ECE 884 Deep Learning

Lecture 12: Training Neural Networks

03/04/2021

Review of last lecture

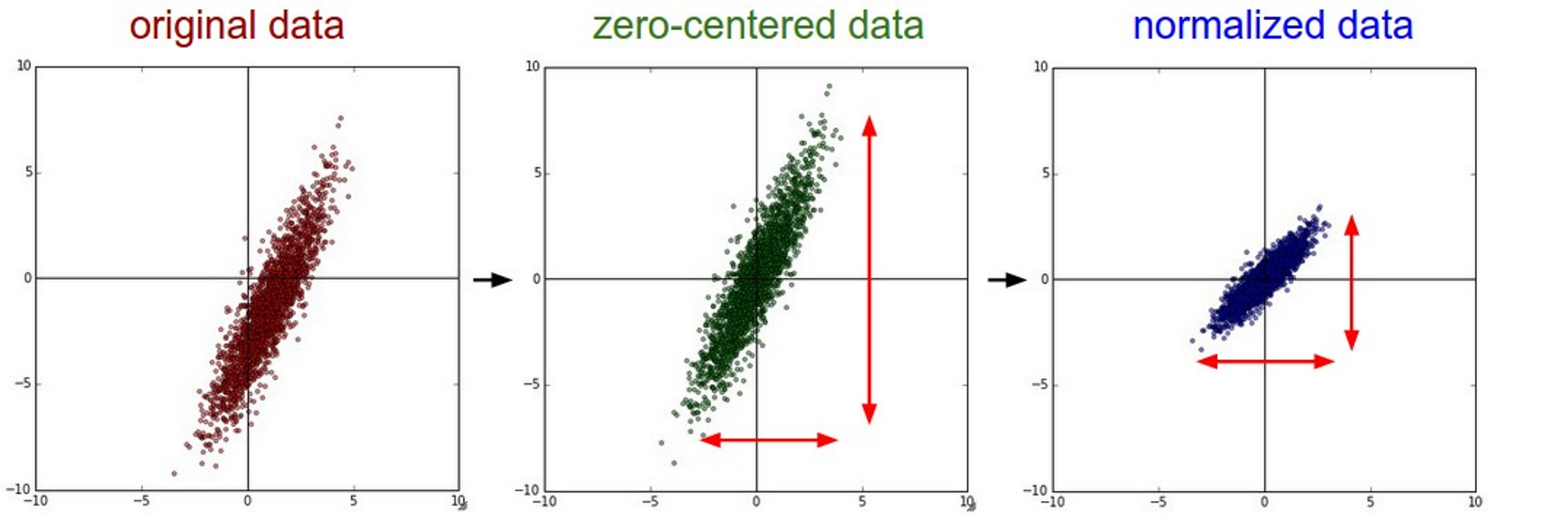
- Backward Propagation (Backpropagation)

Today’s lecture

- Deep Neural Network Training Pipeline
  - Data preprocessing
  - Regularization for Neural Networks
  - Learning Rate Schedules

Data Preprocessing

Data Preprocessing

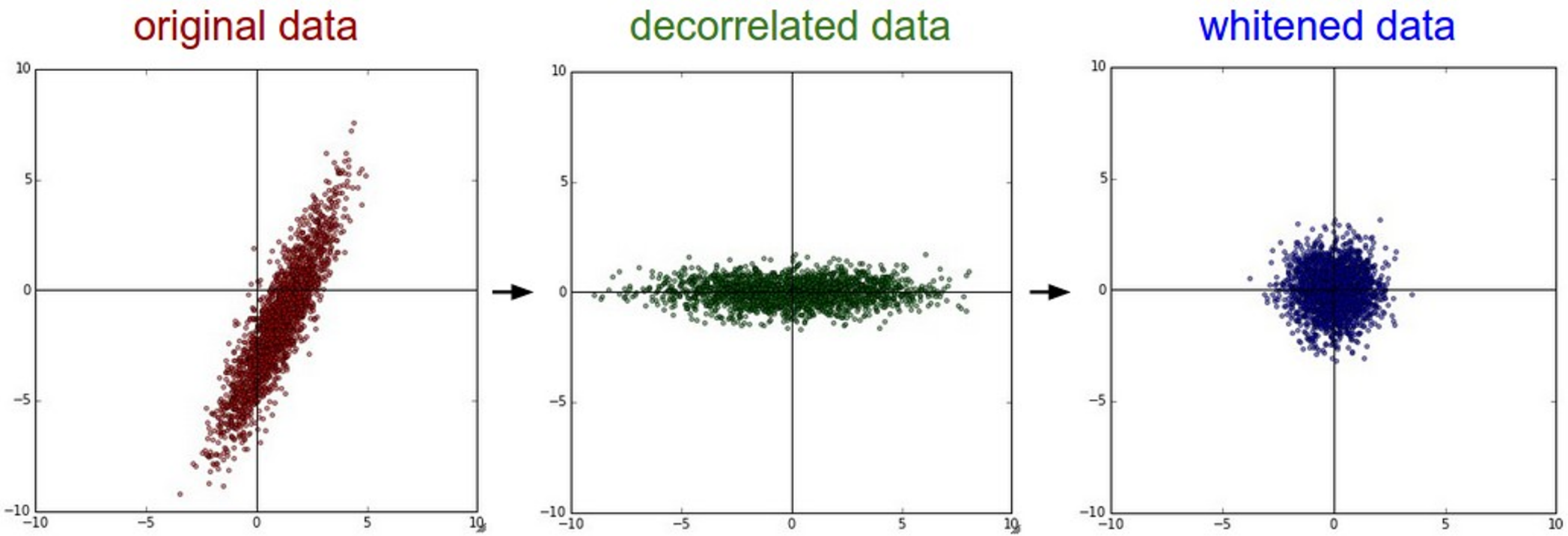



```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Data Preprocessing

In practice, you may also see **PCA** and **Whitening** of the data

