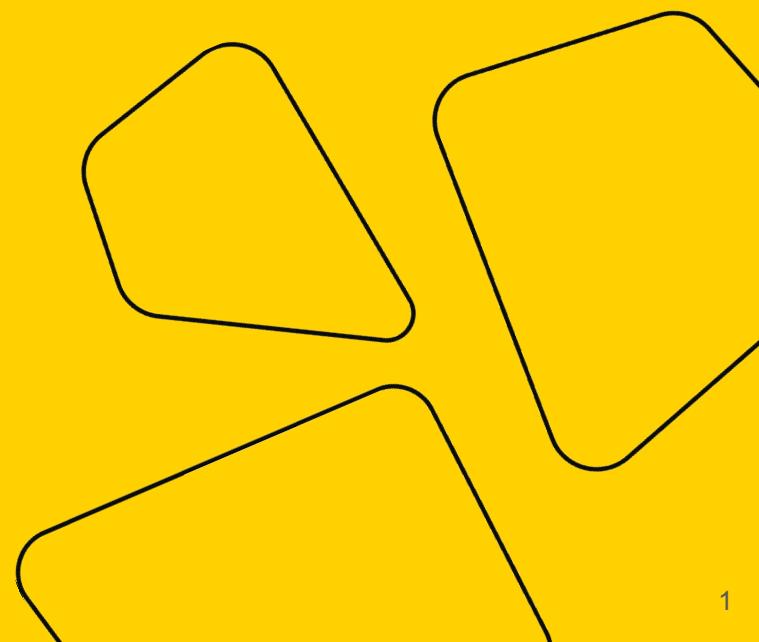
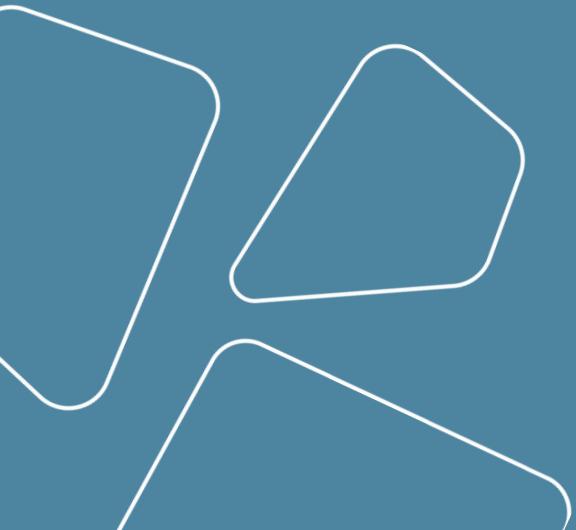


Lecture 09: NN Optimization and Regularization

Iurii Efimov



Outline



1. Optimizers
 - a. SGD
 - b. Momentum
 - c. RMSProp
 - d. Adam
2. Data normalization
 - a. Batch Norm
 - b. Layer Norm
3. Regularization
 - a. Dropout
4. Augmentation
 - a. Images
 - b. Texts

Recap

girafe
ai

ot

Once again: nonlinearities

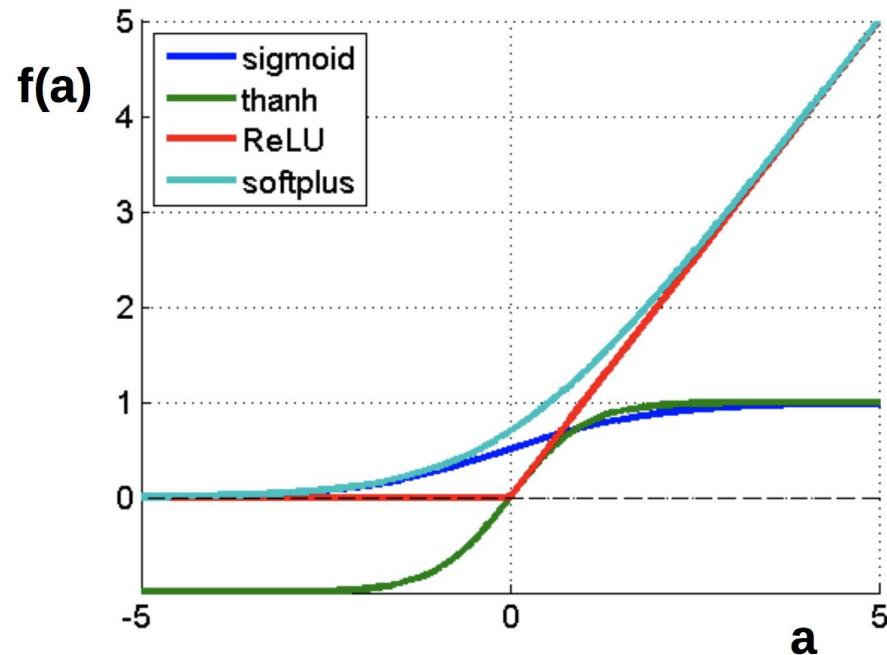


$$f(a) = \frac{1}{1 + e^{-a}}$$

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

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

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



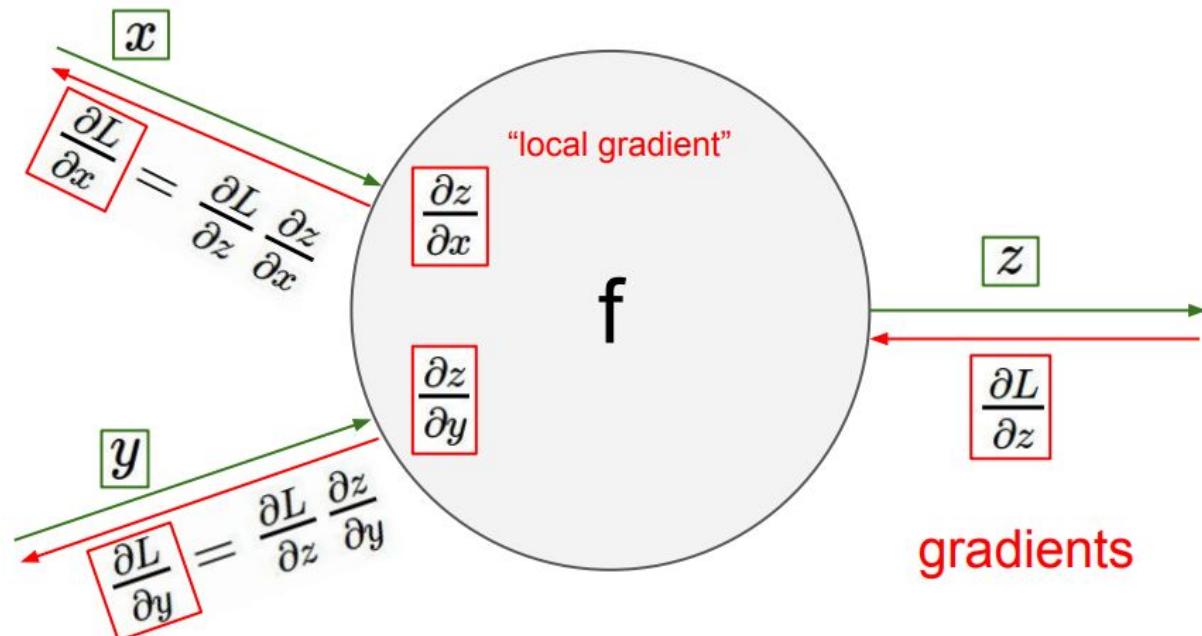
Backpropagation and chain rule



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.





Recap: generally accepted terms

- Layer – a building block for NNs :
 - Dense/Linear/FC layer: $f(x) = Wx+b$
 - Nonlinearity layer: $f(x) = \sigma(x)$
 - Input layer, output layer
 - A few more we will cover later
- Activation function – function applied to layer output
 - Sigmoid
 - \tanh
 - ReLU
 - Any other function to get nonlinear intermediate signal in NN
- Backpropagation – a fancy word for “chain rule”

Optimizers

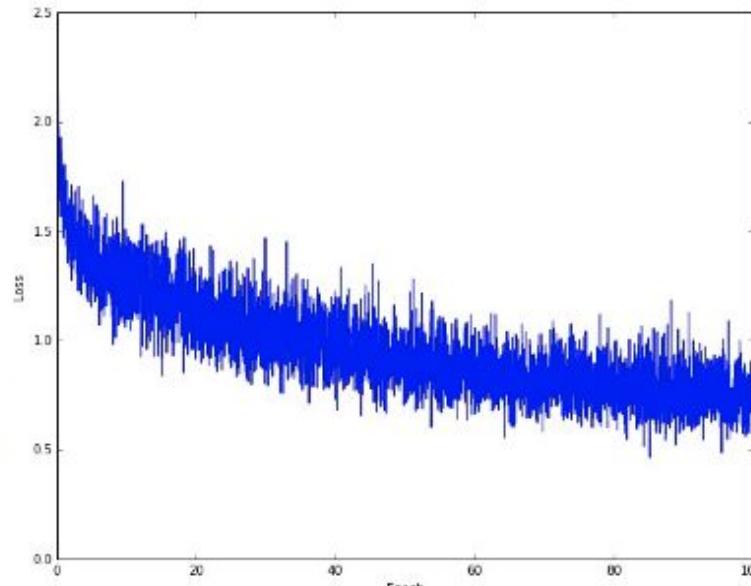
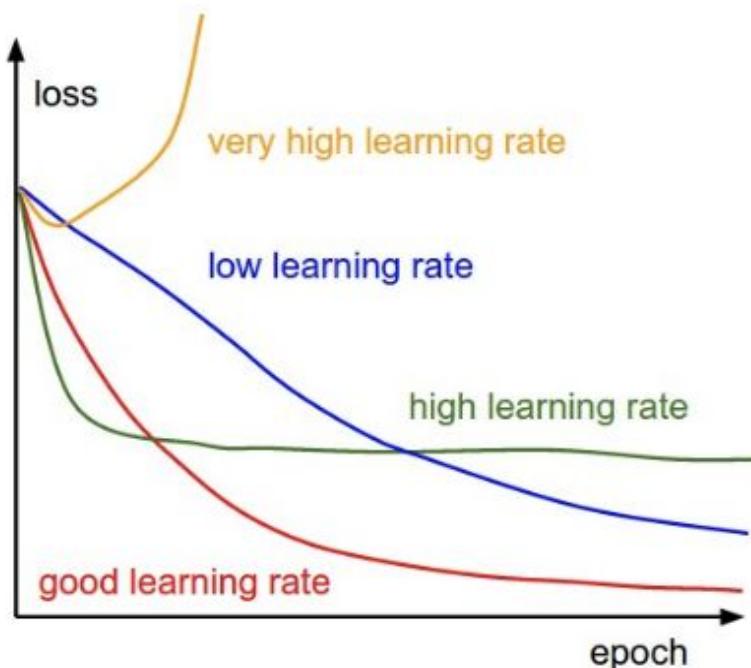
girafe
ai

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.

