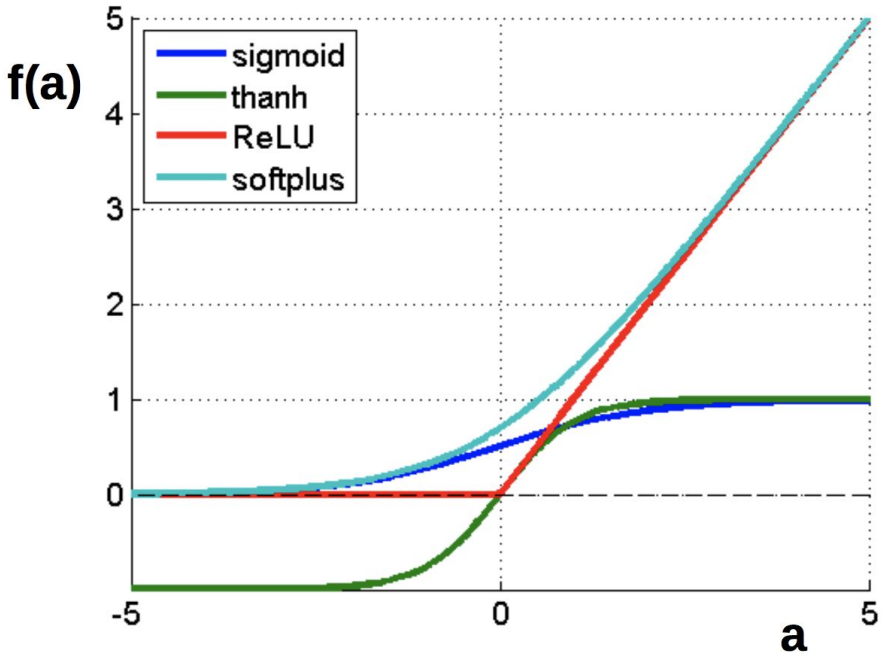
Once again: nonlinearities

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

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

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

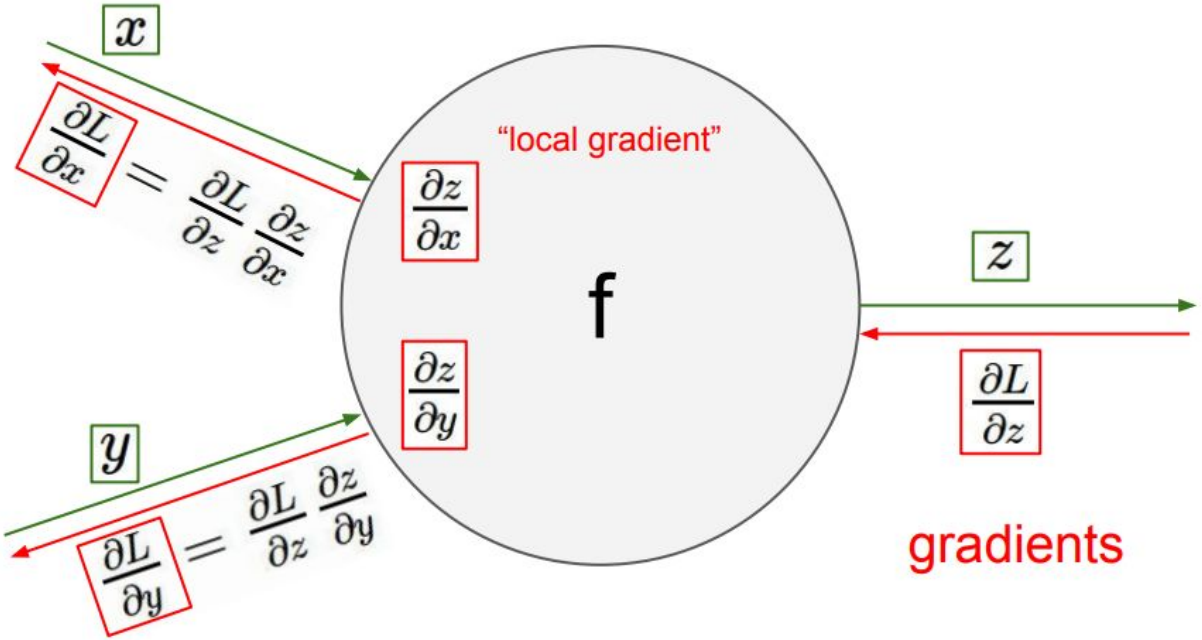
$$f(a) = \log(1 + e^{a})$$



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.

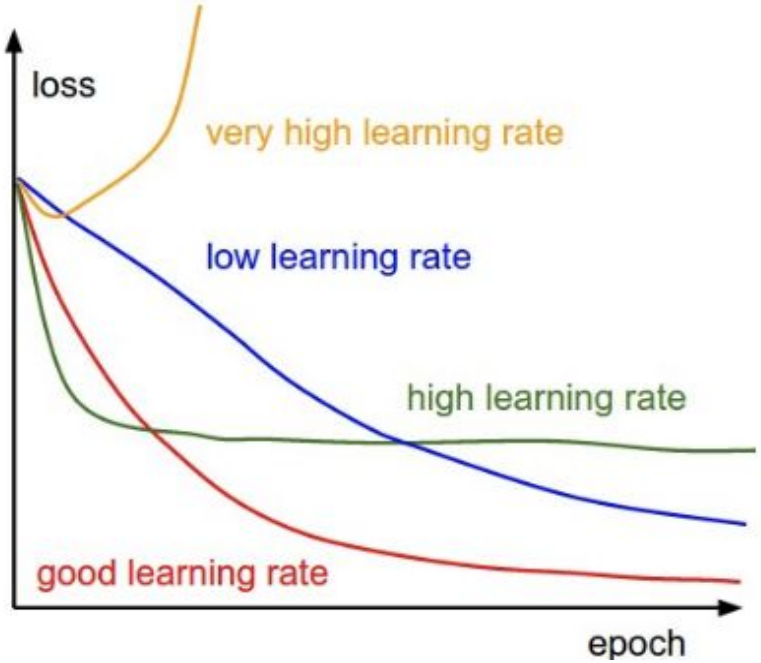
Optimizers

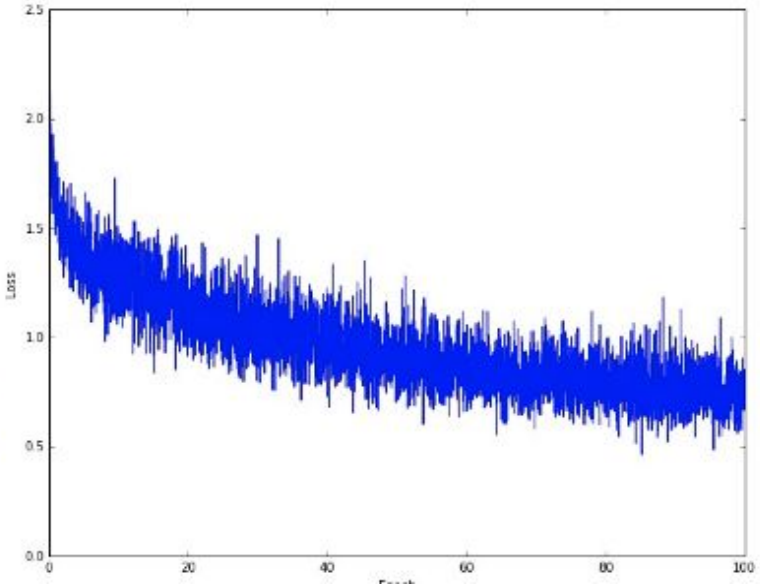
girafe ai

02

Stochastic gradient descent

$$\theta_{t+1} = \theta_t - \text{learning rate} \cdot \frac{\partial}{\partial \theta} Loss$$





SGD main problem

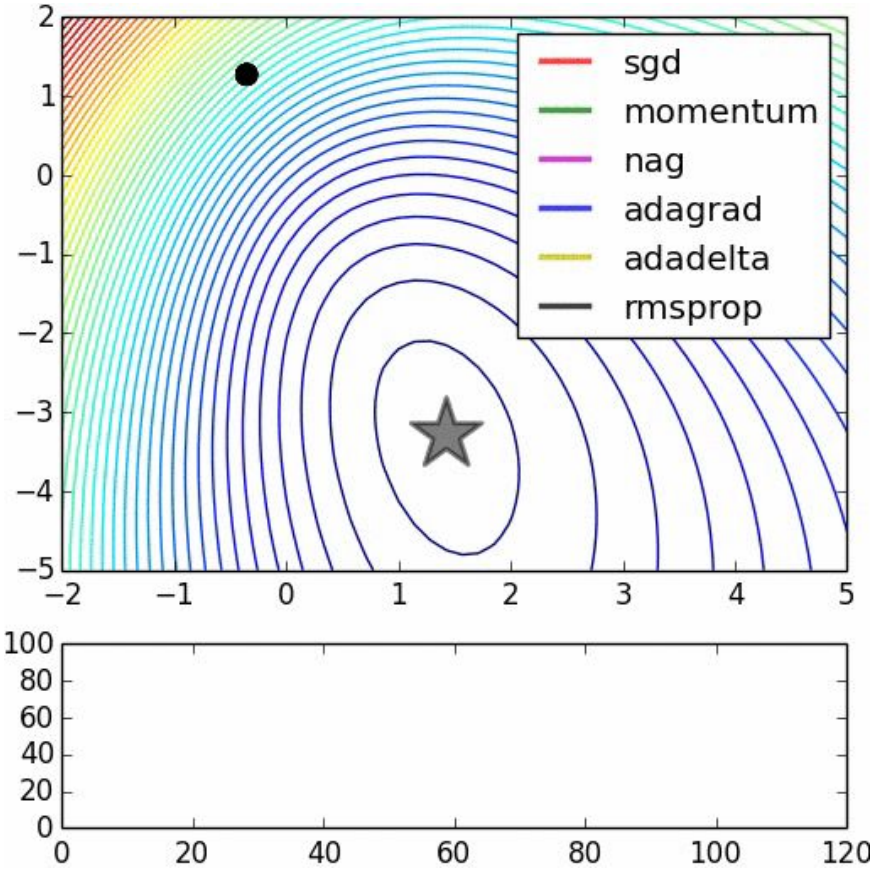
It finds global minimum
only for convex problem.

So we implicitly assume that we work
with a convex loss function surface.