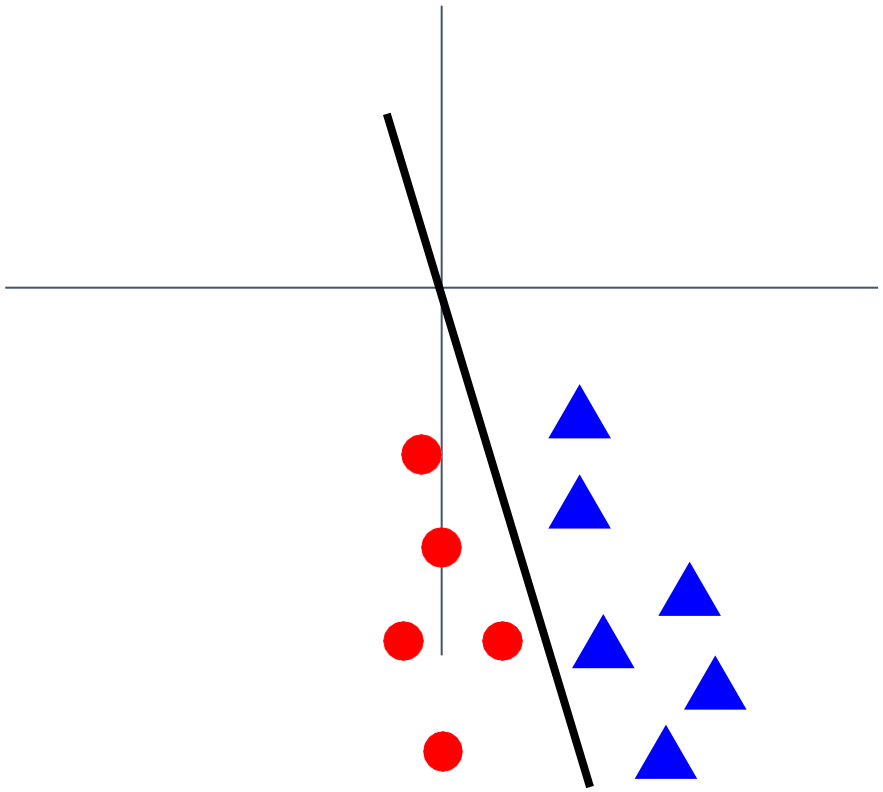


(data has diagonal covariance matrix)

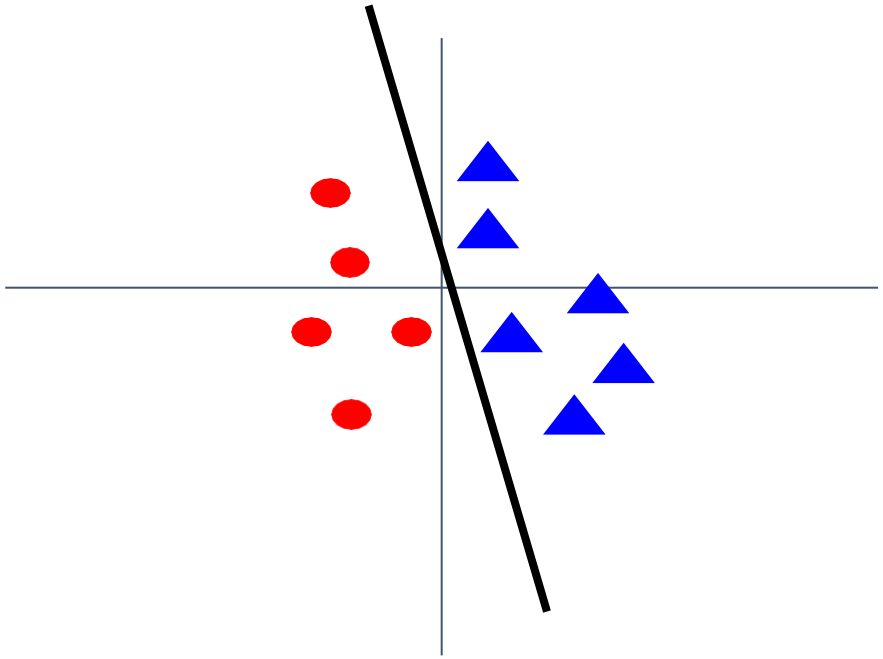
(covariance matrix is the identity matrix)

Data Preprocessing

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

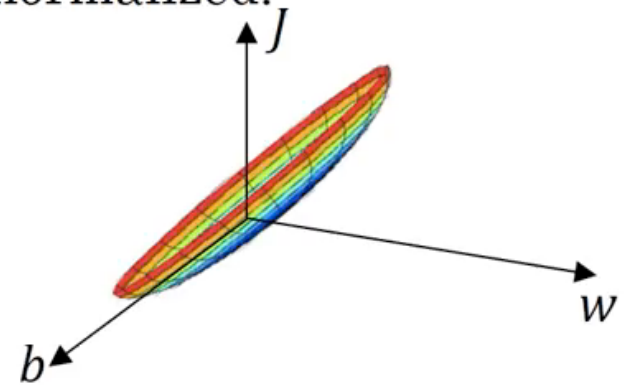


After normalization: less sensitive to small changes in weights; easier to optimize

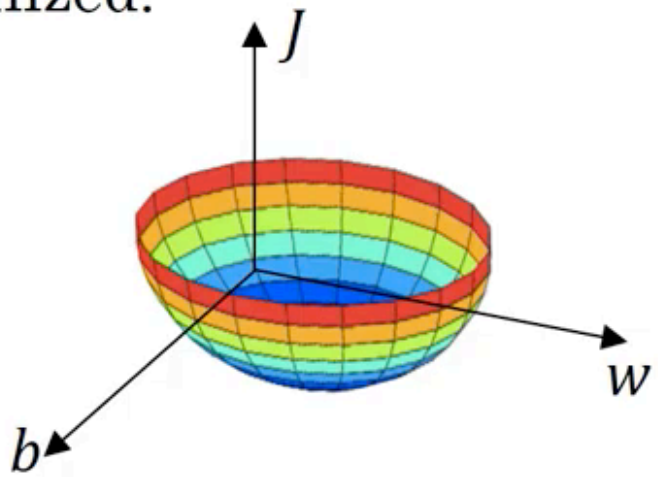


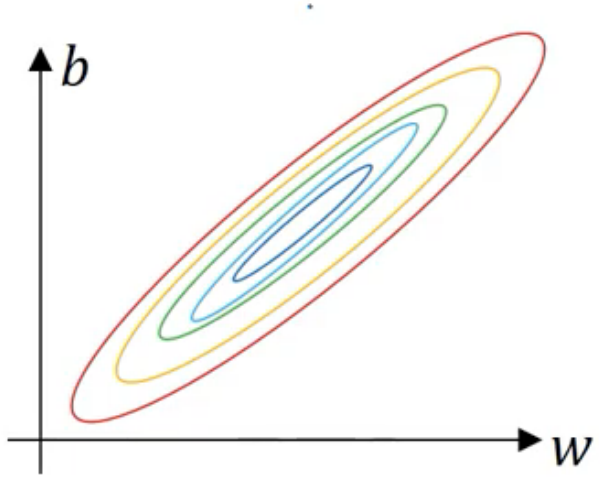
Data Preprocessing

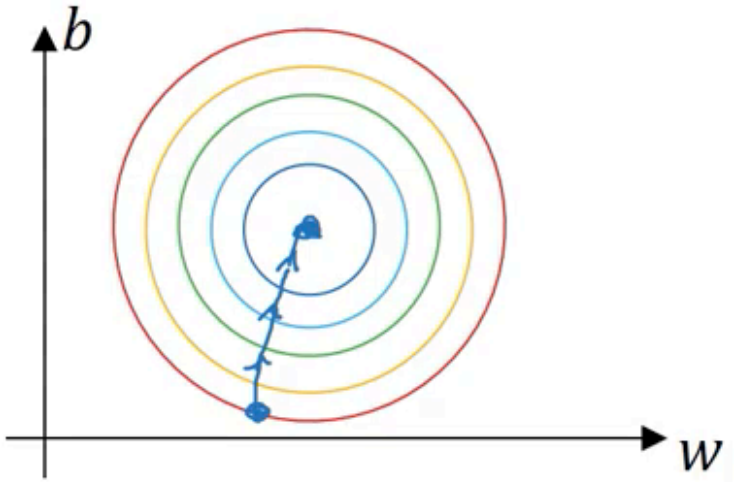
Unnormalized:



Normalized:

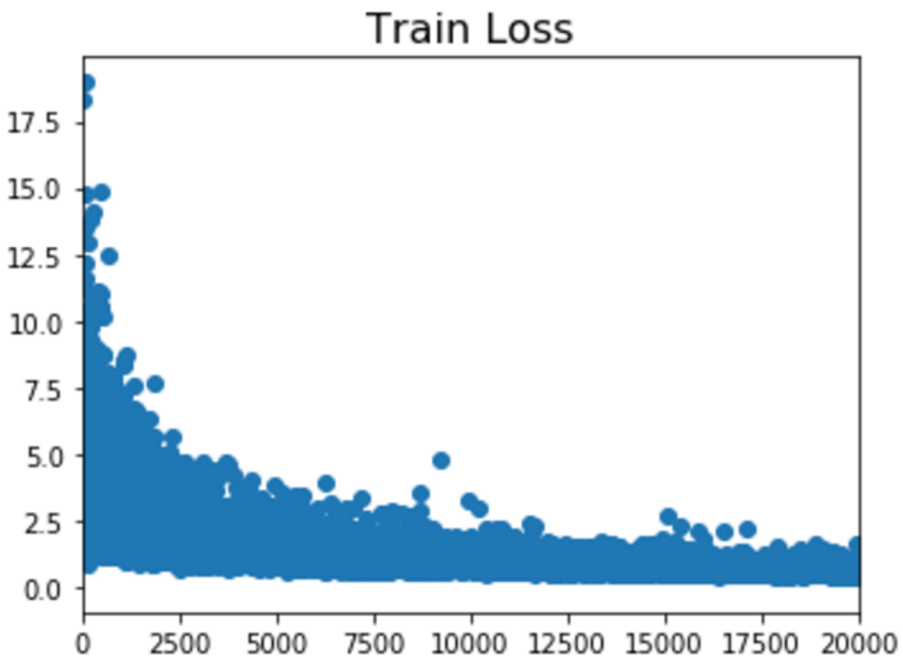


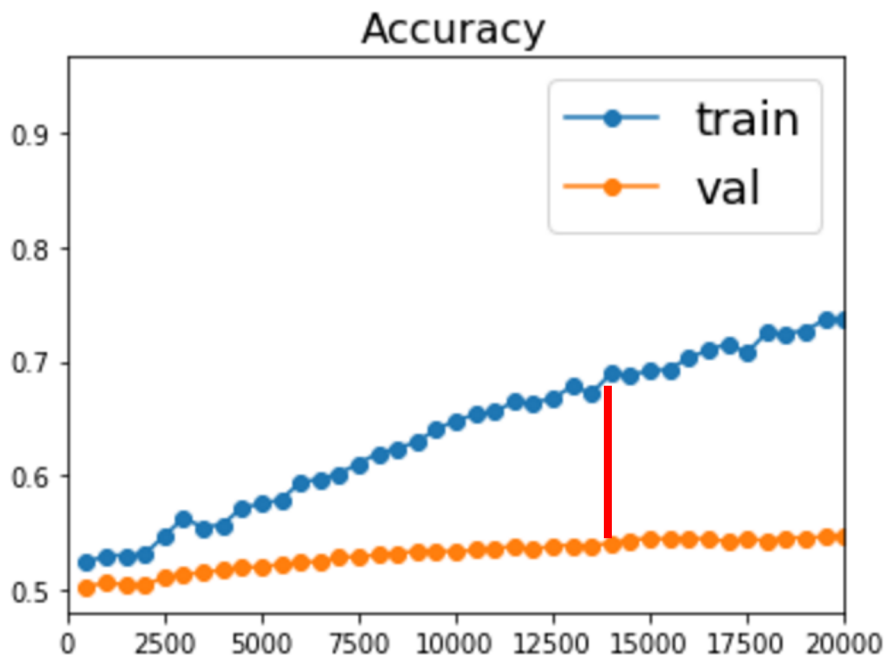




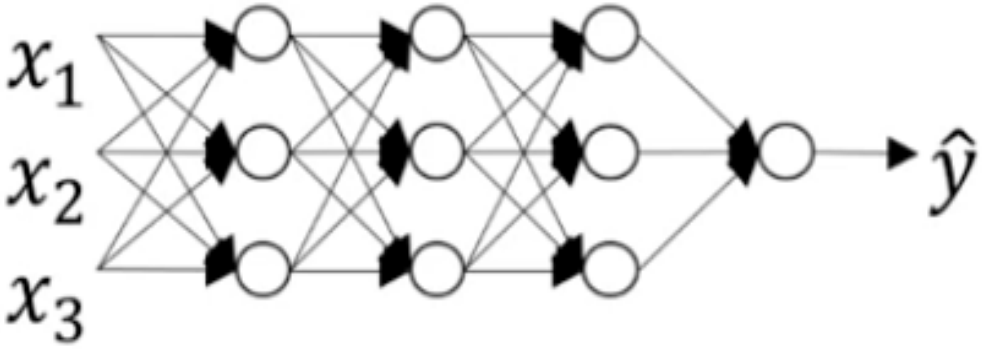
Regularization for Neural Networks

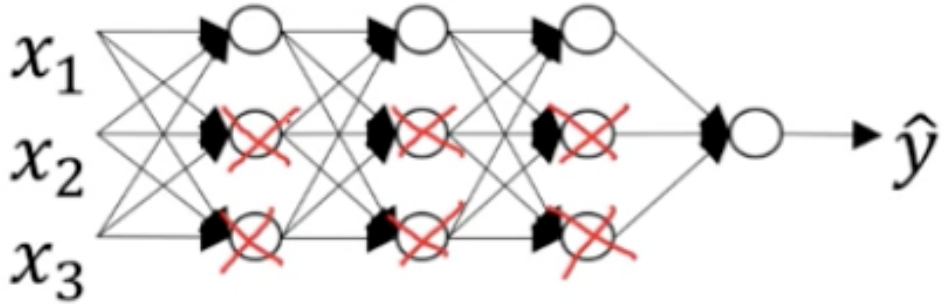
Regularization for Neural Networks





Regularization: Penalize Neural Network Complexity





Regularization#1: Add term to the loss

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

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

L2 Regularization for DNN

Regularized objective:

$$\hat{L}(w) = \frac{\lambda}{2}\|w\|_2^2 + \sum_{i=1}^{n} l(w, x_i, y_i)$$