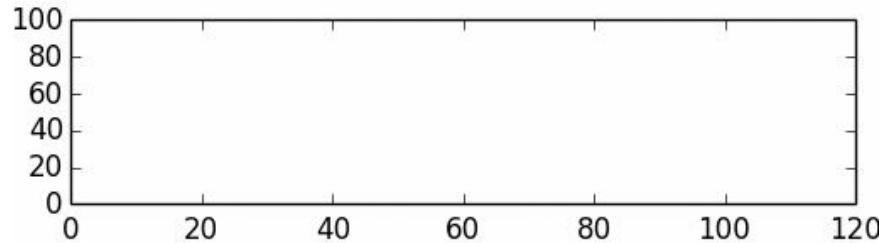
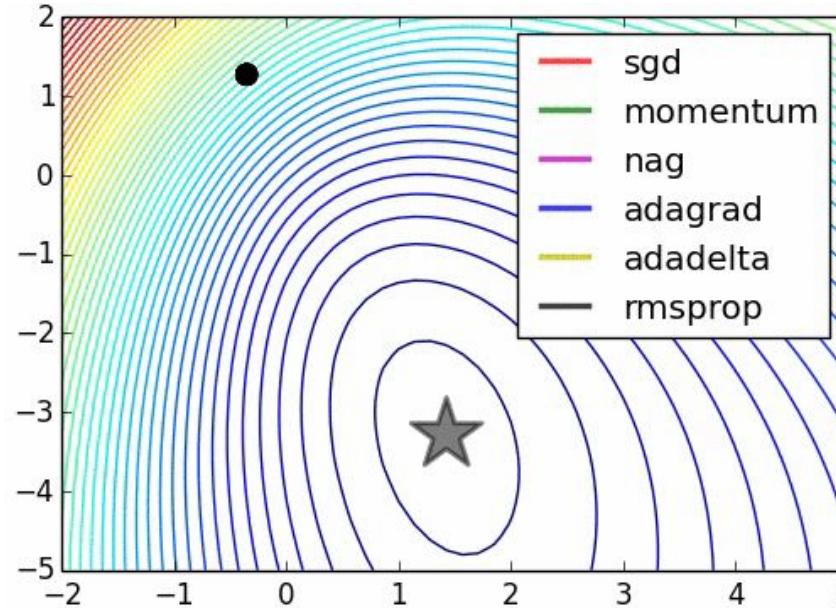


Optimizers



There are lots of optimizers:

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



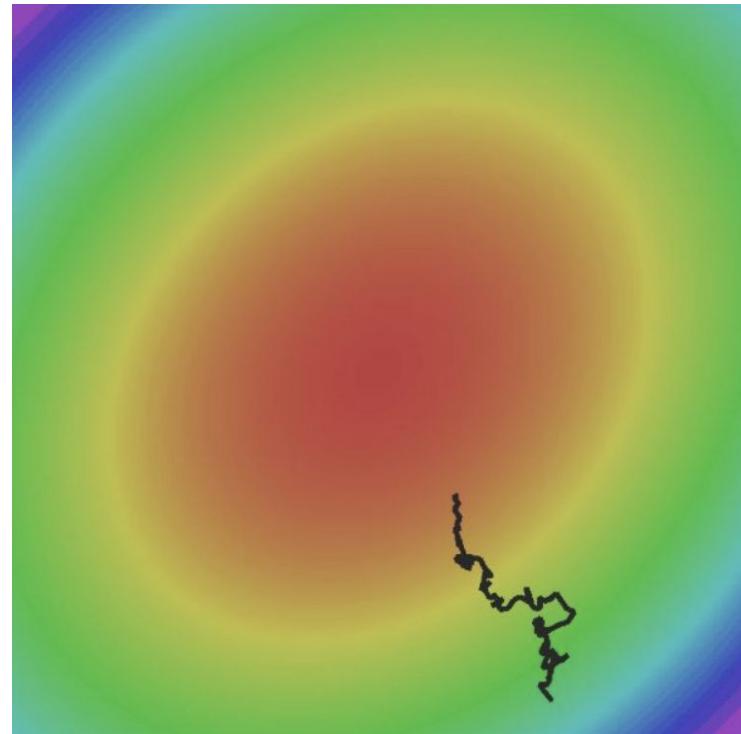
Optimization: SGD



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

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

Averaging over mini batches
=> noisy gradient



First idea: momentum



Simple SGD

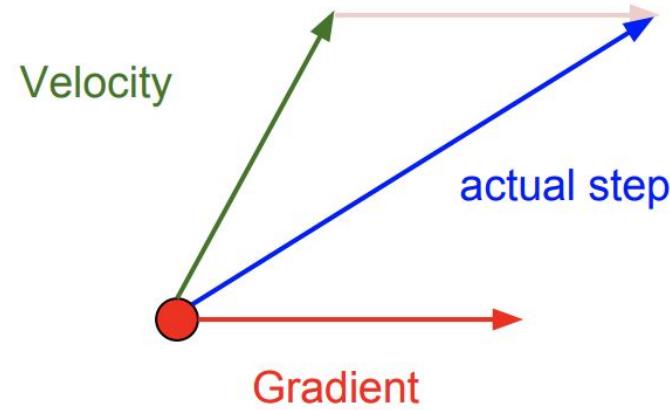
$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD with momentum

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

$$x_{t+1} = x_t - \alpha v_{t+1}$$

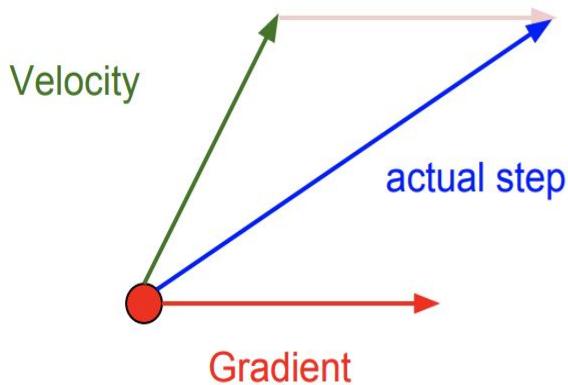
Momentum update:



Nesterov momentum



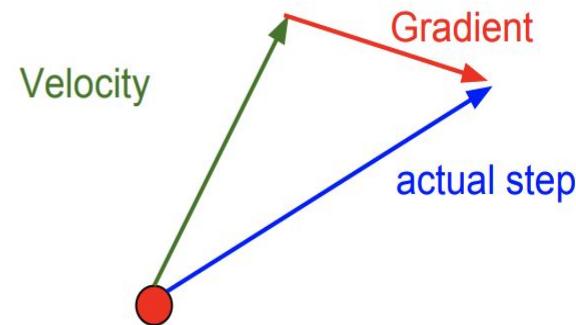
Momentum update:



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

$$x_{t+1} = x_t - \alpha v_{t+1}$$

Nesterov Momentum

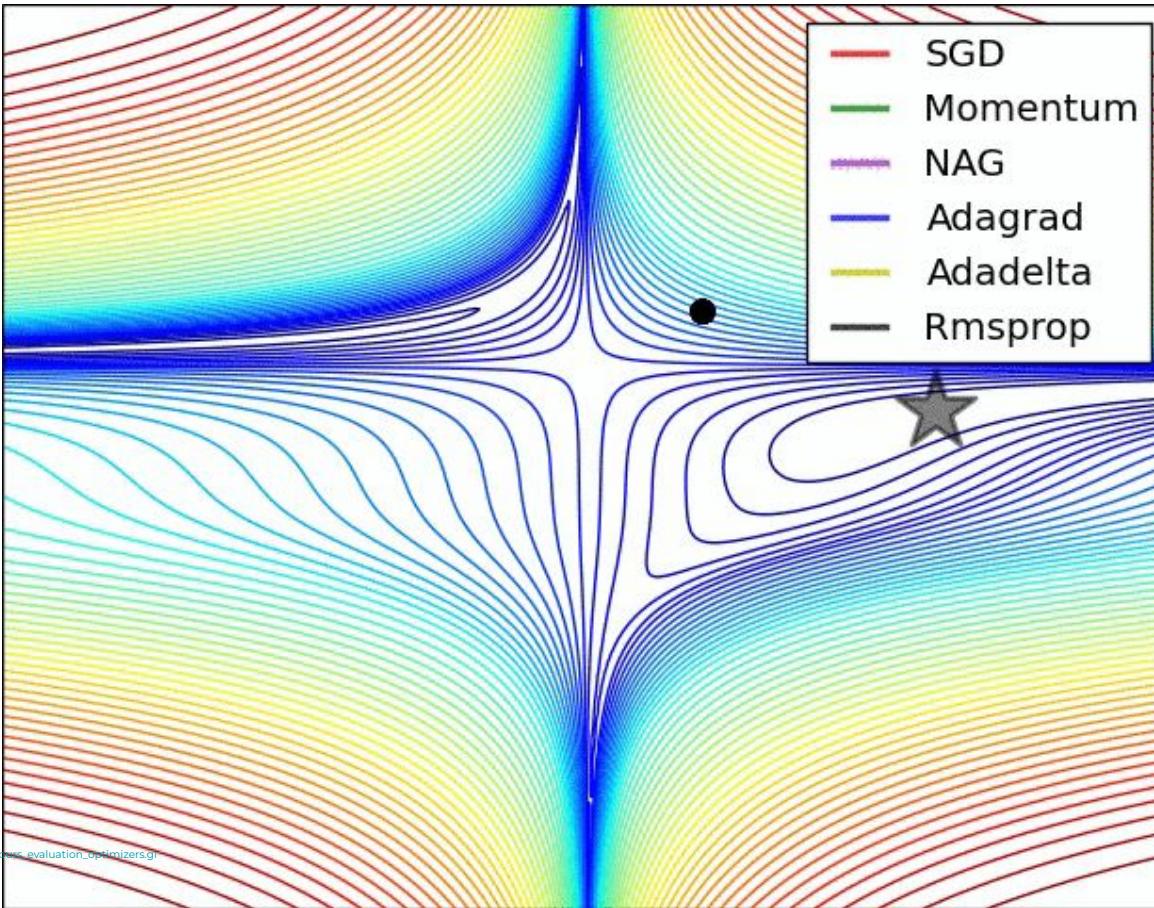


$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

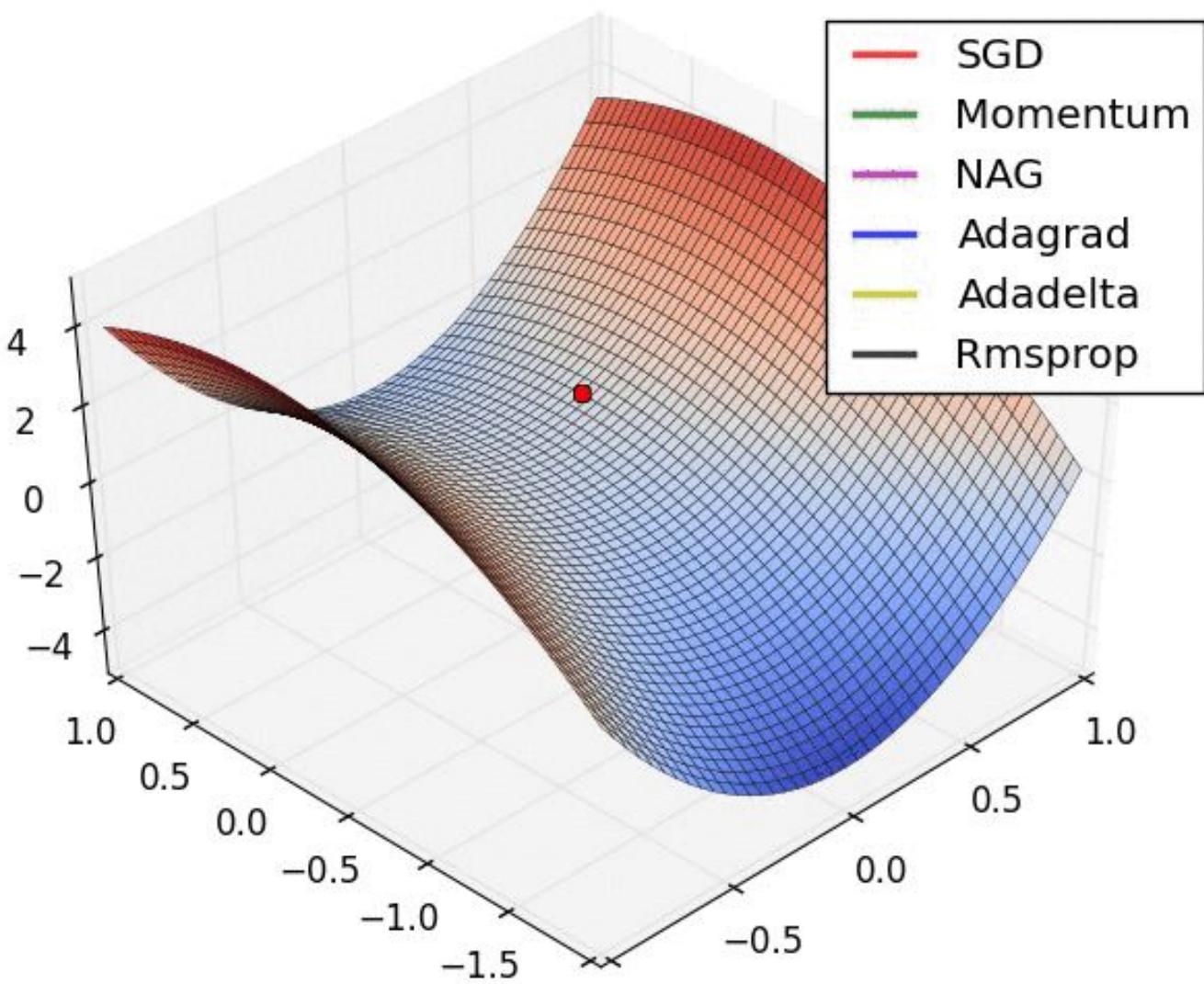
$$x_{t+1} = x_t + v_{t+1}$$



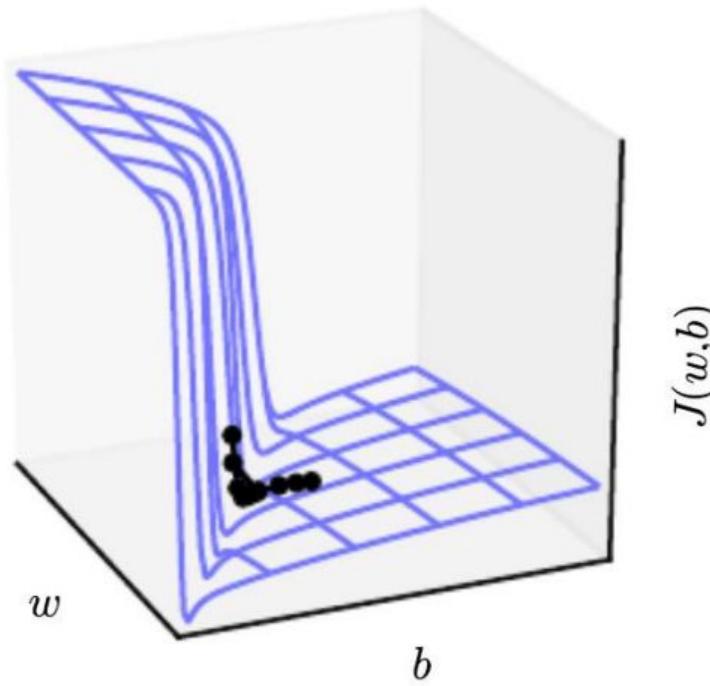
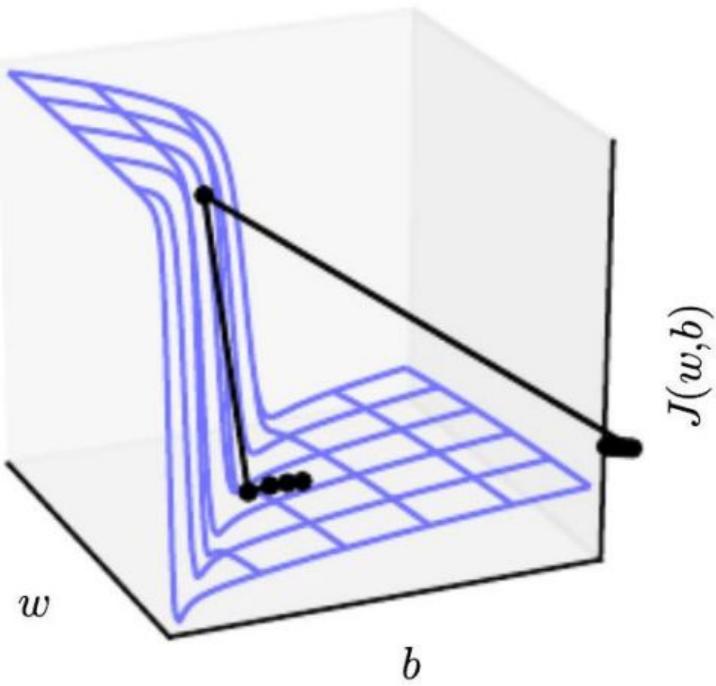
Comparing momentums



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



Second idea: step size



Different dimensions are different



RMSProp - SGD with exponential cache

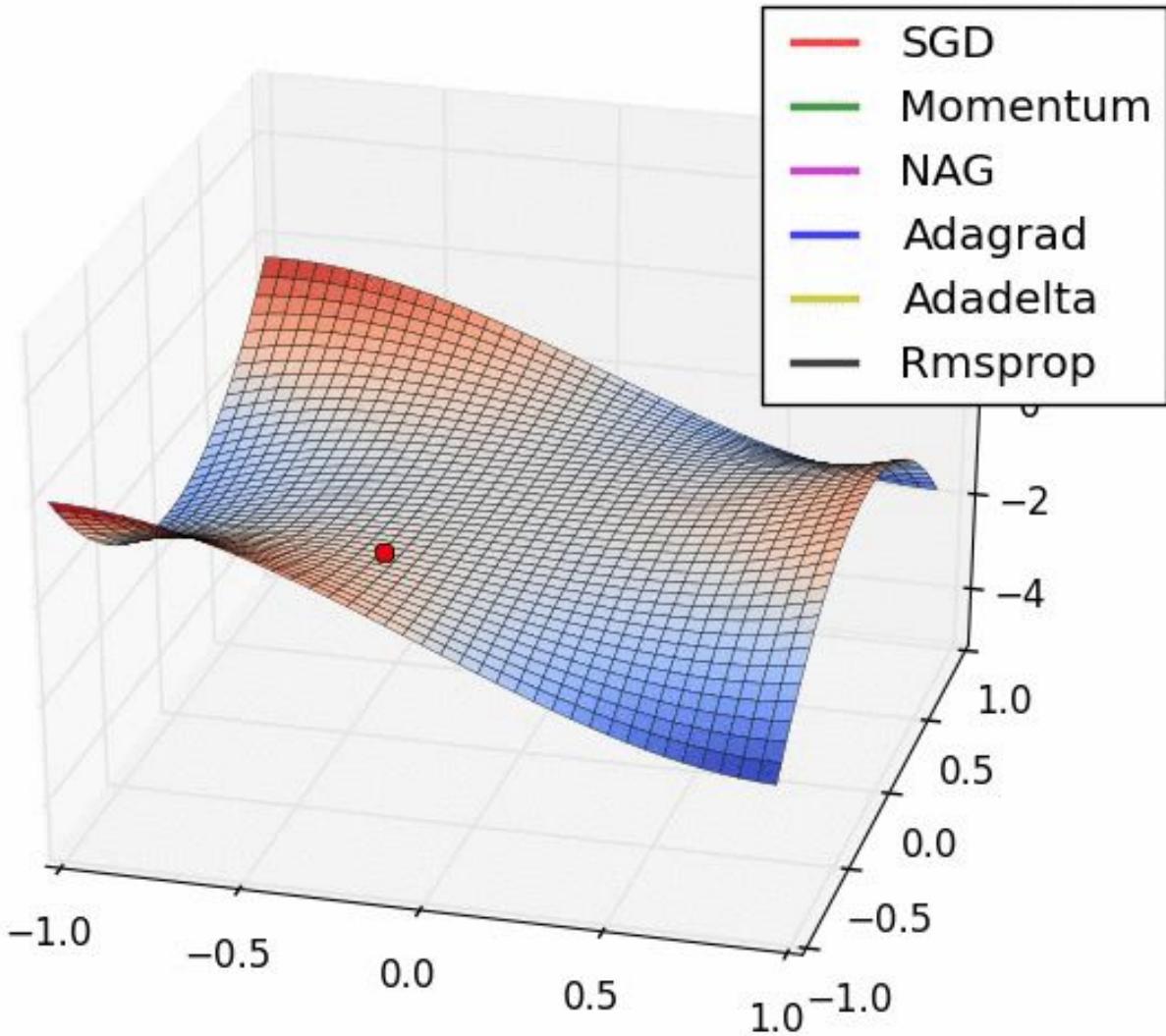
$$\text{cache}_{t+1} = \beta \text{cache}_t + (1 - \beta)(\nabla f(x_t))^2$$

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

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





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>

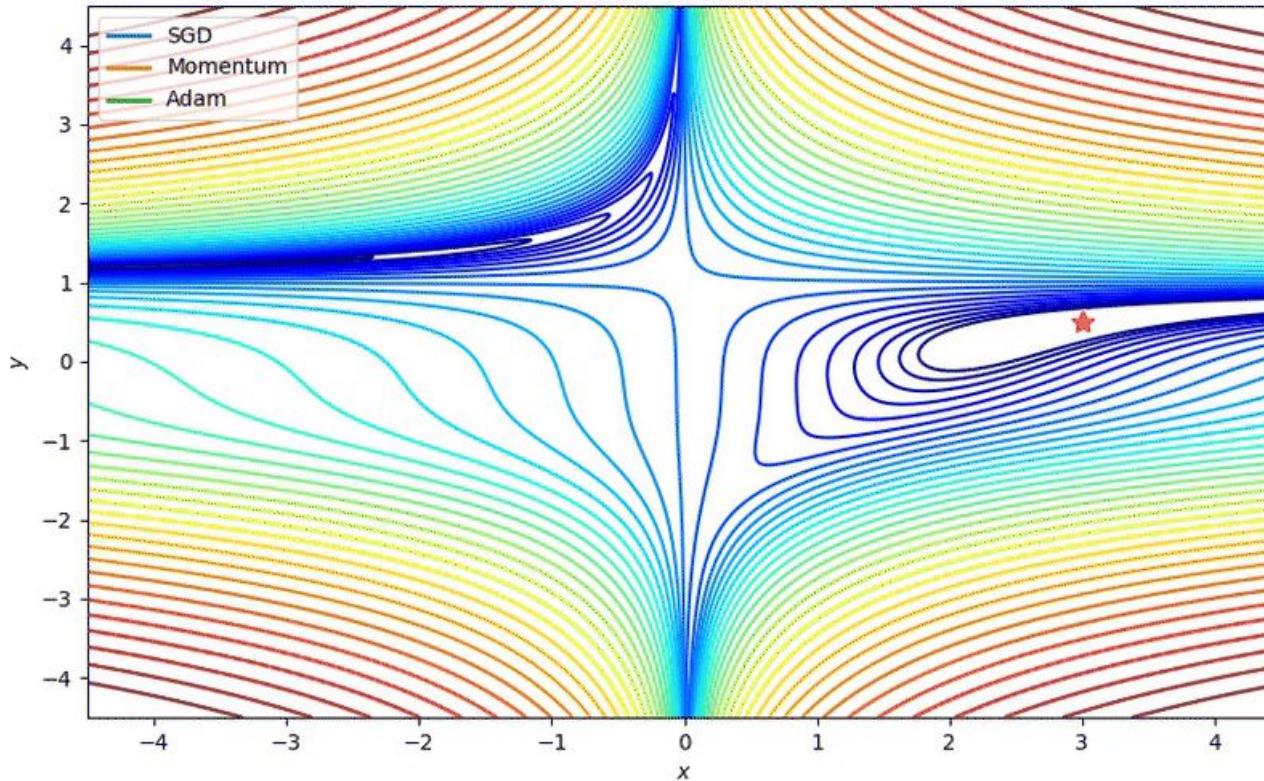
Although it can be fixed by AMSGrad or [Yogi](#).

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



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

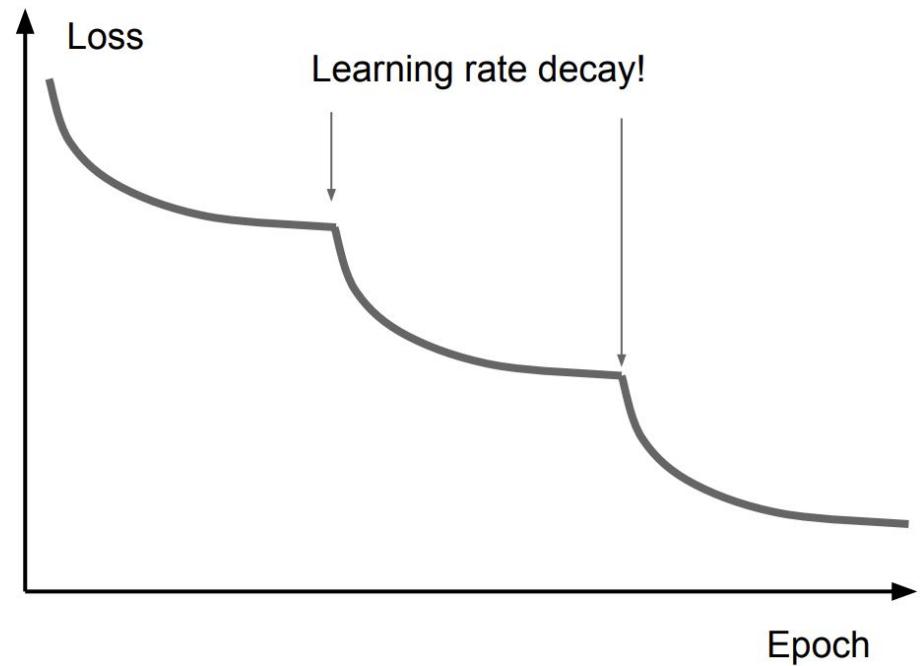
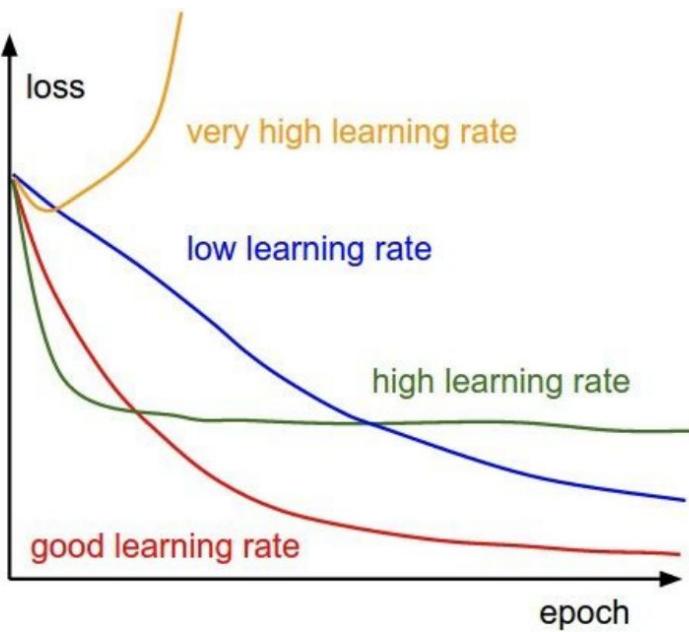