Optimizers

There are lots of optimizers:

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

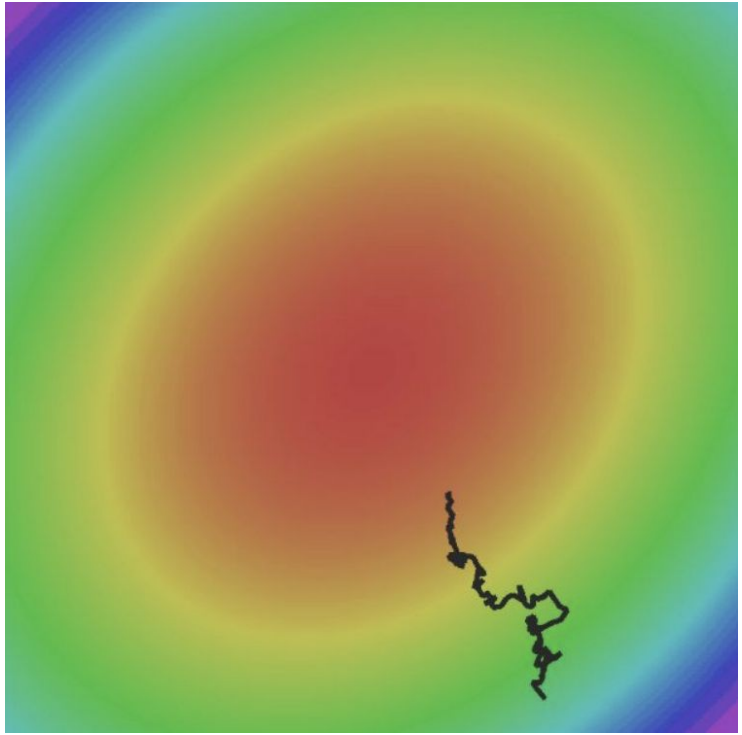


Optimization: SGD

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

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

Averaging over mini batches
=> noisy gradient



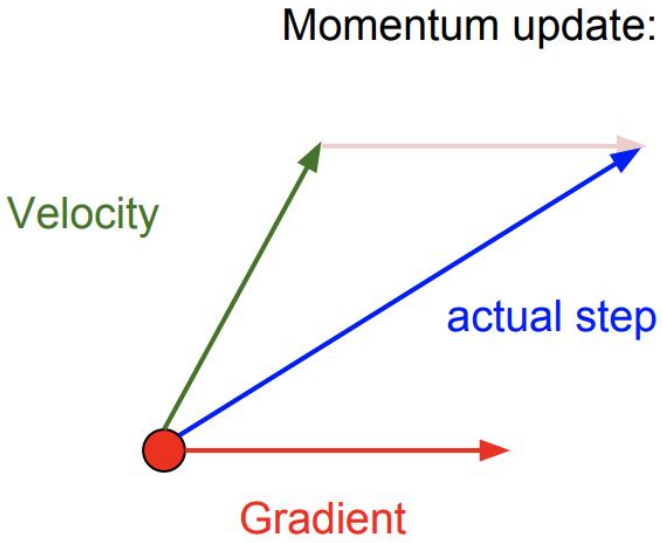
First idea: momentum

Simple SGD

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

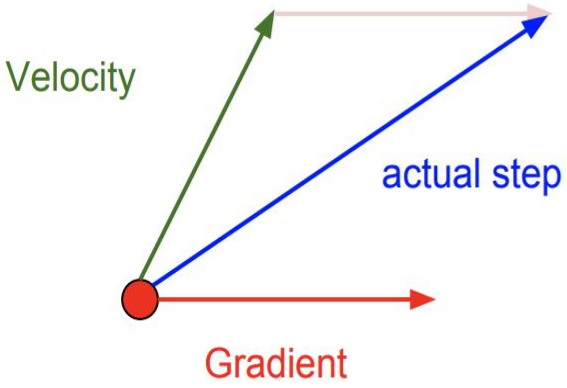
SGD with momentum

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



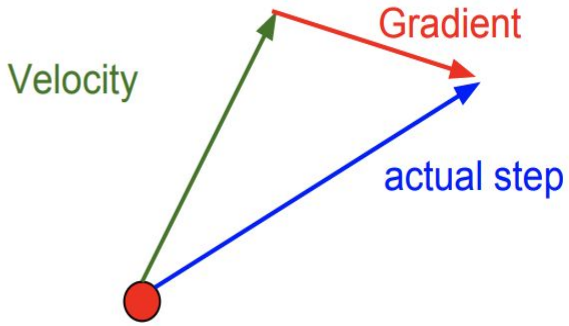
Nesterov momentum

Momentum update:



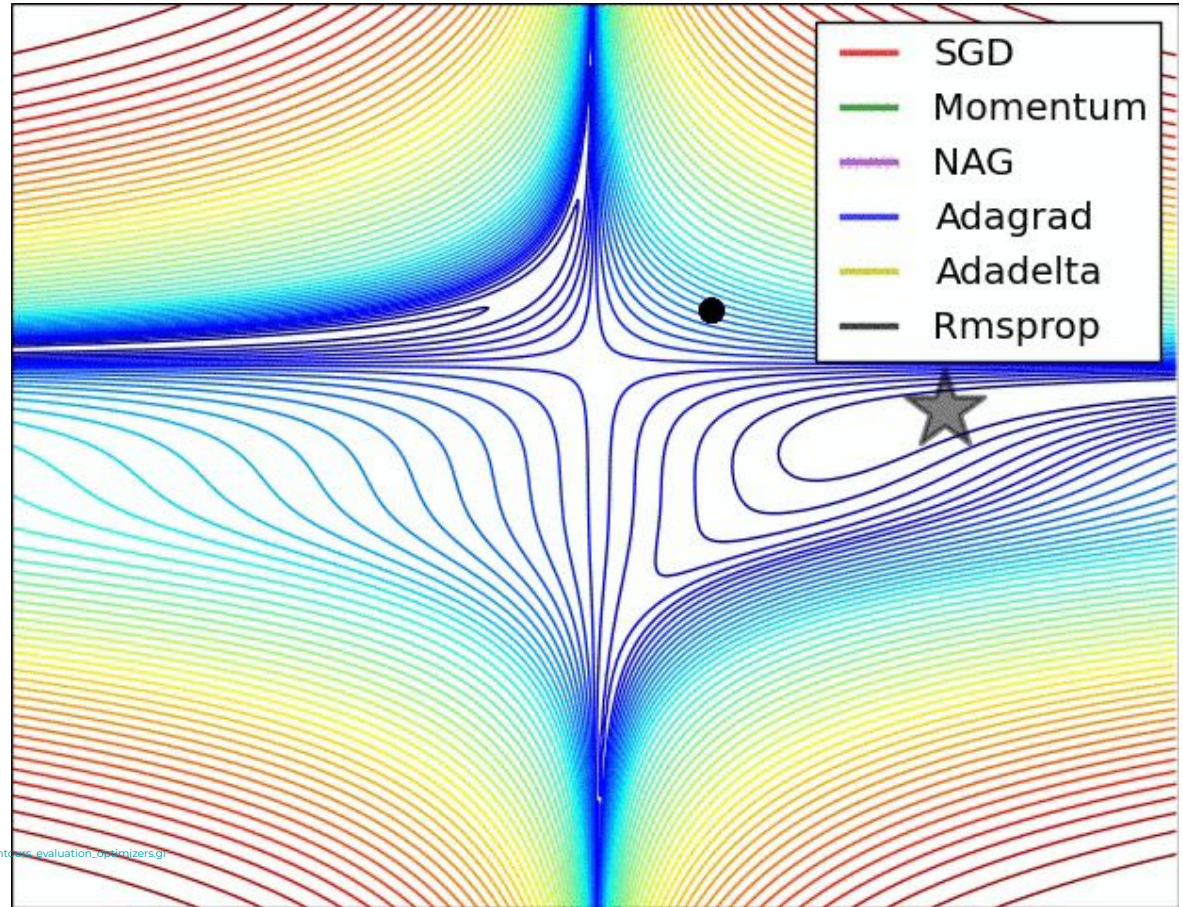
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Nesterov Momentum