Gradient of objective:

$$\nabla\hat{L}(w) = \lambda w + \sum_{i=1}^{n} \nabla l(w, x_i, y_i)$$

SGD update:

$$w \leftarrow w - \eta\big(\lambda w + \nabla l(w, x_i, y_i)\big)$$
$$w \leftarrow (1 - \eta\lambda)w - \eta\nabla l(w, x_i, y_i)$$

Interpretation: weight decay

L1 Regularization for DNN

Regularized objective:

$$\hat{L}(w) = \lambda \|w\|_1 + \sum_{i=1}^{n} l(w, x_i, y_i)$$

$$= \lambda \sum_{d} |w_d| + \sum_{i=1}^{n} l(w, x_i, y_i)$$

Gradient: $\nabla \hat{L}(w) = \lambda \operatorname{sgn}(w) + \sum_{i=1}^{n} \nabla l(w, x_i, y_i)$

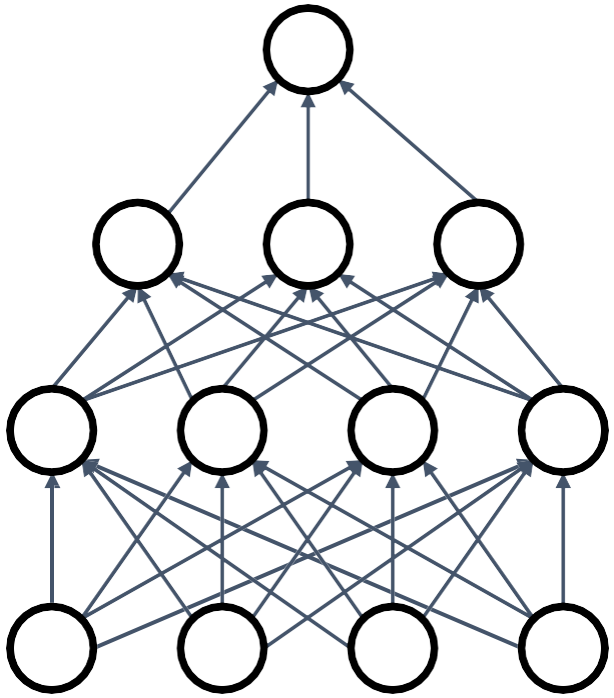
SGD update:

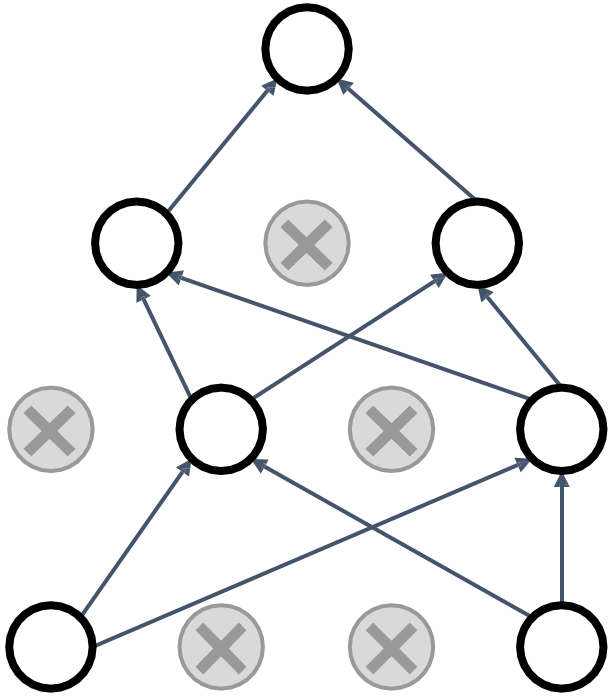
$$w \leftarrow w - \eta\lambda \operatorname{sgn}(w) - \eta \nabla l(w, x_i, y_i)$$

Interpretation: encouraging sparsity

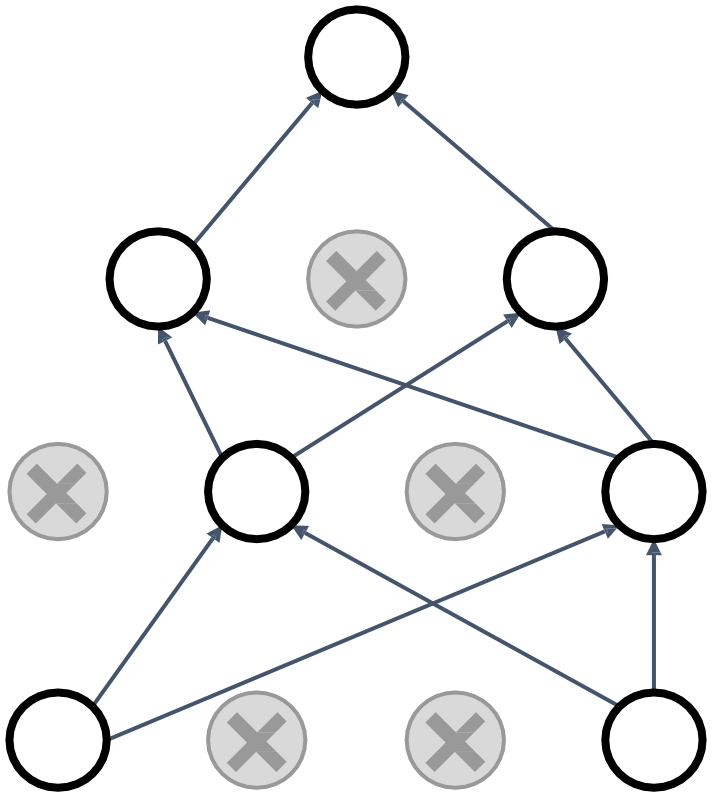
Regularization#2: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common





Dropout Interpretations



Interpretations:

#1: Can't reply on any one feature. Spread out the weights, similar to L2 regularization.

#2: Dropout is training a large **ensemble** of models (that share parameters).