119



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

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)

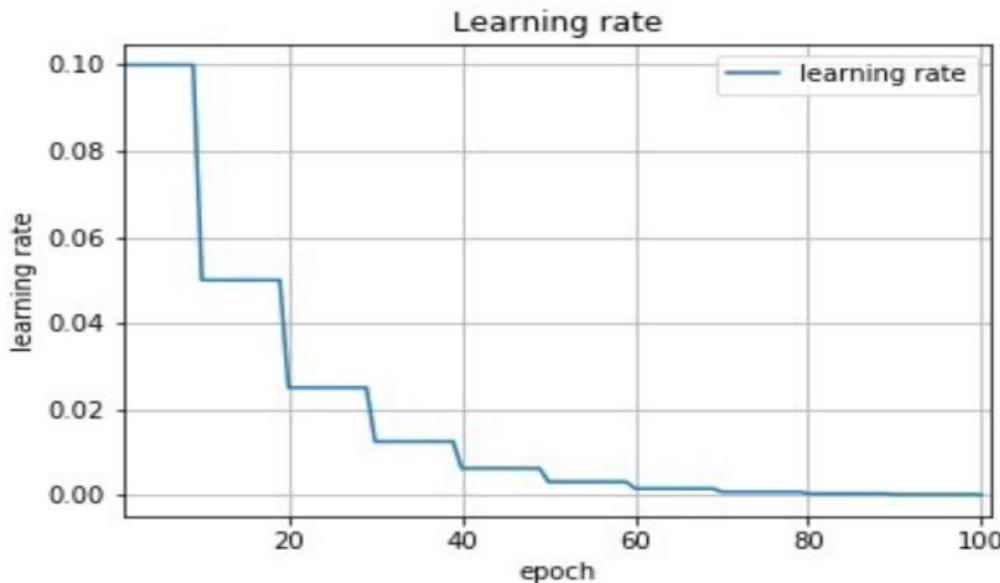
for epoch in range(20):
    for input, target in dataset:
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        loss.backward()
        optimizer.step()
    scheduler.step()
```



Learning rate schedulers

Step decay

$$lr(\text{epoch}) = lr_0 \lfloor \frac{\text{epoch}}{\text{step}} \rfloor$$

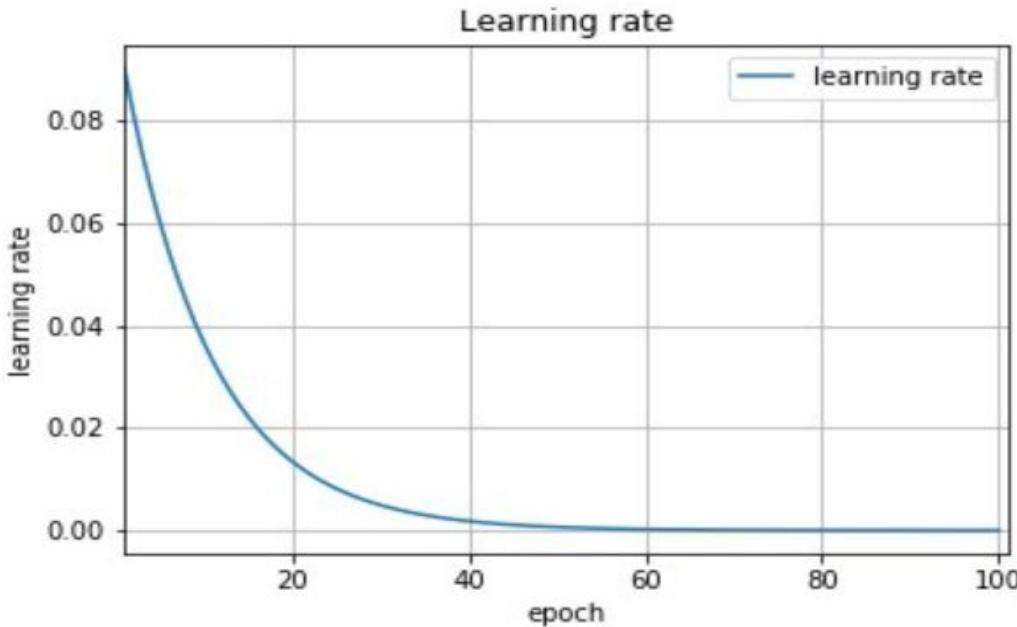




Learning rate schedulers

Exponential decay

$$lr(\text{epoch}) = lr_0 \cdot e^{-\alpha \cdot \text{epoch}}$$

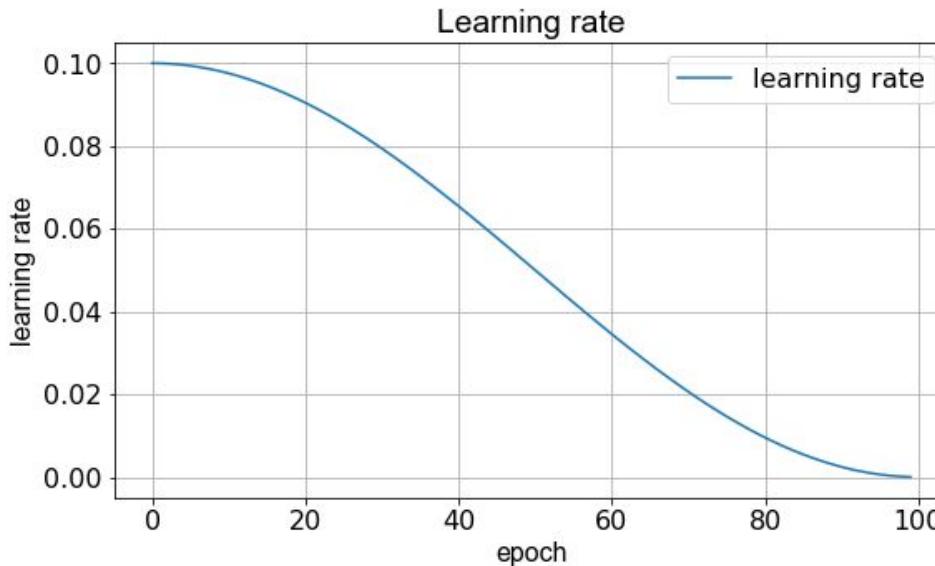




Learning rate schedulers

Cosine decay

$$lr(\text{epoch}) = \frac{lr_0}{2} \cdot \left(1 + \cos\left(\frac{\text{epoch} \cdot \pi}{\text{max epoch}}\right)\right)$$



Learning rate schedulers



Be careful with them:

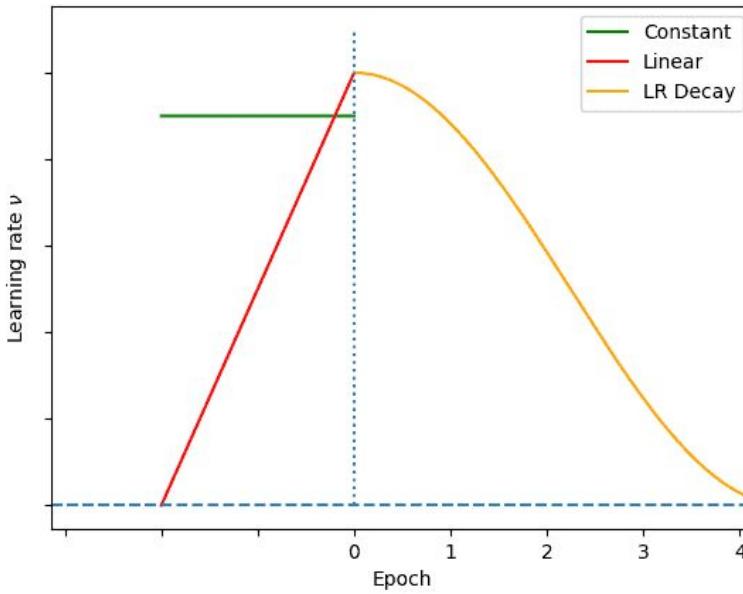
- Start with constant LR
- See how model training performs
- Begin adjusting LR only if there is no other way of improving model performance left to try
- Or apply Reduce LR on Plateau



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>





Weights initialization

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

$$\text{Var}(s) = \text{Var}\left(\sum_i^n w_i x_i\right)$$

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

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

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

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

Pytorch module

[https://pytorch.org/docs/stable/
nn.init.html](https://pytorch.org/docs/stable/nn.init.html)



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

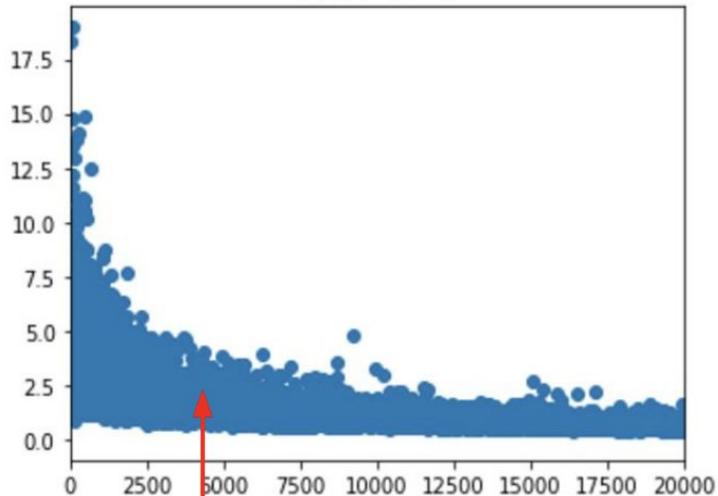
Normalization

girafe
ai

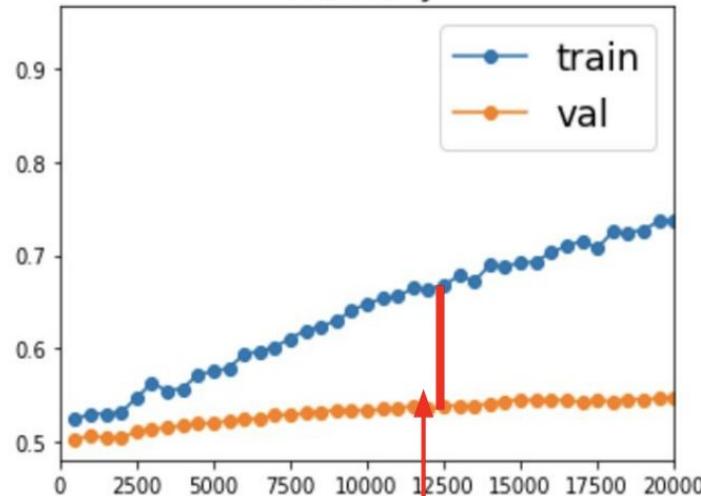
04



Train Loss



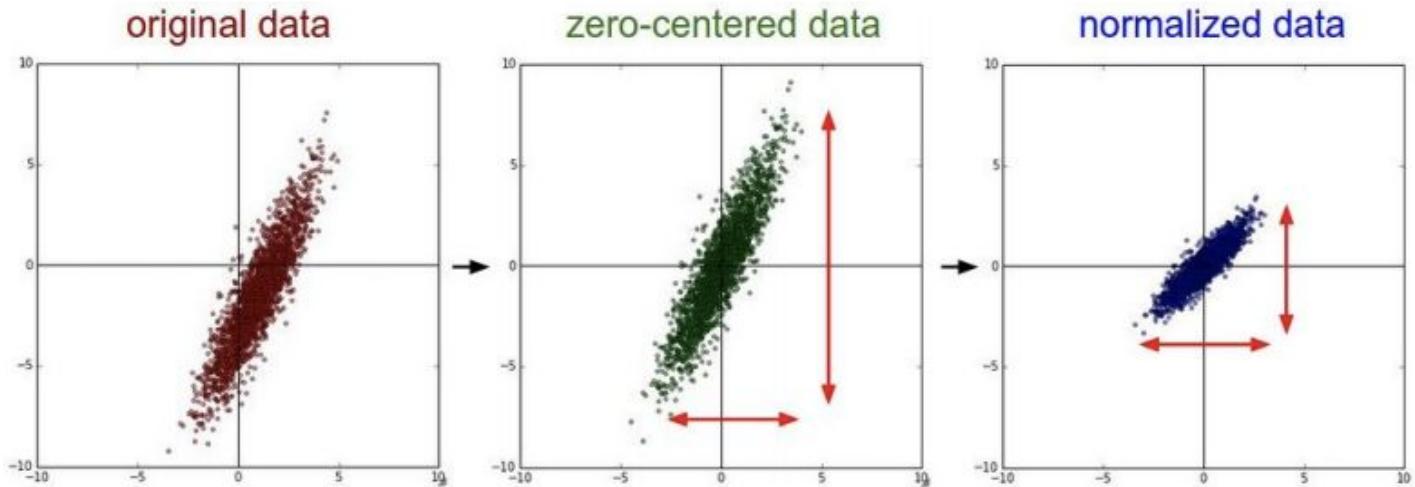
Accuracy



Better optimization algorithms help reduce training loss

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

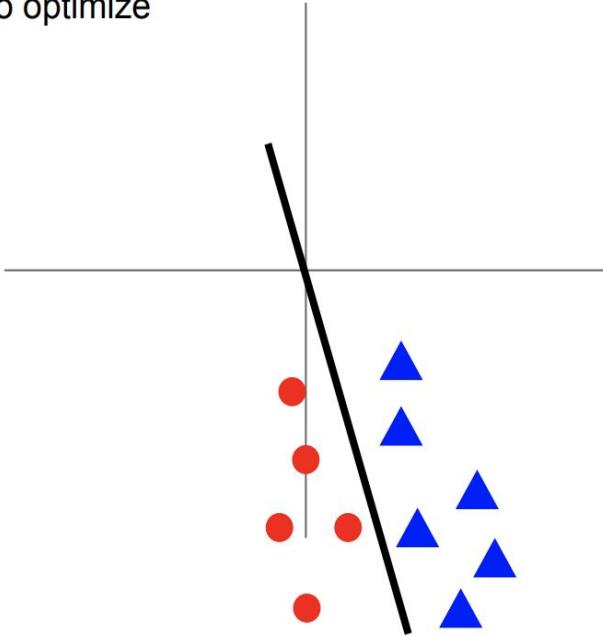
Data normalization



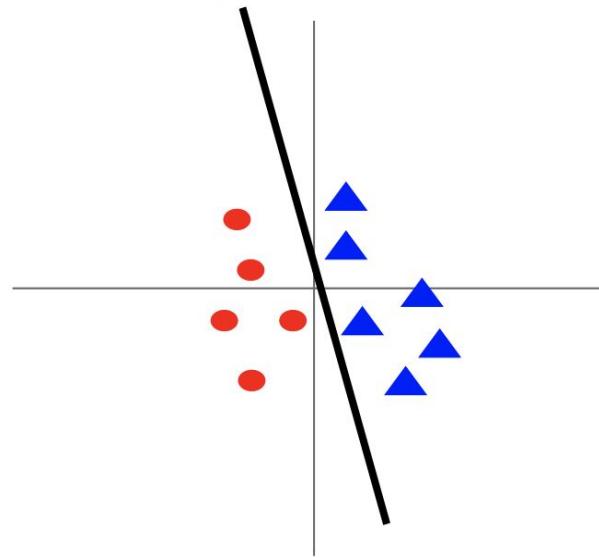


Data normalization

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

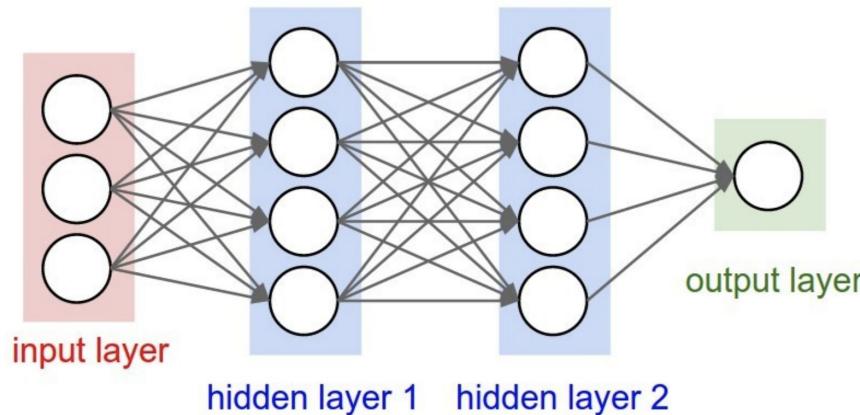




Internal covariate shift

- 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



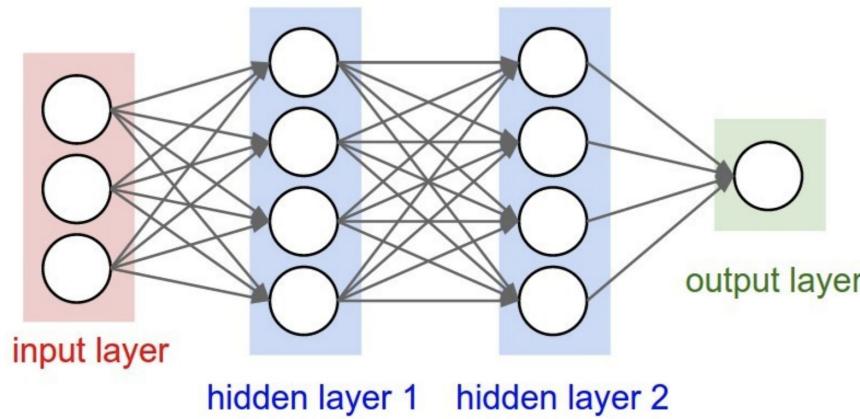


Batch normalization

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

$$\mu_i := \alpha \cdot \text{mean}_{batch} + (1 - \alpha) \cdot \mu_i$$

$$\sigma_i^2 := \alpha \cdot \text{variance}_{batch} + (1 - \alpha) \cdot \sigma_i^2$$



Batch normalization

Original algorithm (2015)

What is this?

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