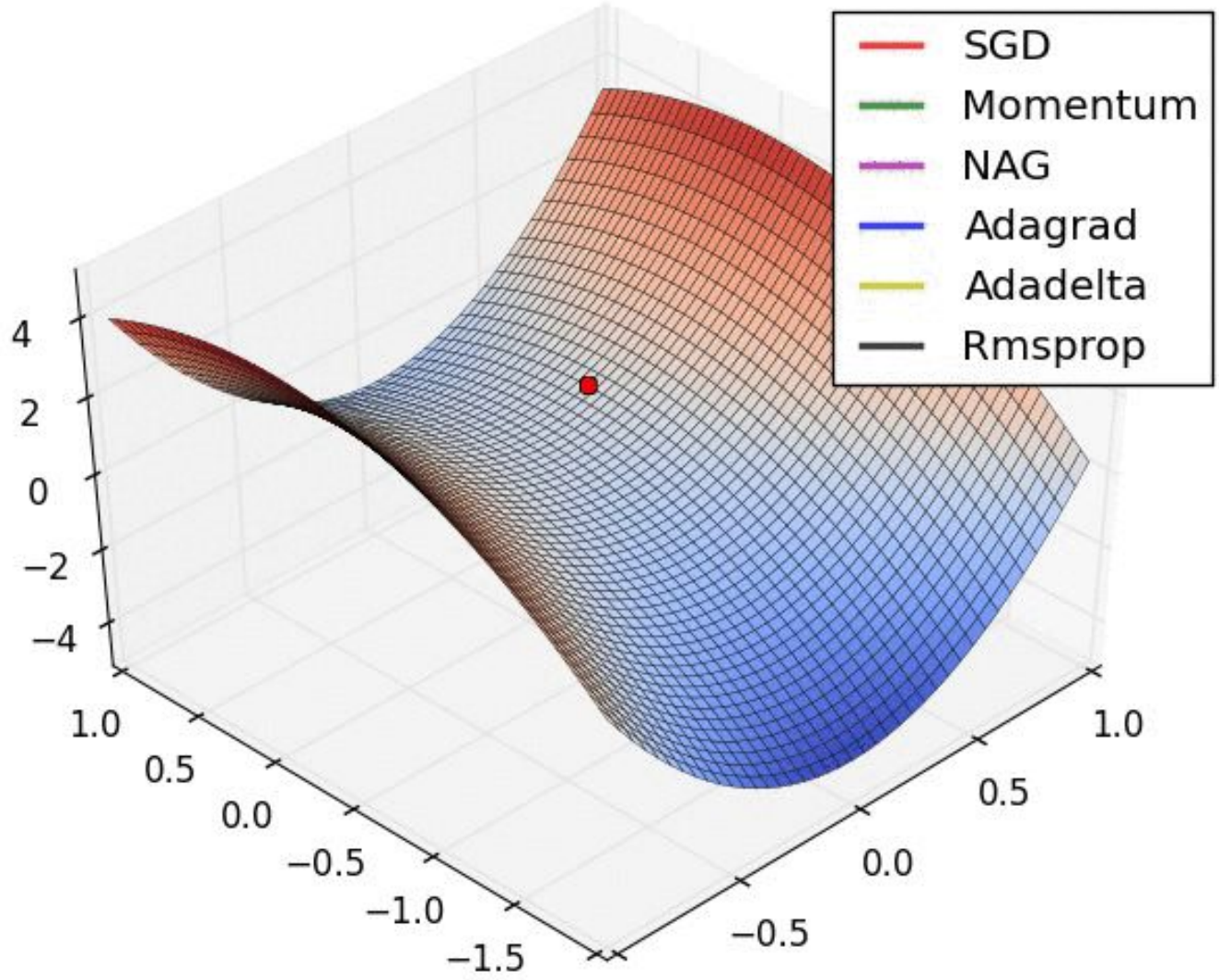


$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

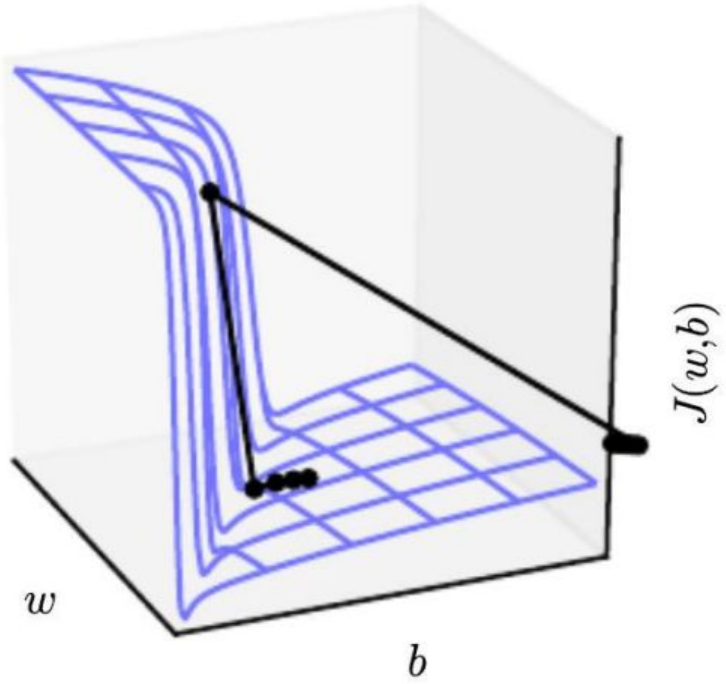
Comparing momentums

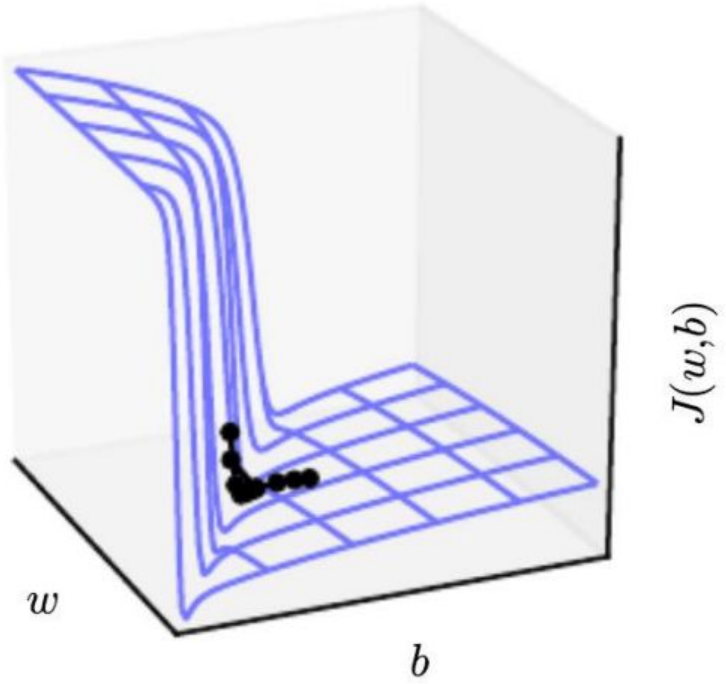


source: https://ruder.io/content/images/2016/09/contours_evaluation_optimizers.gif


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: step size





Different dimensions are different

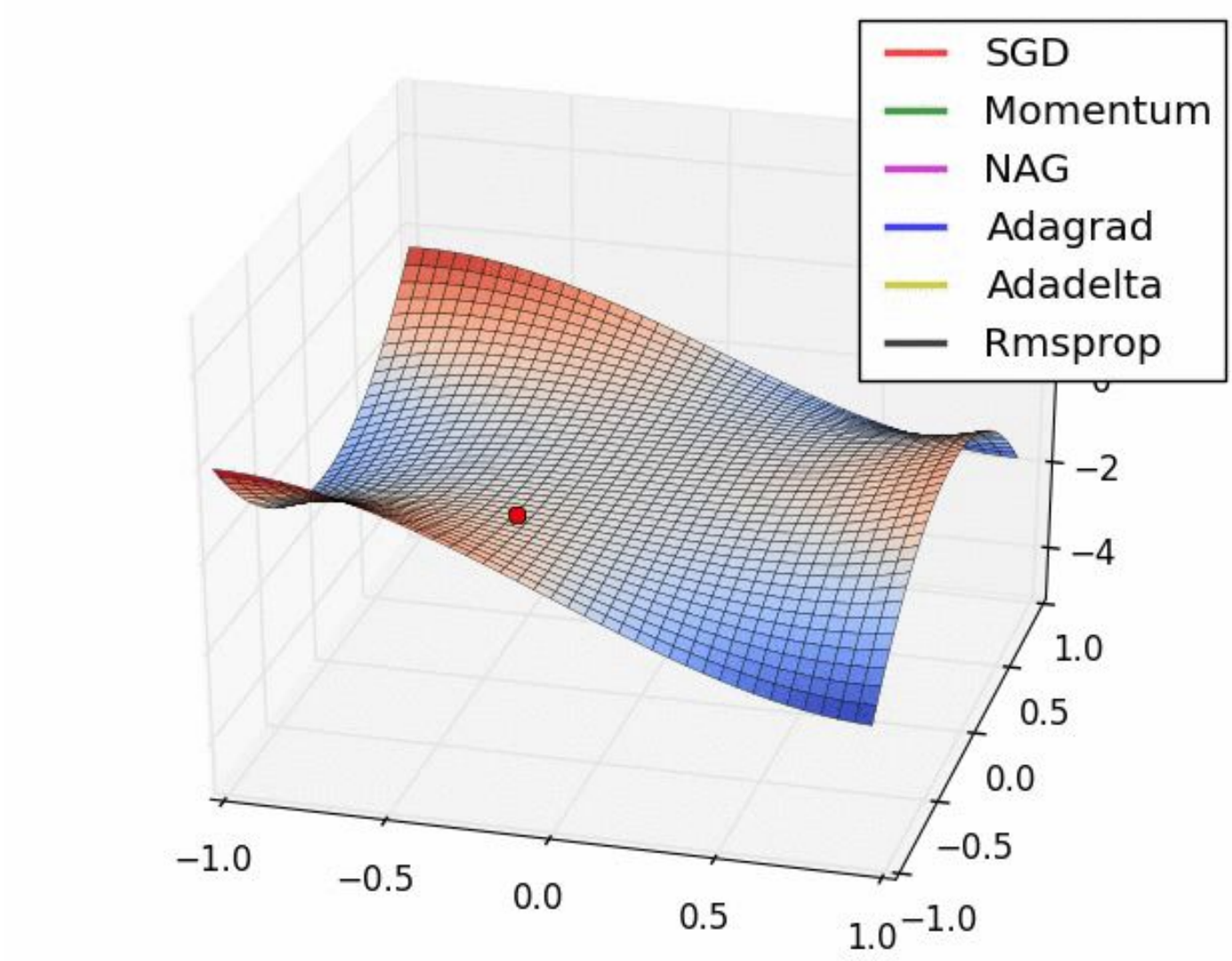
RMSProp - SGD with exponential cache

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

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

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.

Adam costs

We need to store both gradient value and cache, so we need to spend two weights of model additionally.

Second thing is need in warmup of all that caches and momentums.