Dropout: Test Time

Output (label) Input (image)

Dropout makes our output random! $\boldsymbol{y} = f_W(\boldsymbol{x}, \boldsymbol{z})$ Random mask

Want to "average out" the randomness at test-time

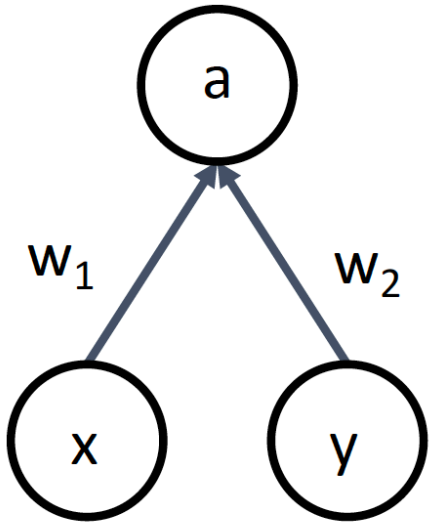
$$y = f(x) = E_z[f(x, z)] = \int p(z) f(x, z) dz$$

But this integral seems hard …

Dropout: Test Time

Want to approximate the integral

$$y = f(x) = E_z[f(x,z)] = \int p(z) f(x,z) dz$$



Consider a single neuron:

At test time we have: $E[a] = w_1 x + w_2 y$

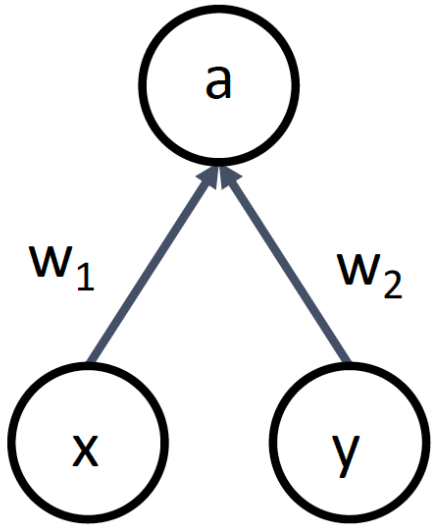