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

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

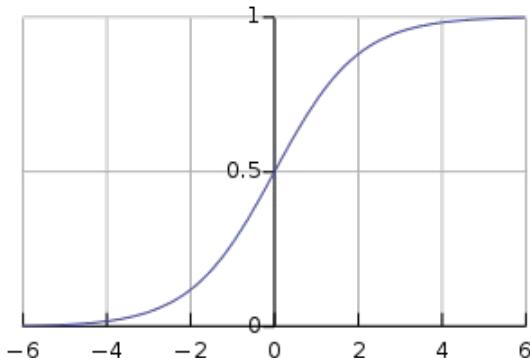
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

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

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



Batch normalization



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

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

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

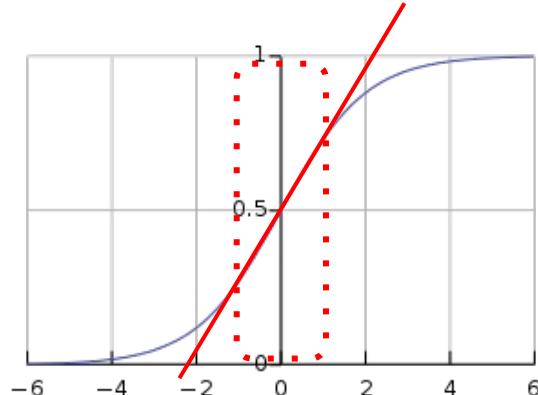
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

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

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



Batch normalization



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

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

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

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

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

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



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: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

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

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

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

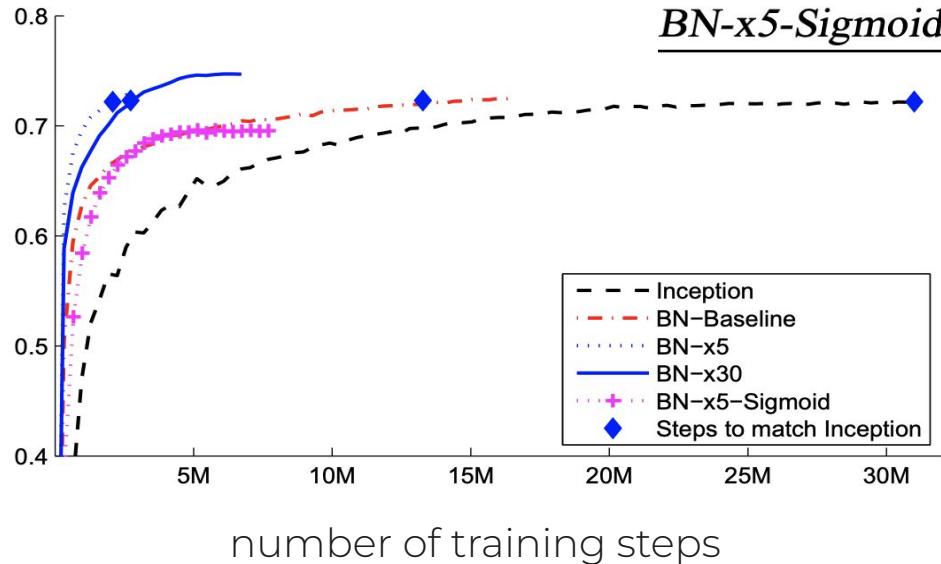
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

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



Training acceleration

accuracy





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`

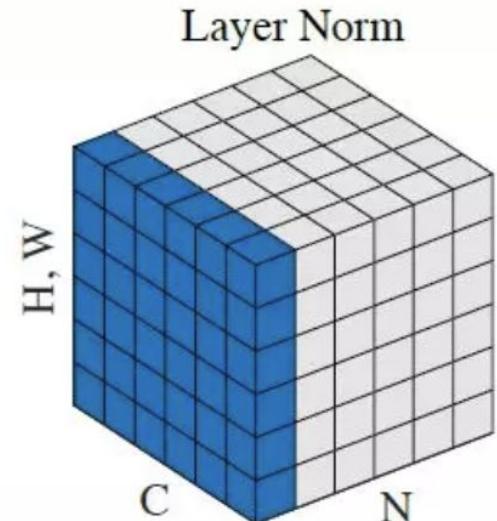
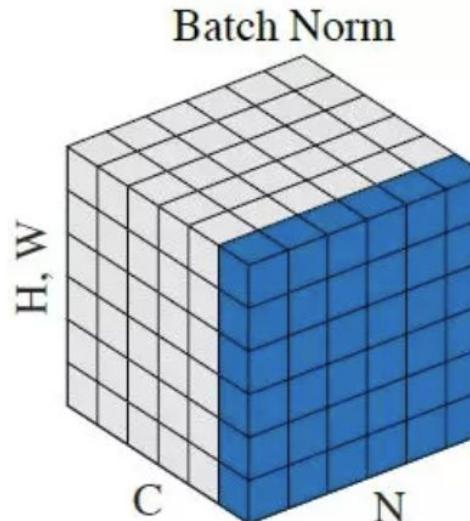
Layer normalization



Same idea, but applied object-wise

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

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



Regularization

girafe
ai

05

What is regularization anyway?



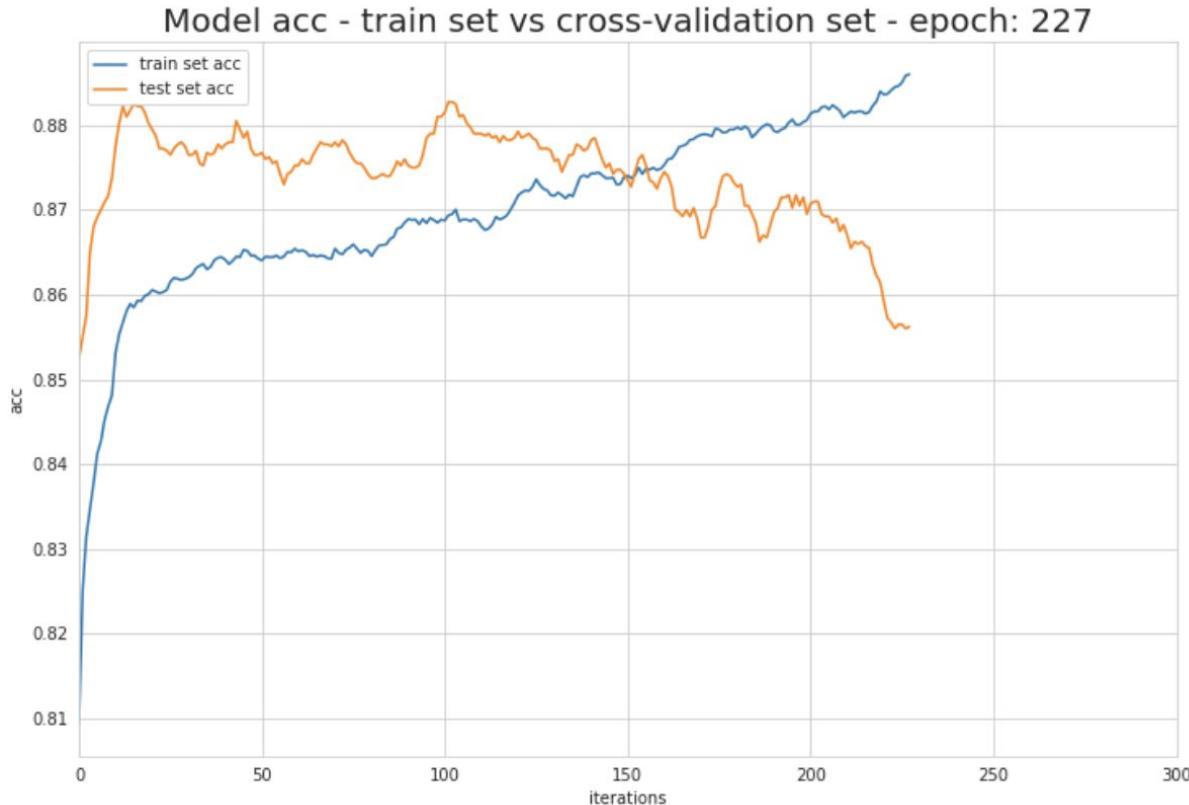
Regularization is a process that changes the result answer to be "simpler". It is often used to obtain results for ill-posed problems or to prevent overfitting.

(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:

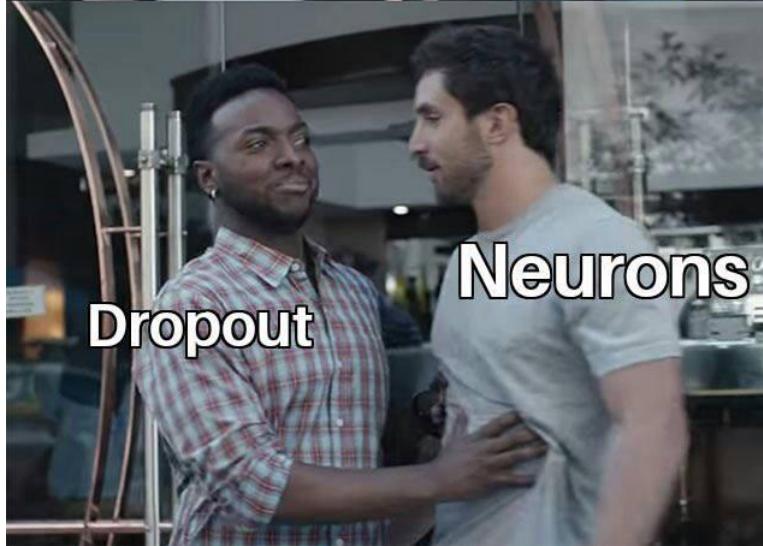
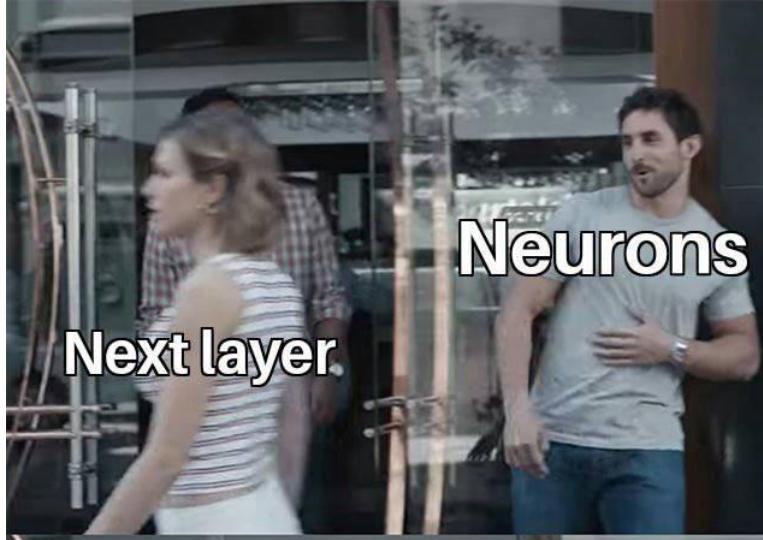
- L2 regularization:
- L1 regularization:
- Elastic Net (L1 + L2):

$$R(W) = \|W\|_2^2$$

$$R(W) = \|W\|_1$$

$$R(W) = \beta \|W\|_2^2 + \|W\|_1$$

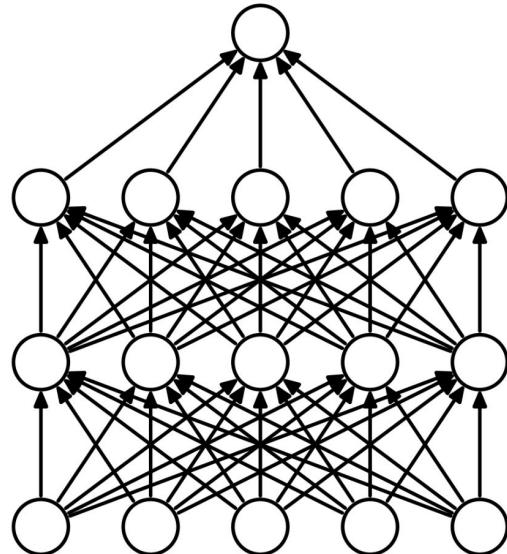
Dropout



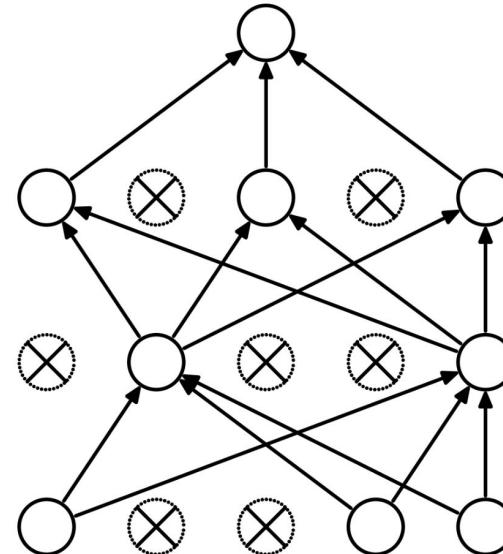
Dropout



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

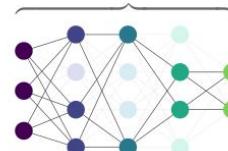
Dropout



Some neurons are “dropped” during training.

Prevents overfitting.