Diederik et al, Adam: A Method for Stochastic Optimization, ICLR 2015

Adam is not a silver bullet

It doesn't converge on relatively simple convex task
https://arxiv.org/pdf/1904.09237

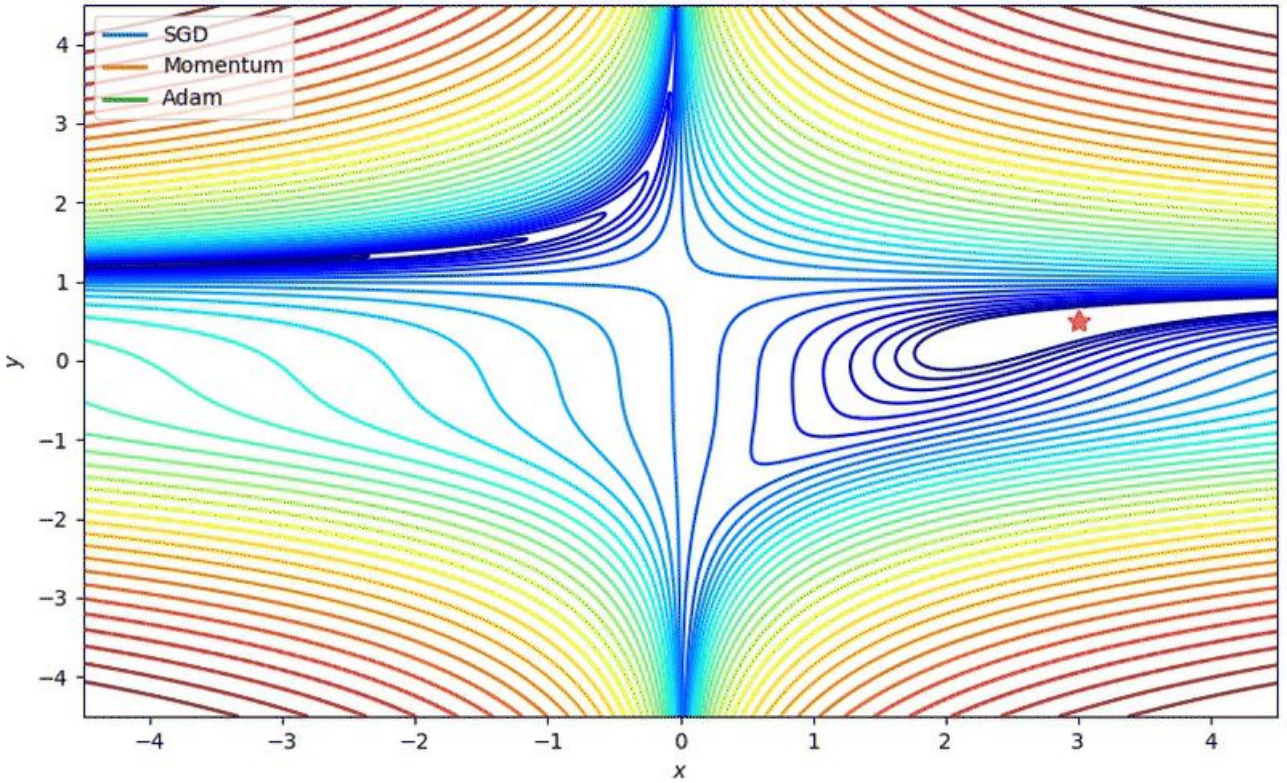
Although it can be fixed by AMSGrad or Yogi.

Also simpler proof of convergence for Adam exists.

Every instrument is relevant for its task set.

Great material on this topic https://d2l.ai/chapter_optimization/adam.html

Comparing optimizers



AdamW and NAdam

Further improvements of Adam algorithm

https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html

https://pytorch.org/docs/stable/generated/torch.optim.NAdam.html

AdamW is a stochastic optimization method that modifies the typical implementation of weight decay in Adam, by decoupling weight decay from the gradient update. To see this, L_2 regularization in Adam is usually implemented with the below modification where w_t is the rate of the weight decay at time t:

$$g_t = \nabla f(\theta_t) + w_t \theta_t$$

while AdamW adjusts the weight decay term to appear in the gradient update:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \left(\frac{1}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t + w_{t,i} \theta_{t,i} \right), \forall t$$

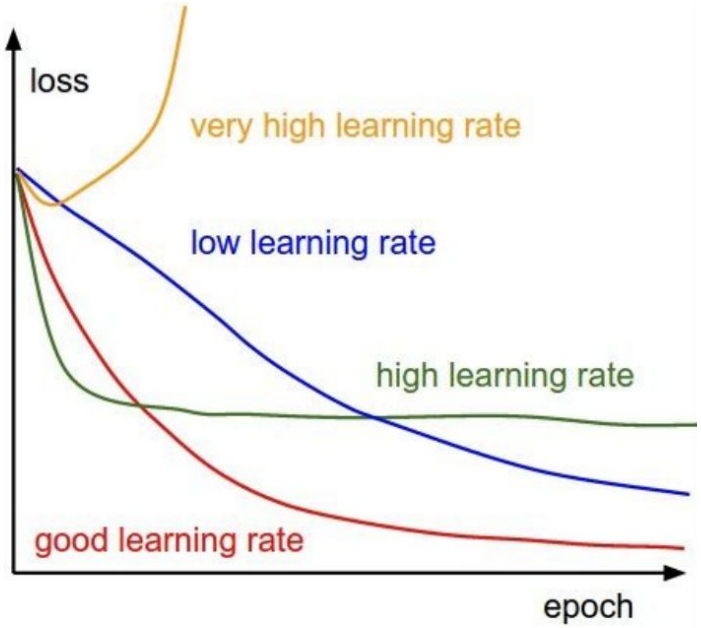
Learning rate

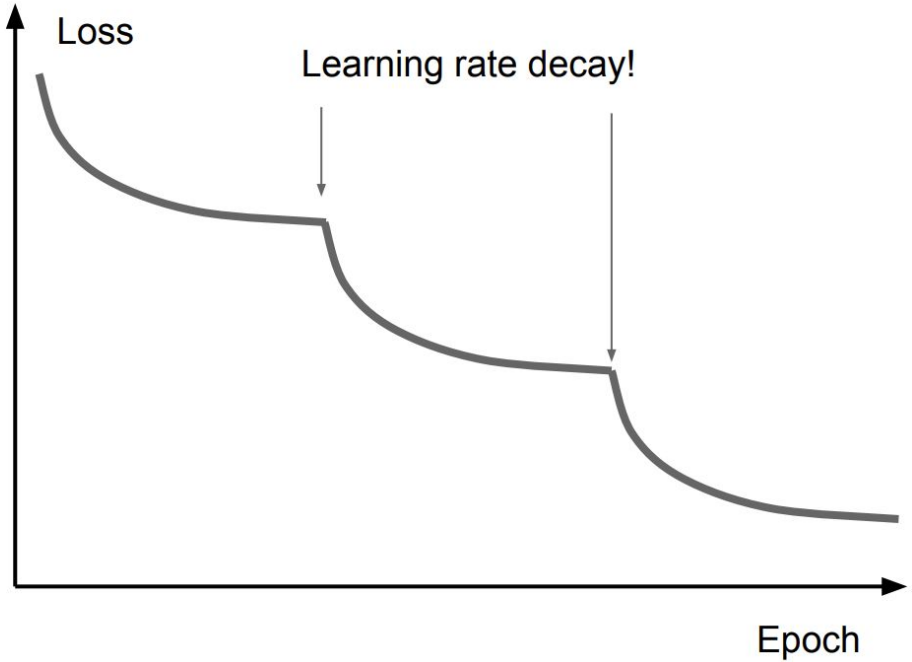
girafe
ai

03

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

Efficient learning rate value





Schedulers

PyTorch implements interface to decrease learning rate gradually
https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate

- ExponentialLR
- ReduceLROnPlateau

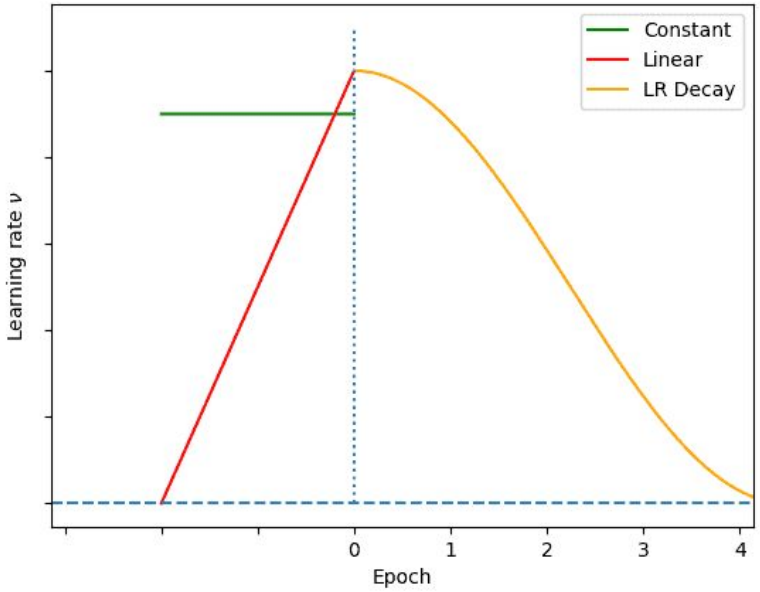
```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
scheduler = ExponentialLR(optimizer, gamma=0.9)

for epoch in range(20):
    for input, target in dataset:
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        loss.backward()
        optimizer.step()
    scheduler.step()
```

Learning Rate warm-up

Researchers proposed to use of a small LR that gradually increases to the initial LR (called Warm-up steps) to avoid poor convergence

https://www.baeldung.com/cs/learning-rate-warm-up



Notes on Transformers optimization

https://www.kaggle.com/competitions/us-patent-phrase-to-phrase-matching/discussion/330119

Also good readings
https://www.kaggle.com/thedevastator/discussion?orderBy=votes

Weights initialization

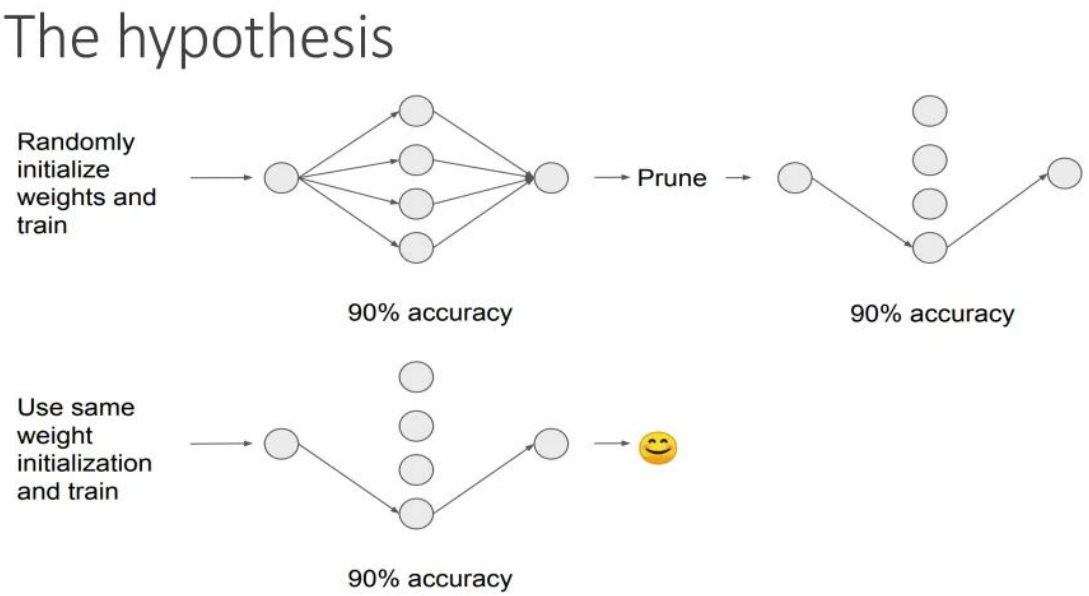
- All zero initialization
  - pitfall
- Small random numbers
- Calibrated random numbers