During training we have:

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ &= \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

Dropout: Test Time

Want to approximate the integral

$$y = f(x) = E_z[f(x,z)] = \int p(z) f(x,z) dz$$



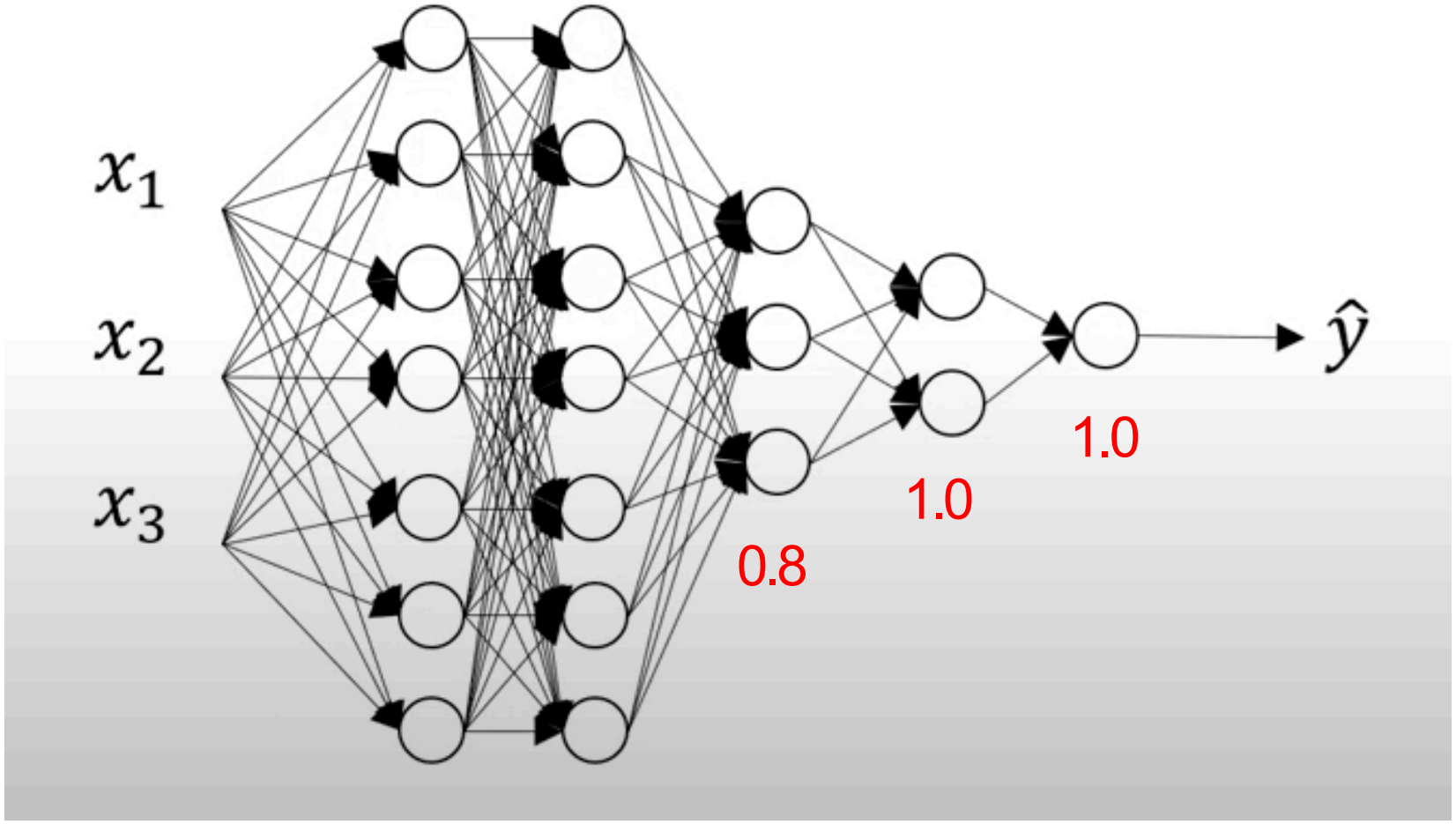
Consider a single neuron:

At test time we have: $E[a] = w_1 x + w_2 y$

During training we have: $E[a] = \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) = \frac{1}{2}(w_1 x + w_2 y)$

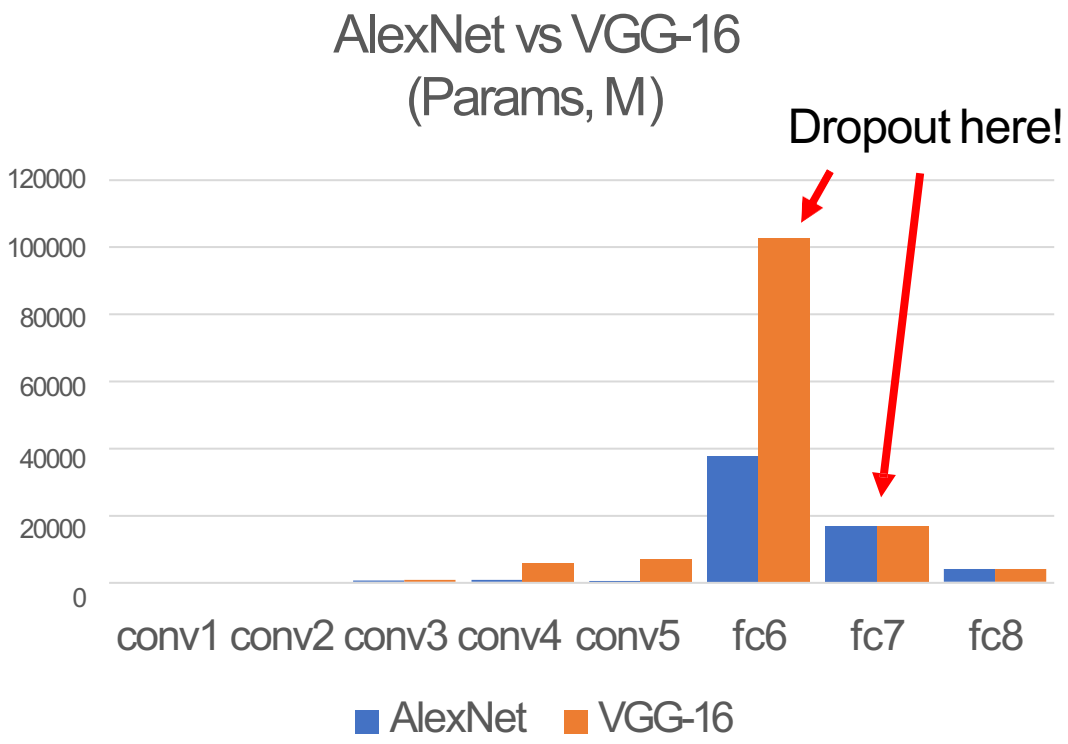
At test time, drop nothing and **multiply** by dropout probability