The network with dropout during a single forward pass



Nodes set to zero during forward passes

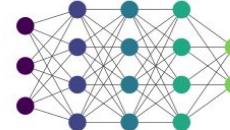


Dropout

Network regularization

For each forward pass during training, set the output of each node to zero with probability P .

For testing and inference use the entire network



Dropout is the equivalent of training several independent, smaller networks on the same task. The final model is like an ensemble of smaller networks, reducing variance and providing more robust predictions.

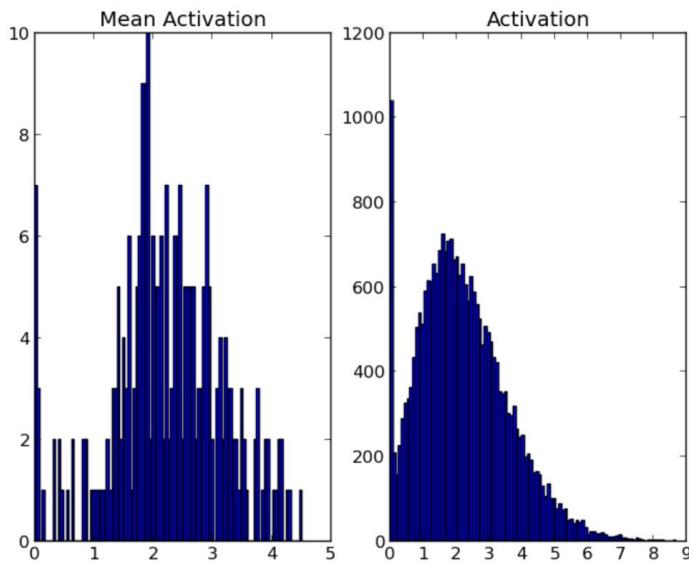
Actually, on test case output should be normalized. See sources for more info.

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

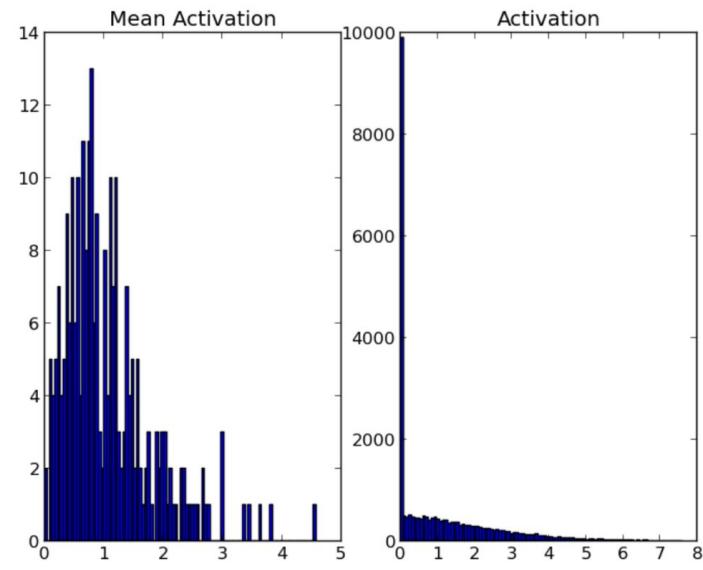


Effect on weights sparsity

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.

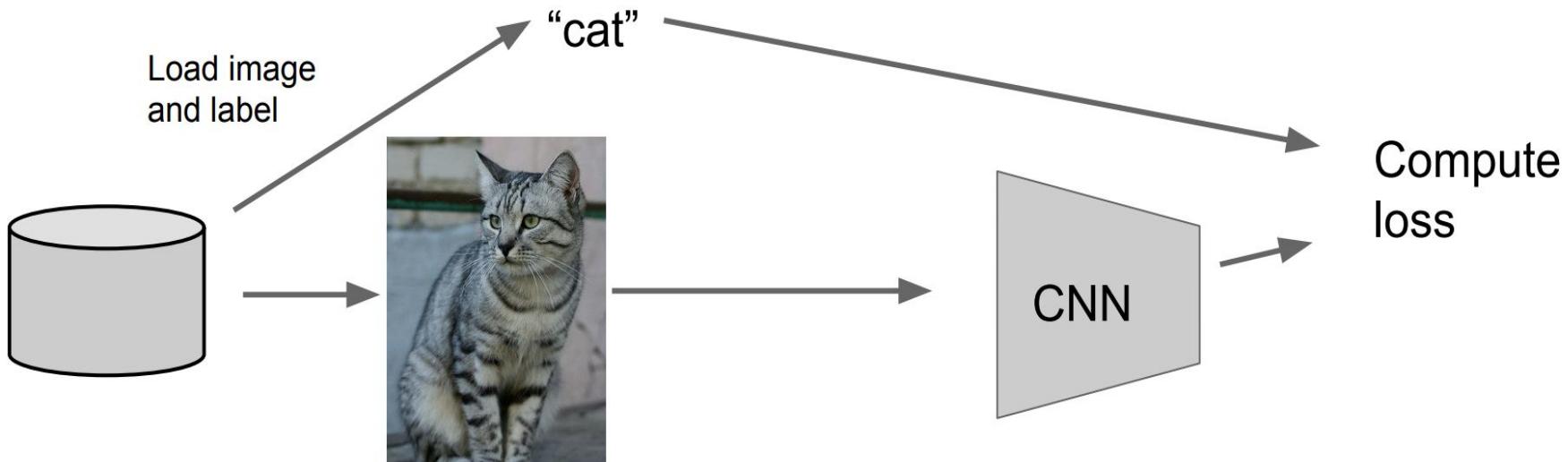
Usually not there are 1 to 3 of them in architecture.

Augmentation

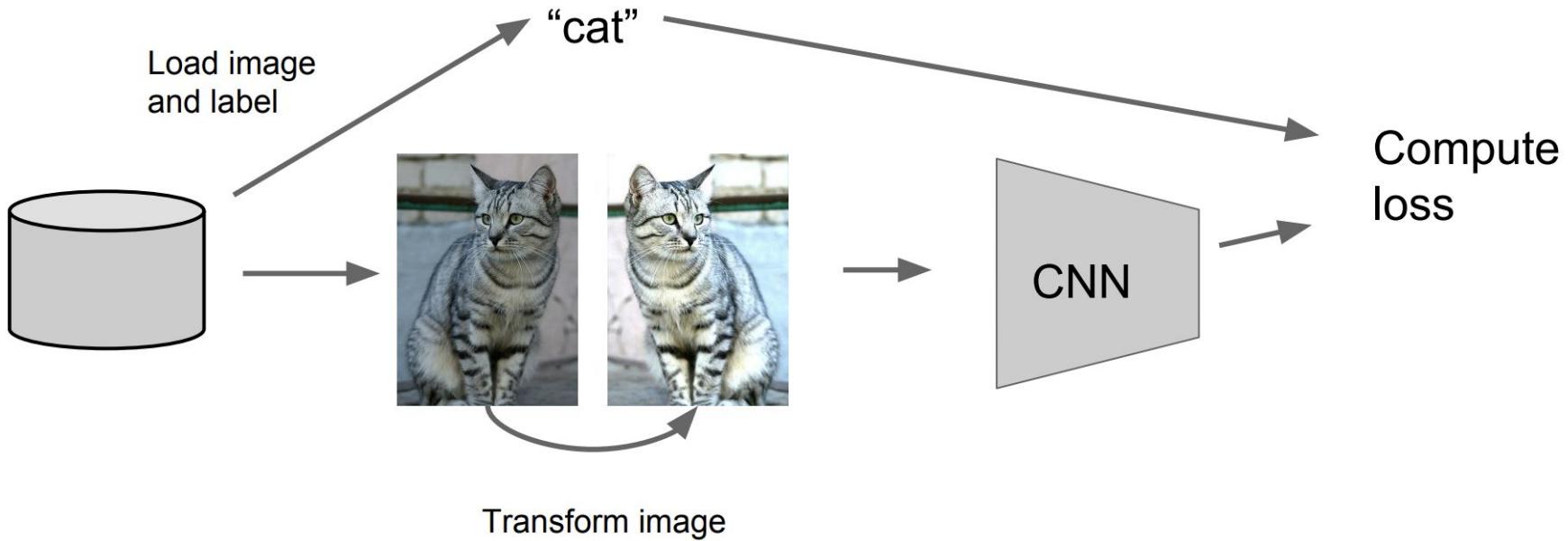
girafe
ai

06

Regular data flow



Data augmentation





Many ways to augment

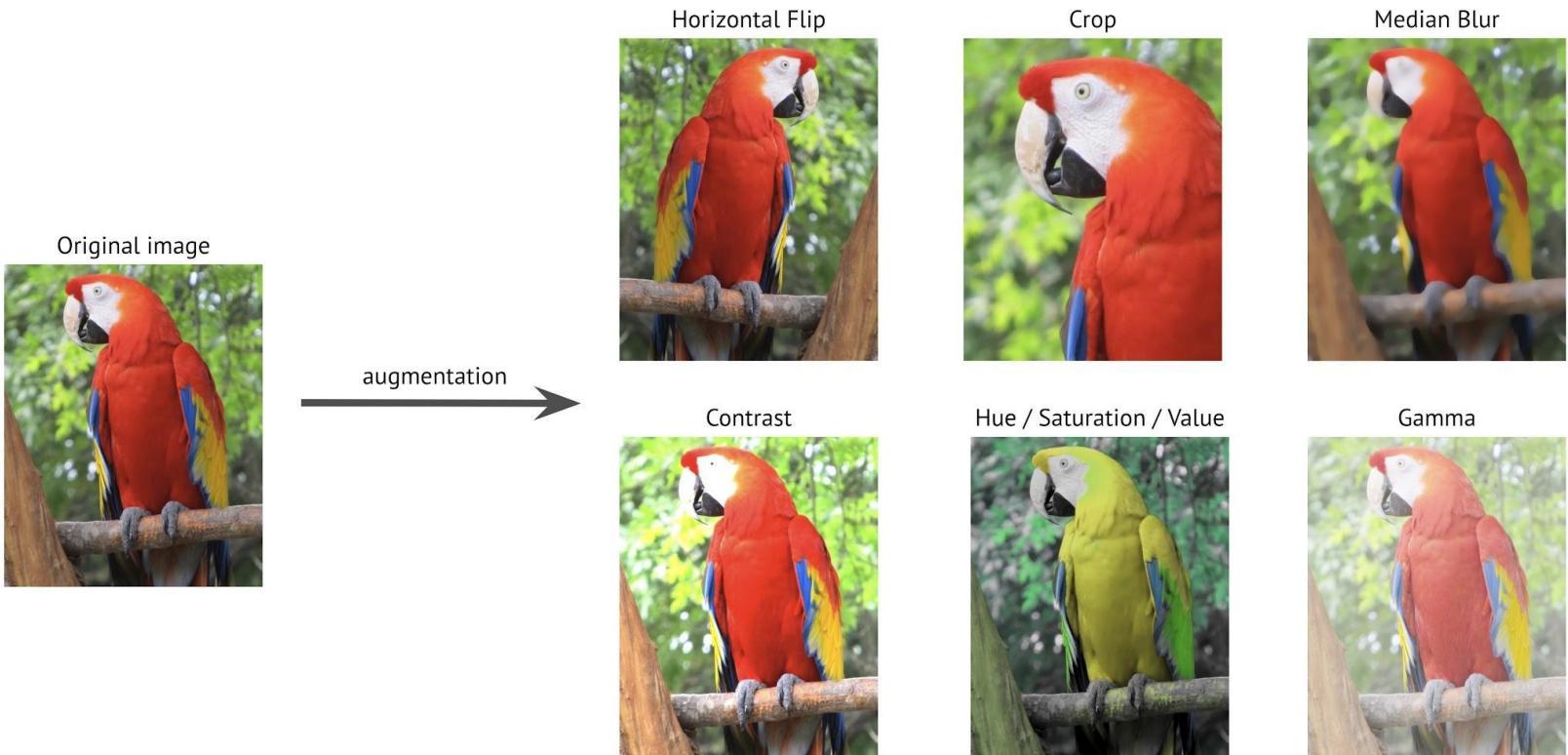




Image augmentations

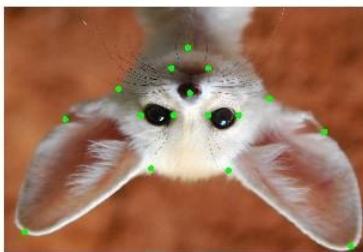
Original



HorizontalFlip



VerticalFlip



ShiftScaleRotate



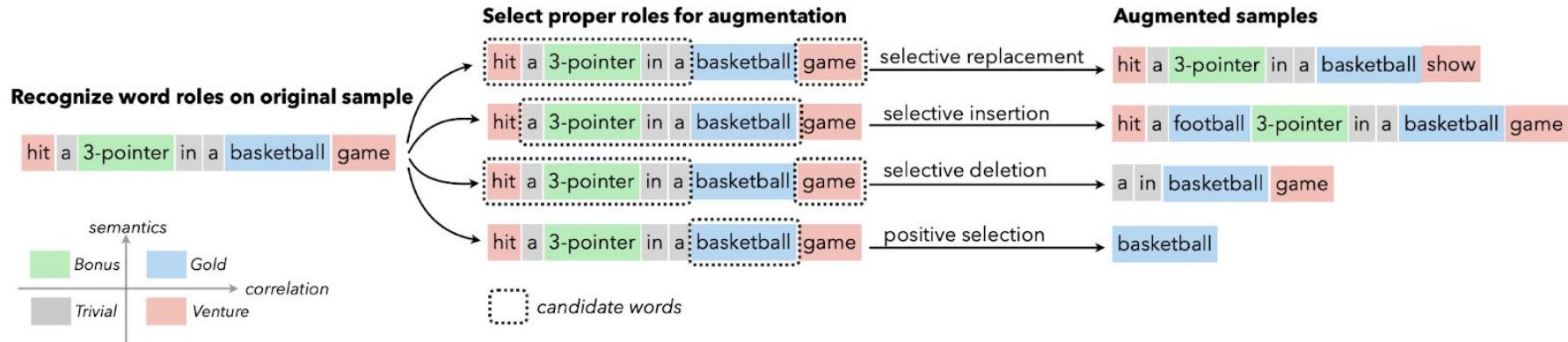
- <https://pytorch.org/vision/stable/transforms.html>
- <https://albumentations.ai/>
- <https://augmentor.readthedocs.io/en/stable/>
- <https://imgaug.readthedocs.io/en/stable/>

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/hlpaug/>
- <https://github.com/QData/TextAttack>

Revise

A decorative graphic in the bottom-left corner consists of several white-outlined geometric shapes on a teal background. It includes a large hexagon on the left, a smaller pentagon in the center, and some irregular, wavy lines at the bottom.

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

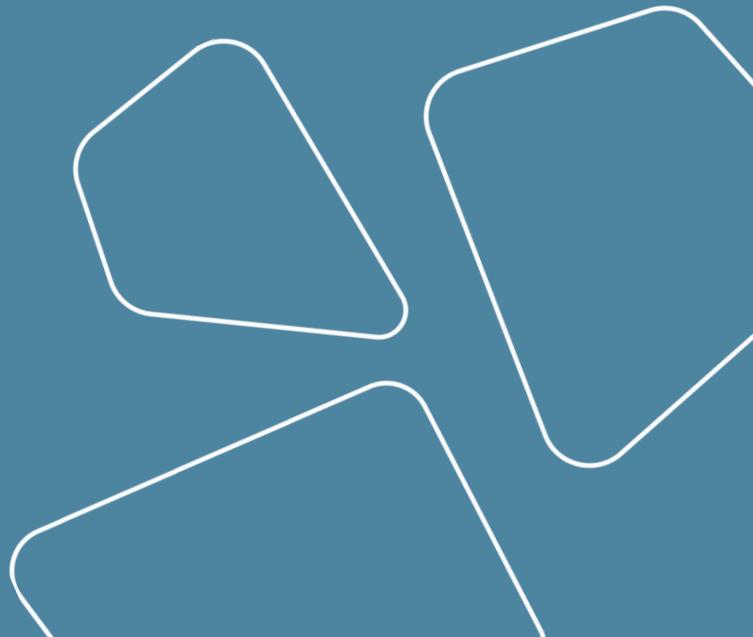
- https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Thanks for attention!

Questions?



girafe
ai



Extra

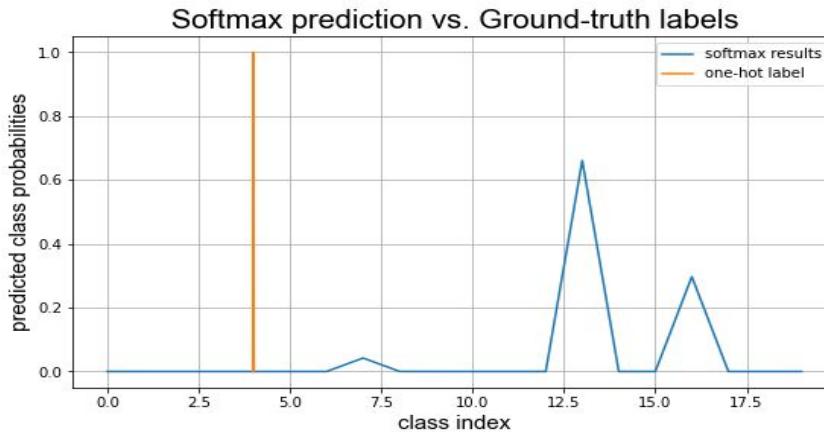
girafe
ai

Label Smoothing Regularization

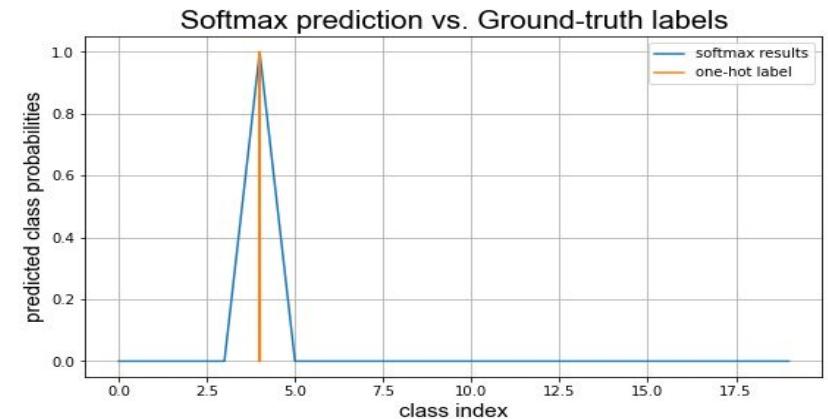


Motivation (for classification):

- cross-entropy loss maximizes log-likelihood between one-hot targets and model predictions
- hard (one-hot) targets → model predicts large logits
- model is overconfident in predictions → may lead to overfitting



(a) no smoothing



(b) apply smoothing source



Label Smoothing Regularization

Motivation (for classification):

- Interpretation in terms of minimized cost function: minimize standard CE with new term

$$H(q', p) = - \sum_{k=1}^K \log p(k) q'(k) = \text{standard CE} + \epsilon \text{regularization term}$$