Pytorch module
https://pytorch.org/docs/stable/nn.init.html

$$\begin{aligned}
\mathrm{Var}(s) &= \mathrm{Var}\left(\sum_i^n w_i x_i\right) \\
&= \sum_i^n \mathrm{Var}(w_i x_i) \\
&= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= (n\mathrm{Var}(w))\ \mathrm{Var}(x)
\end{aligned}$$

Lottery Ticket Hypothesis

A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.



https://medium.com/@sayan112207/understanding-the-lottery-ticket-hypothesis-7d303f60616c
https://www.lesswrong.com/posts/Z7R6jFjce3J2Ryj44/exploring-the-lottery-ticket-hypothesis

Sum up: optimization

- Adam is great basic choice
- Even for RMSProp learning rate matters
- Use learning rate decay
- Monitor your model quality
- Weights initialization is done well by default
    - but you need to know how

Inductive bias

girafe ai

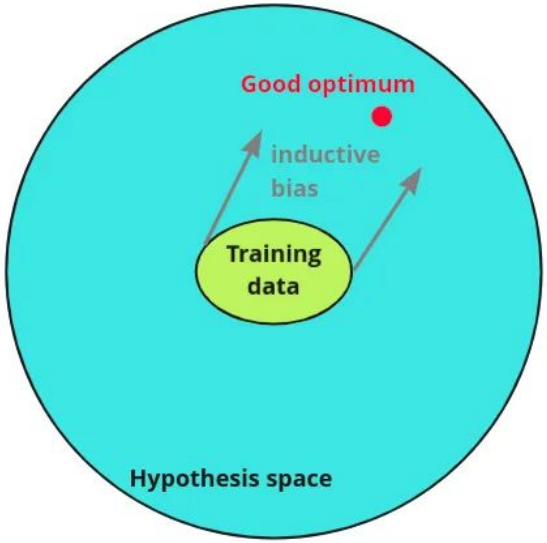
04

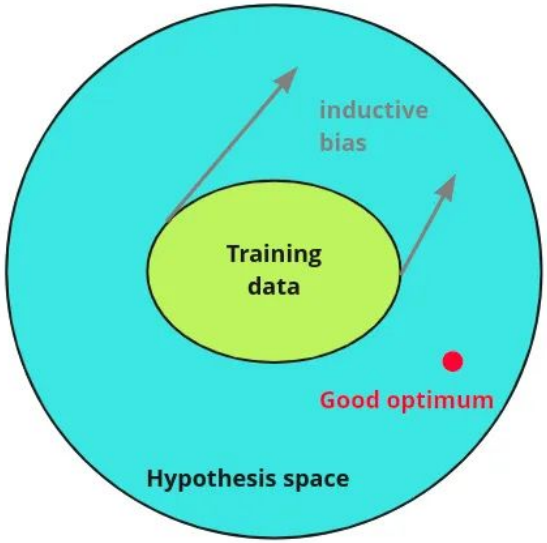
Inductive bias

set of (explicit or implicit) assumptions made by a learning algorithm in order to perform induction, that is, to generalize a finite set of observation (training data) into a general model of the domain.

source

- knn
- svm
- normalization
- dropout
- regularization
- augmentation



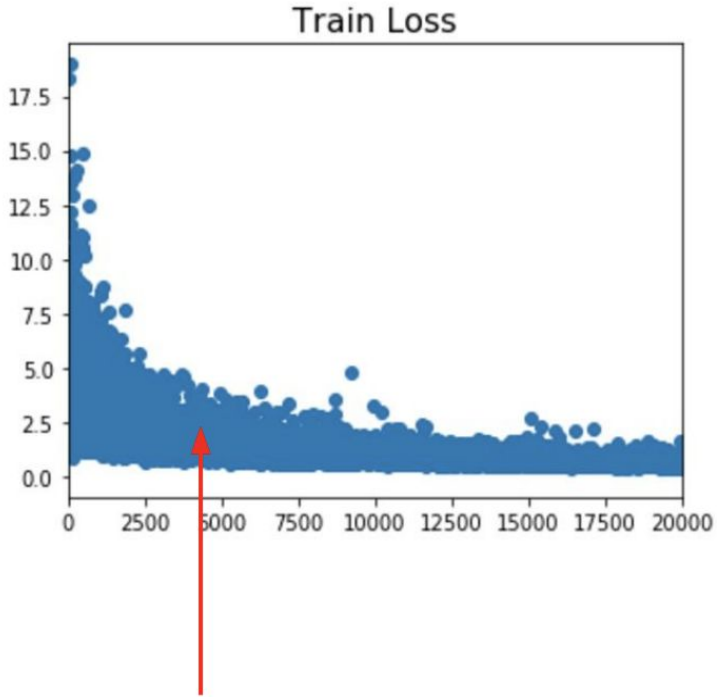


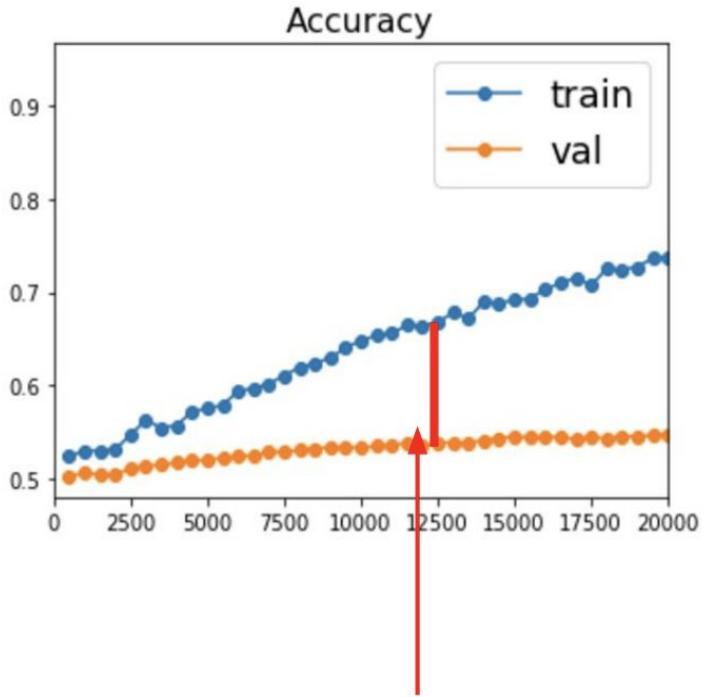
more info

Normalization

girafe
ai

05

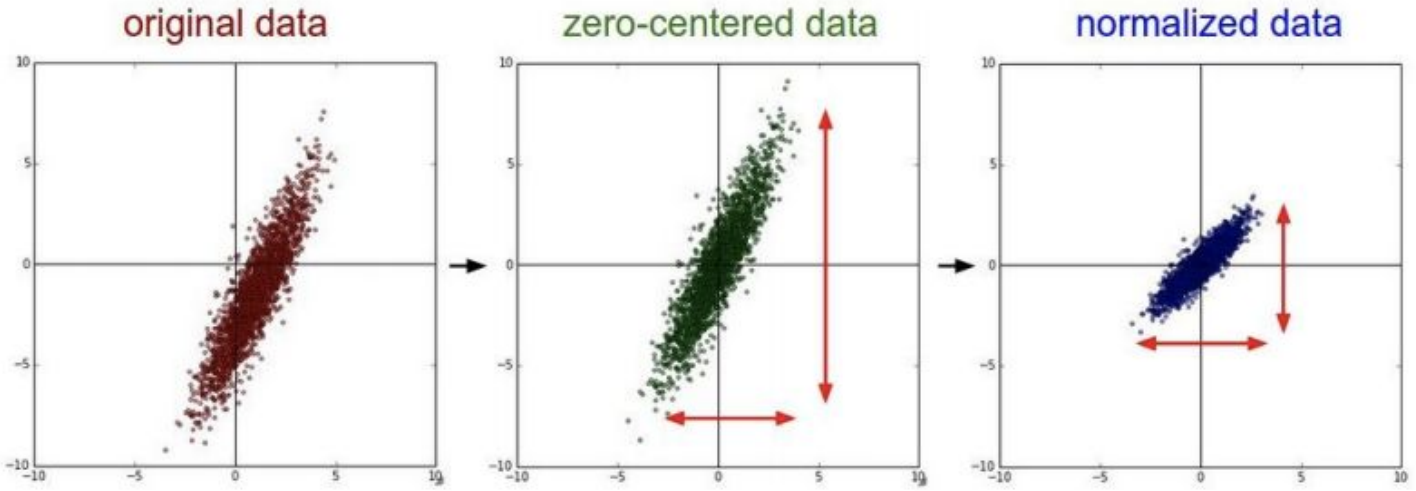




Better optimization algorithms help reduce training loss

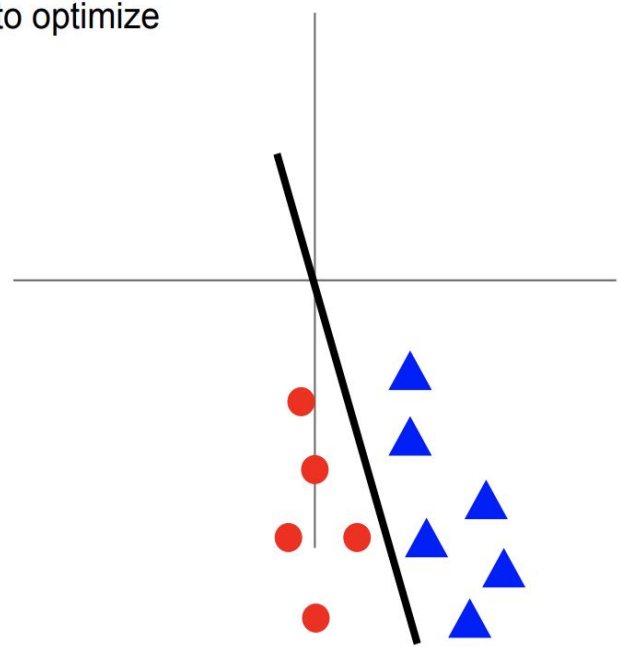
But we really care about error on new data - how to reduce the gap?

Data normalization

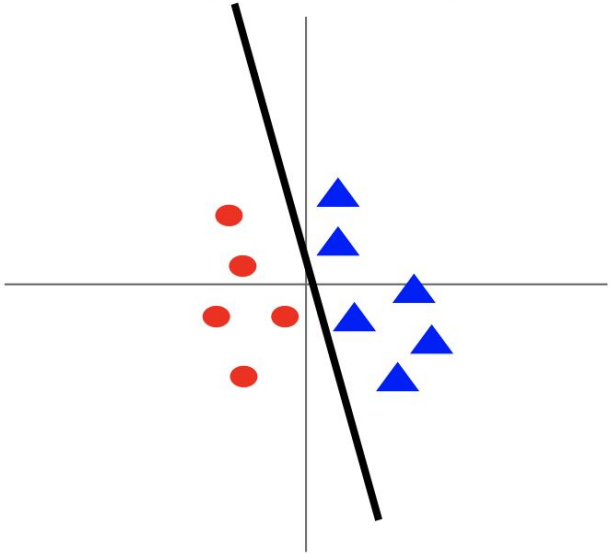


Data normalization

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



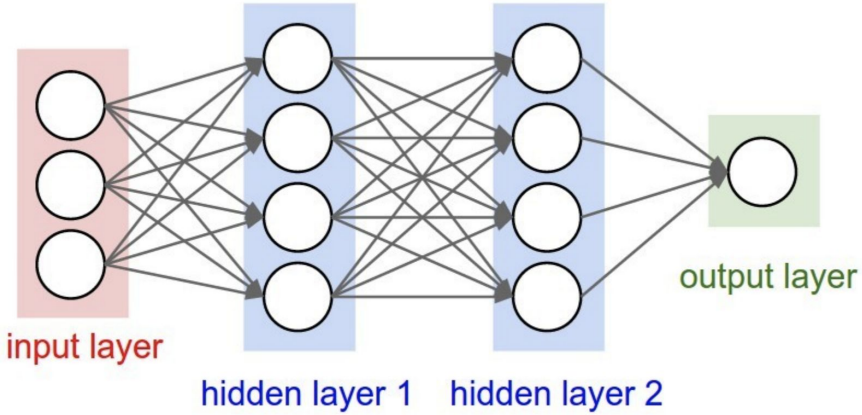
After normalization: less sensitive to small changes in weights; easier to optimize



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 its inputs. Some of them become larger, some – smaller
- Now the neuron needs to be re-tuned for it's new inputs

[Ioffe et al, Batch Normalization: Accelerating Deep Network Training by reducing Internal Covariate Shift, 2015]

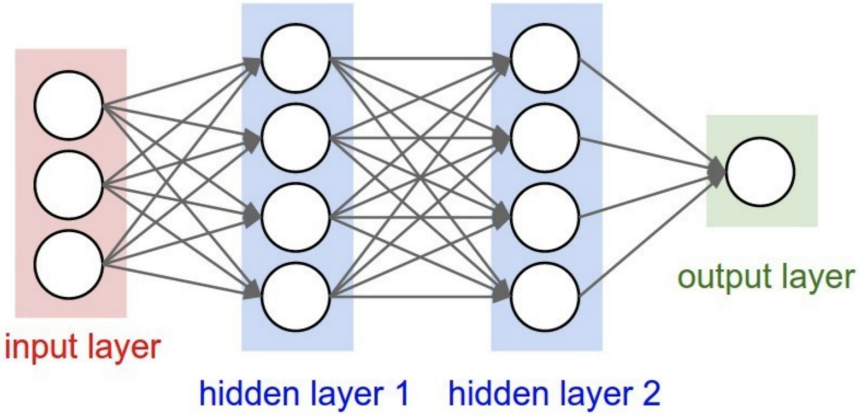


Batch normalization

Another view on BN mechanism

https://arxiv.org/pdf/1805.11604

https://arxiv.org/pdf/1902.08129



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: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

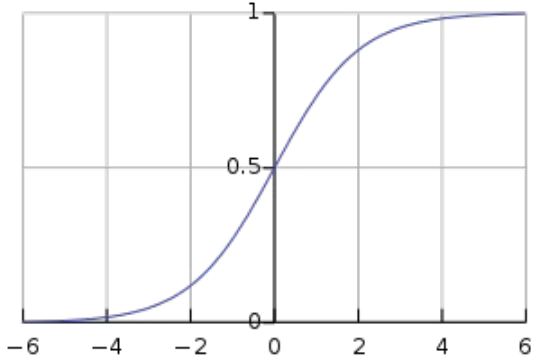
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

Batch normalization



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

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

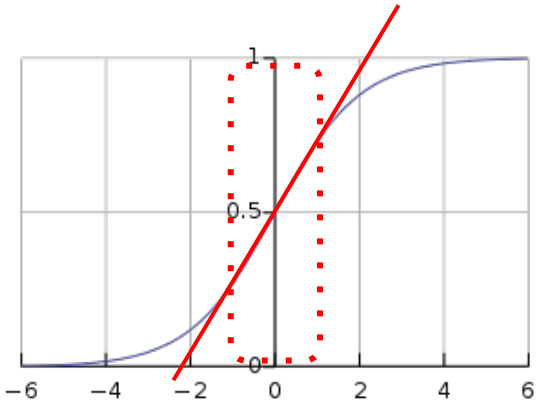
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

Batch normalization



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

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

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

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

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

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

Batch normalization

Original algorithm (2015)

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

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

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

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

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

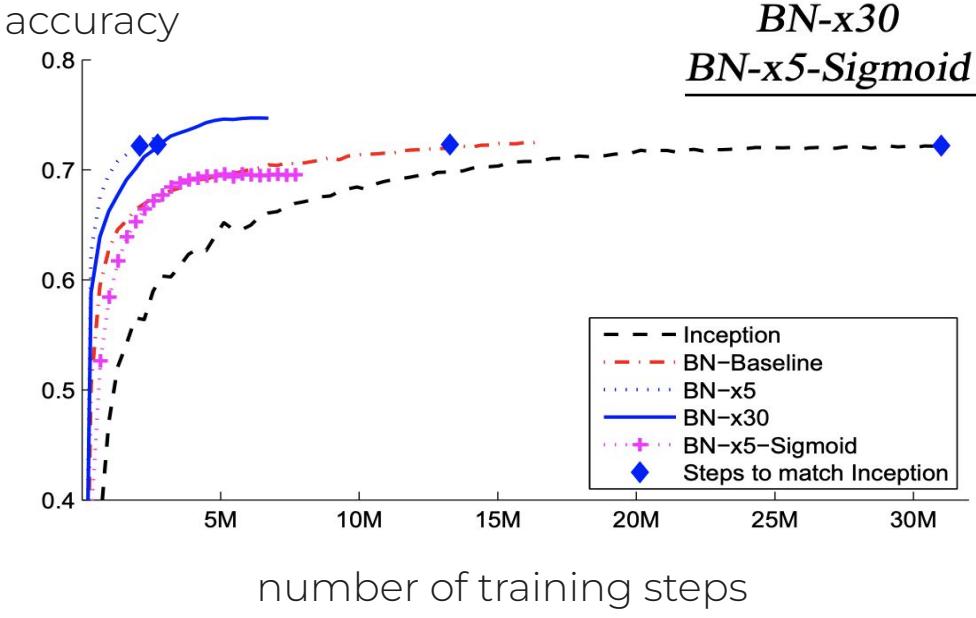
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

What is this?

This transformation should be able to represent the identity transform.

Training acceleration

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%



Batchnorm in Pytorch

https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html

https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

Parameters

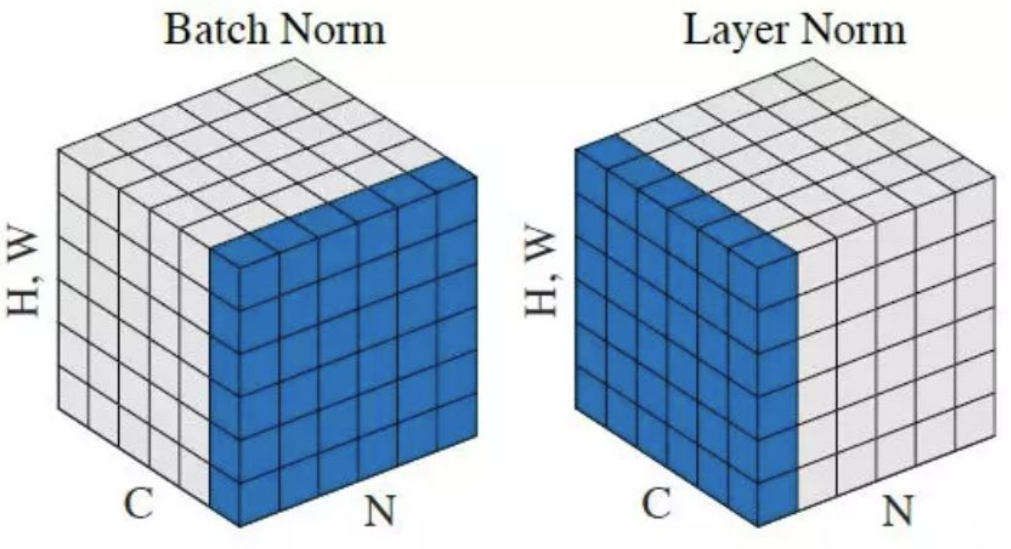
- **num_features** (*int*) – C from an expected input of size (N, C, H, W)
- **eps** (*float*) – a value added to the denominator for numerical stability. Default: 1e-5
- **momentum** (*Optional*[*float*]) – the value used for the running_mean and running_var computation. Can be set to `None` for cumulative moving average (i.e. simple average). Default: 0.1
- **affine** (*bool*) – a boolean value that when set to `True`, this module has learnable affine parameters. Default: `True`
- **track_running_stats** (*bool*) – a boolean value that when set to `True`, this module tracks the running mean and variance, and when set to `False`, this module does not track such statistics, and initializes statistics buffers `running_mean` and `running_var` as `None`. When these buffers are `None`, this module always uses batch statistics. in both training and eval modes. Default: `True`