$$(1-\epsilon)H(q, p) + \epsilon H(u, p)$$

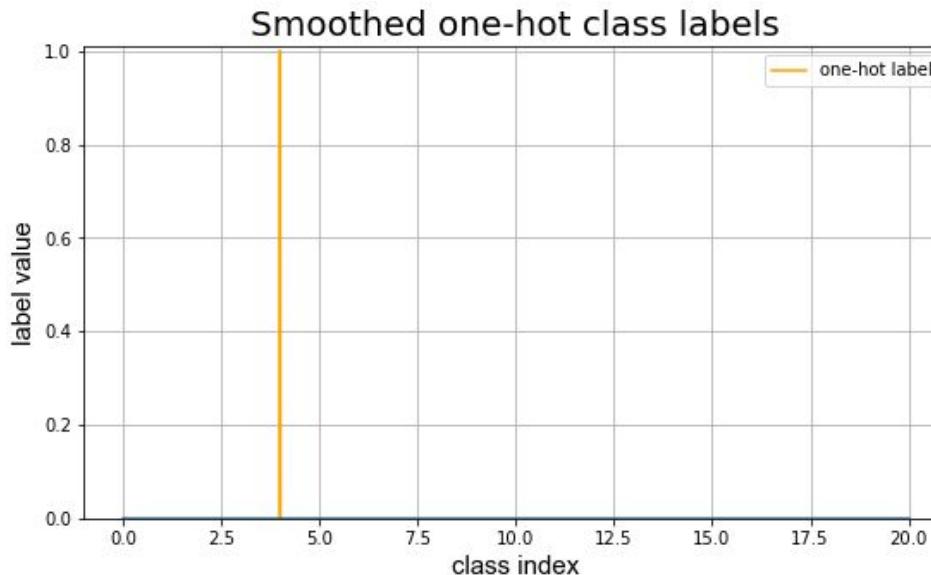
Thus, LSR is equivalent to replacing a single cross-entropy loss $H(q, p)$ with a pair of such losses $H(q, p)$ and $H(u, p)$.



Label Smoothing Regularization

- hard one-hot targets:

$$y_k = \begin{cases} 1, & \text{if } k - \text{ground-truth class, } k \in \{1, \dots, K\} \\ 0, & \text{otherwise} \end{cases}$$

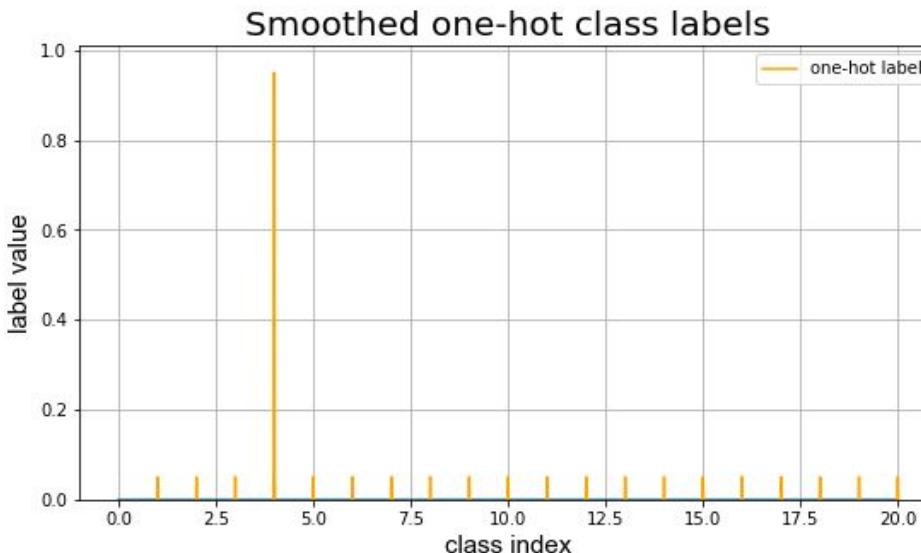




Label Smoothing Regularization

- smooth one-hot targets: add a smoothing term α : $0 < \alpha \ll 1$

$$y_k = \begin{cases} 1 - \alpha, & \text{if } k - \text{ground-truth class, } k \in \{1, \dots, K\} \\ \frac{\alpha}{K}, & \text{otherwise} \end{cases}$$





Label Smoothing Regularization

- smooth one-hot targets: add a smoothing term α : $0 < \alpha \ll 1$

$$y_k = \begin{cases} 1 - \alpha, & \text{if } k - \text{ground-truth class, } k \in \{1, \dots, K\} \\ \frac{\alpha}{K}, & \text{otherwise} \end{cases}$$

- encourages the difference between the logit of the correct class and the logits of incorrect classes to be a constant dependent on α
- encourages the activations of the penultimate layer to be close to the template of the correct class and equally distant to the templates of the incorrect classes.



Label Smoothing: Results

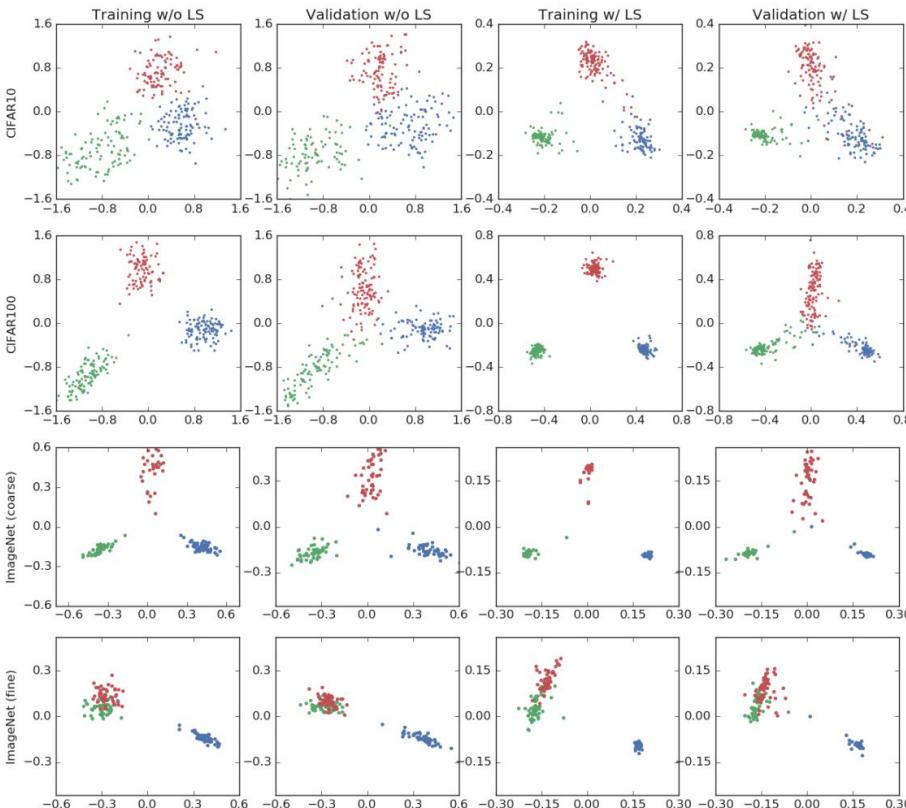


Figure 1: Visualization of penultimate layer's activations of: AlexNet/CIFAR-10 (first row), CIFAR-100/ResNet-56 (second row) and ImageNet/Inception-v4 with three semantically different classes (third row) and two semantically similar classes plus a third one (fourth row).



Mixup

- Idea: mix features and labels with linear interpolation

Contribution Motivated by these issues, we introduce a simple and data-agnostic data augmentation routine, termed *mixup* (Section 2). In a nutshell, *mixup* constructs virtual training examples

$$\tilde{x} = \lambda x_i + (1 - \lambda)x_j, \quad \text{where } x_i, x_j \text{ are raw input vectors}$$

$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j, \quad \text{where } y_i, y_j \text{ are one-hot label encodings}$$

(x_i, y_i) and (x_j, y_j) are two examples drawn at random from our training data, and $\lambda \in [0, 1]$. Therefore, *mixup* extends the training distribution by incorporating the prior knowledge that linear interpolations of feature vectors should lead to linear interpolations of the associated targets. *mixup* can be implemented in a few lines of code, and introduces minimal computation overhead.

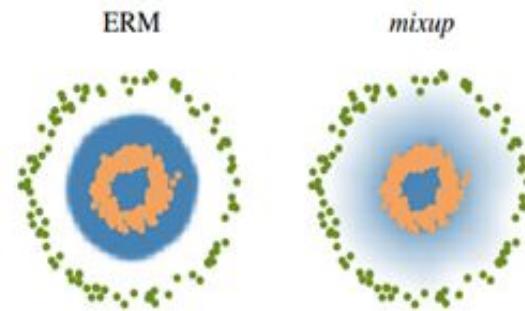


Mixup

- Extremely strong regularizer
- A form data augmentation,
- According to authors learned manifold is smoother
- (see toy model example)