Layer normalization

Same idea, but applied object-wise

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

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



Regularization

girafe ai

06

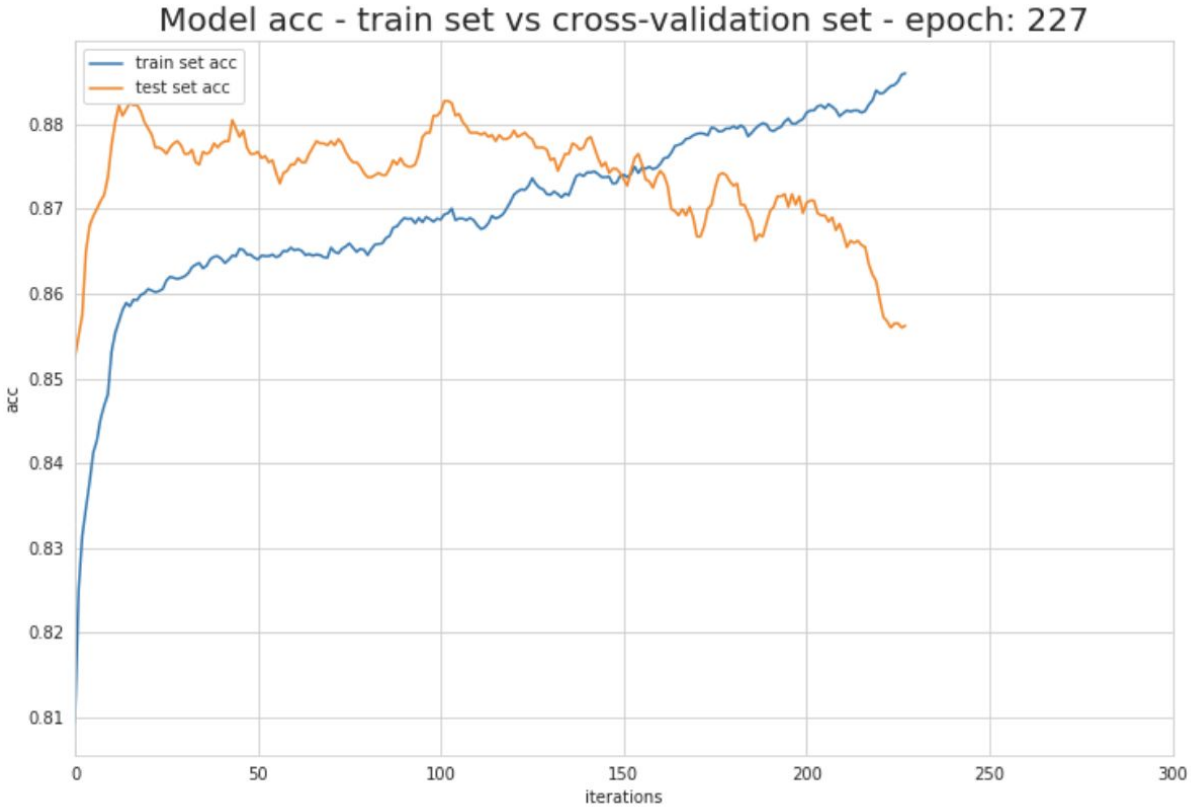
What is regularization anyway?

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

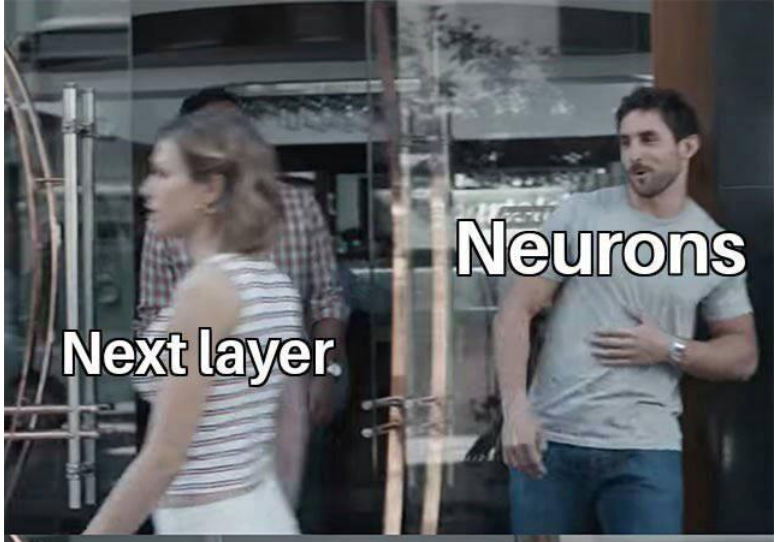
$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

Adding some extra term to the loss function.

Common cases:

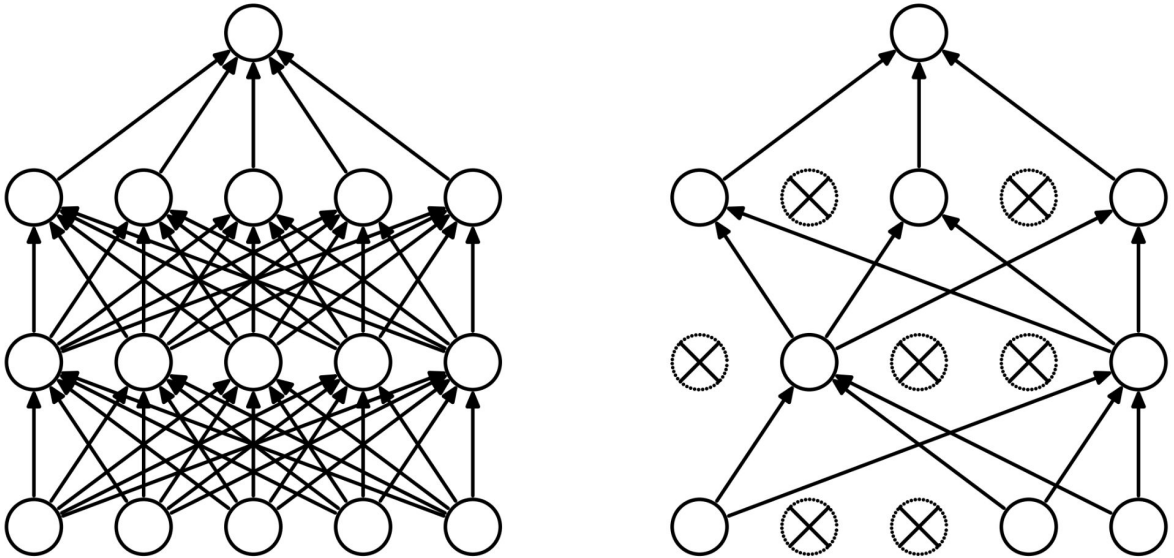
- L2 regularization: $R(W) = \|W\|_2^2$
- L1 regularization: $R(W) = \|W\|_1$
- Elastic Net (L1 + L2): $R(W) = \beta\|W\|_2^2 + \|W\|_1$

Dropout



Dropout

During each forward pass set each input feature value to zero with probability p.



(a) Standard Neural Net

(b) After applying dropout.

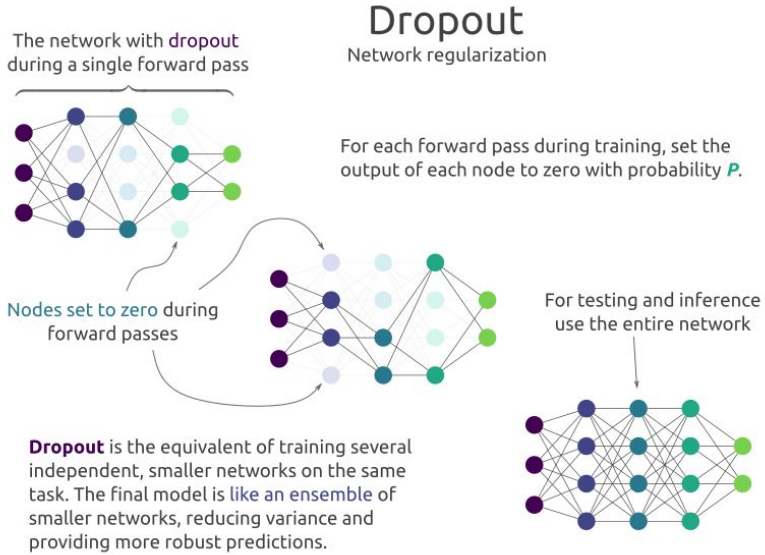
Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Srivastava et al, Journal of Machine Learning Research, 2014

Dropout

Some neurons are “dropped” during training.

Prevents overfitting.

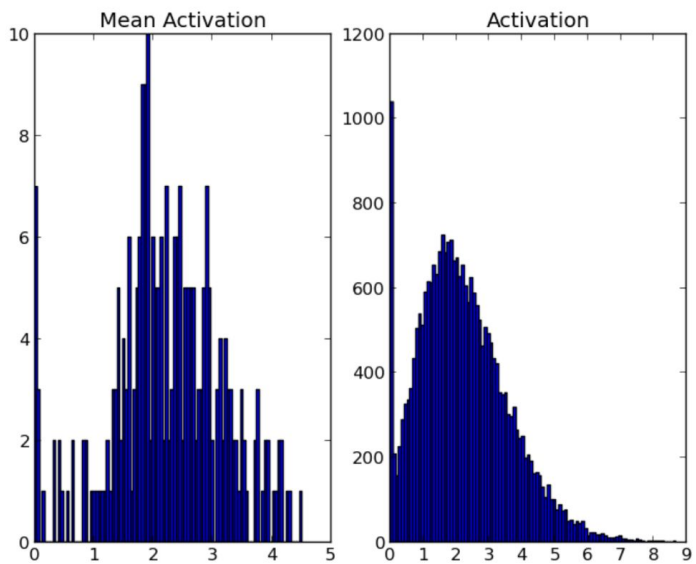


Actually, on test case output should be normalized. See sources for more info.

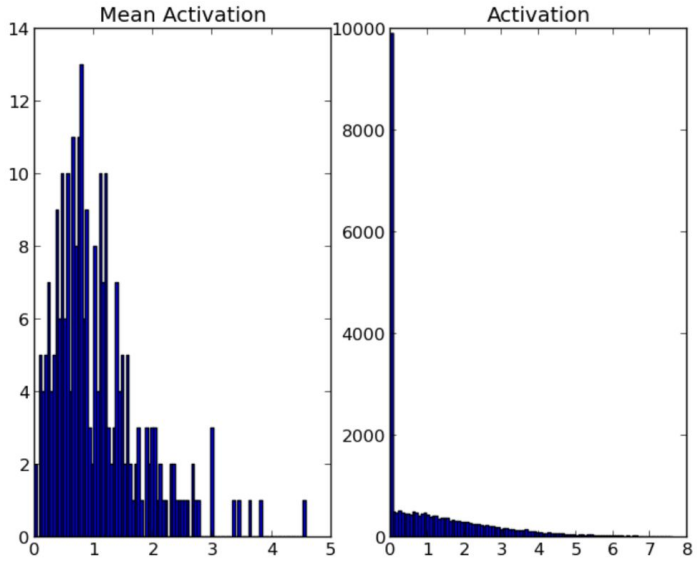
Dropout doesn’t decrease number of FLOPS in neural network!

Effect on weights sparsity

Without dropout we have a lot of activated units, with it most activations are zero, also mean activation mode moves from 2 to 0.7



(a) Without dropout



(b) Dropout with $p = 0.5$.

Dropout inference

For dropout probability p we have to scale distribution back for inference.

It is very simple - just divide by `(1-p)`

https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

Dropout usage

Usually applied to last layers of neural networks.

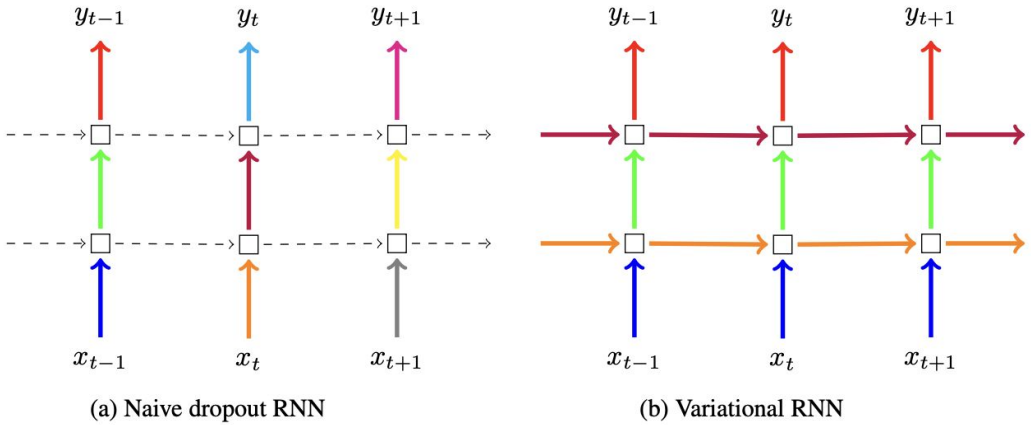
Usually not there are 1 to 3 of them in architecture.

Variational dropout

Also dropout idea became popular in Bayesian community and they suggested several models based on this idea.

- https://arxiv.org/pdf/1506.02557
- https://arxiv.org/pdf/1512.05287v5
- https://arxiv.org/pdf/1701.05369
- https://arxiv.org/pdf/1506.02142v6

Check out Bayes methods course!

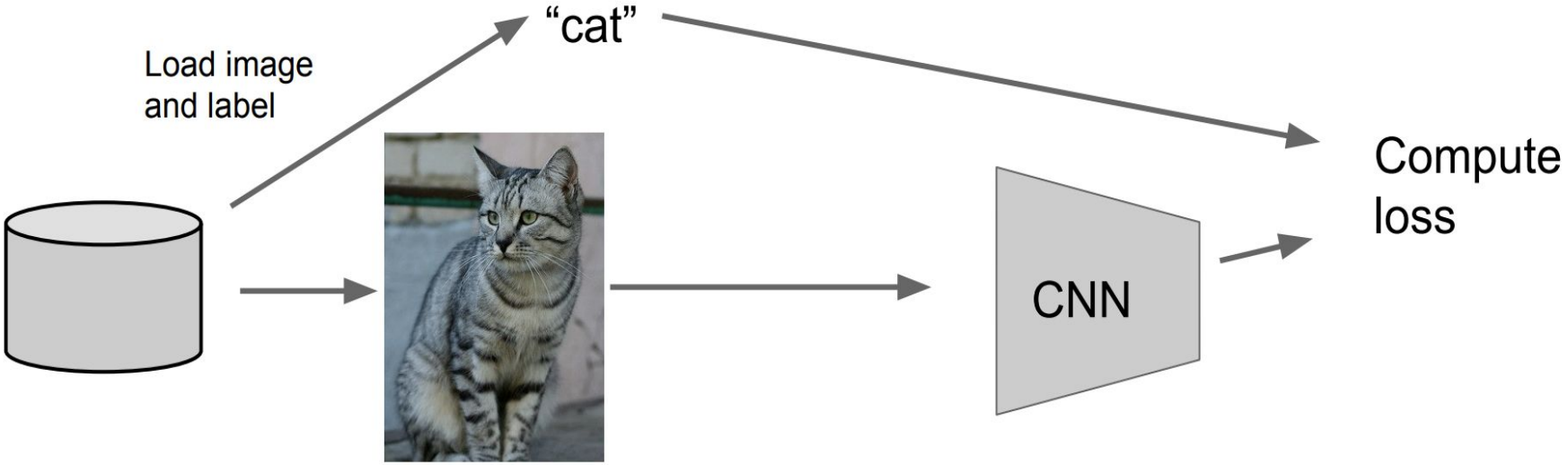


Augmentation

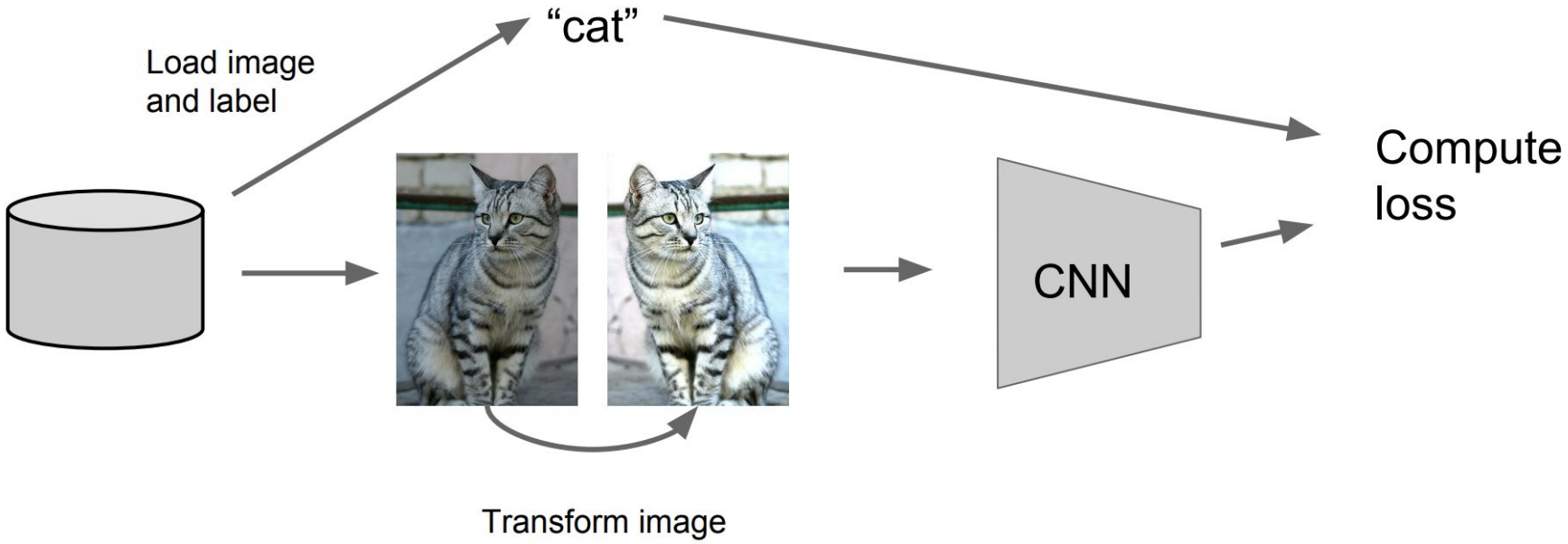
girafe
ai

07

Regular data flow



Data augmentation



Many ways to augment

Original image

augmentation

Horizontal Flip

Crop

Median Blur

Contrast

Hue / Saturation / Value

Gamma

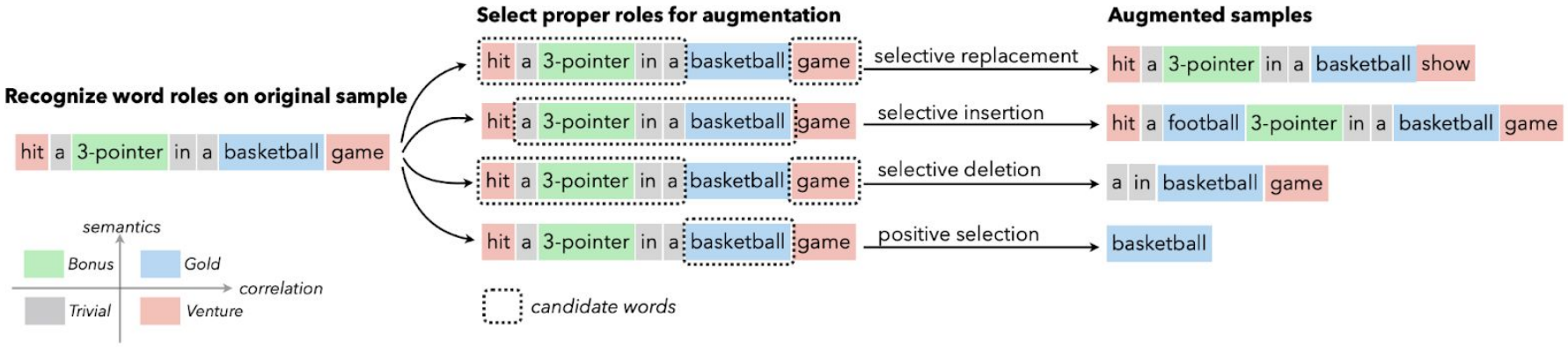
Image augmentations

- https://pytorch.org/vision/stable/transforms.html
- https://albumentations.ai/
- https://augmentor.readthedocs.io/en/stable/
- https://imgaug.readthedocs.io/en/stable/

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

Text augmentations



- 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

Materials used

- https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Thanks for attention!

Questions?