```
# y1, y2 should be one-hot vectors
for (x1, y1), (x2, y2) in zip(loader1, loader2):
    lam = numpy.random.beta(alpha, alpha)
    x = Variable(lam * x1 + (1. - lam) * x2)
    y = Variable(lam * y1 + (1. - lam) * y2)
    optimizer.zero_grad()
    loss(net(x), y).backward()
    optimizer.step()
```

(a) One epoch of *mixup* training in PyTorch.



(b) Effect of *mixup* ($\alpha = 1$) on a toy problem. Green: Class 0. Orange: Class 1. Blue shading indicates $p(y = 1|x)$.



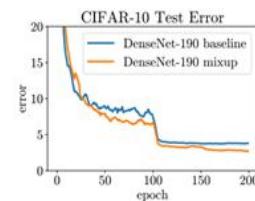
Mixup: Results

Model	Method	Epochs	Top-1 Error	Top-5 Error
ResNet-50	ERM (Goyal et al., 2017)	90	23.5	-
	<i>mixup</i> $\alpha = 0.2$	90	23.3	6.6
ResNet-101	ERM (Goyal et al., 2017)	90	22.1	-
	<i>mixup</i> $\alpha = 0.2$	90	21.5	5.6
ResNeXt-101 32*4d	ERM (Xie et al., 2016)	100	21.2	-
	ERM	90	21.2	5.6
	<i>mixup</i> $\alpha = 0.4$	90	20.7	5.3
ResNeXt-101 64*4d	ERM (Xie et al., 2016)	100	20.4	5.3
	<i>mixup</i> $\alpha = 0.4$	90	19.8	4.9
ResNet-50	ERM	200	23.6	7.0
	<i>mixup</i> $\alpha = 0.2$	200	22.1	6.1
ResNet-101	ERM	200	22.0	6.1
	<i>mixup</i> $\alpha = 0.2$	200	20.8	5.4
ResNeXt-101 32*4d	ERM	200	21.3	5.9
	<i>mixup</i> $\alpha = 0.4$	200	20.1	5.0

Table 1: Validation errors for ERM and *mixup* on the development set of ImageNet-2012.

Dataset	Model	ERM	<i>mixup</i>
CIFAR-10	PreAct ResNet-18	5.6	4.2
	WideResNet-28-10	3.8	2.7
	DenseNet-BC-190	3.7	2.7
CIFAR-100	PreAct ResNet-18	25.6	21.1
	WideResNet-28-10	19.4	17.5
	DenseNet-BC-190	19.0	16.8

(a) Test errors for the CIFAR experiments.



(b) Test error evolution for the best ERM and *mixup* models.

Figure 3: Test errors for ERM and *mixup* on the CIFAR experiments.