Where you do the Dropout



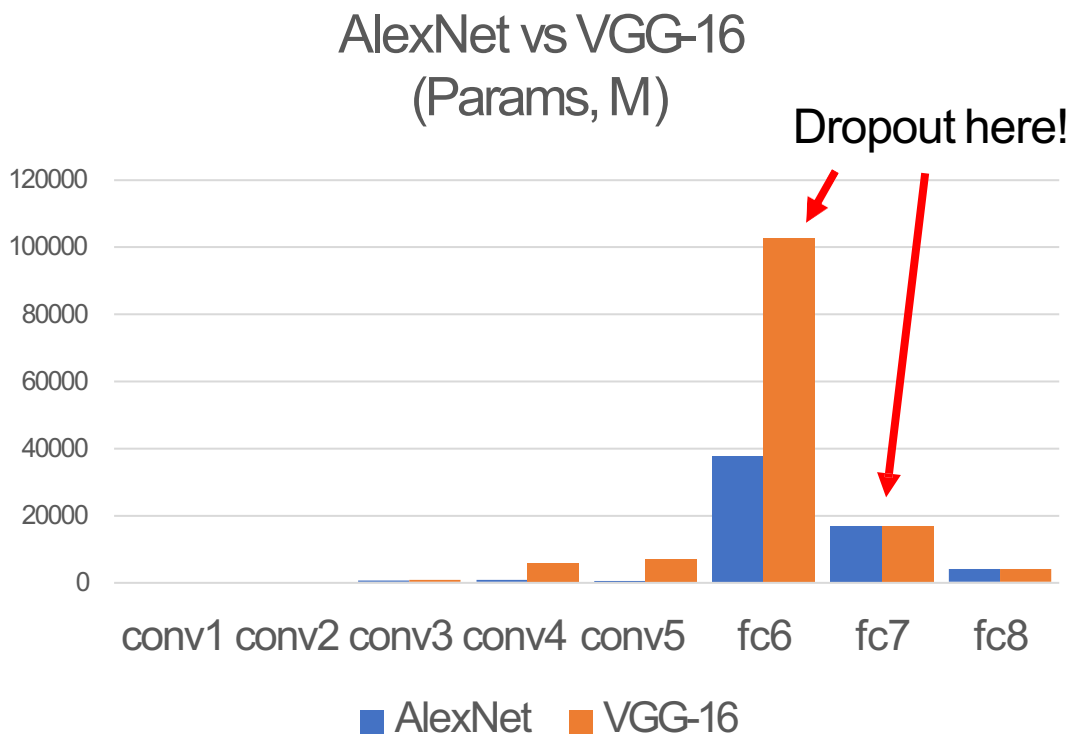
Where you do the Dropout

Recall AlexNet, VGG have most of their parameters in **fully-connected layers**; usually Dropout is applied there



Where you do the Dropout

Recall AlexNet, VGG have most of their parameters in **fully-connected layers**; usually Dropout is applied there



Later architectures (GoogLeNet, ResNet, etc) use global average pooling instead of fully-connected layers: they don't use dropout at all!

Regularization#3: Data Augmentation

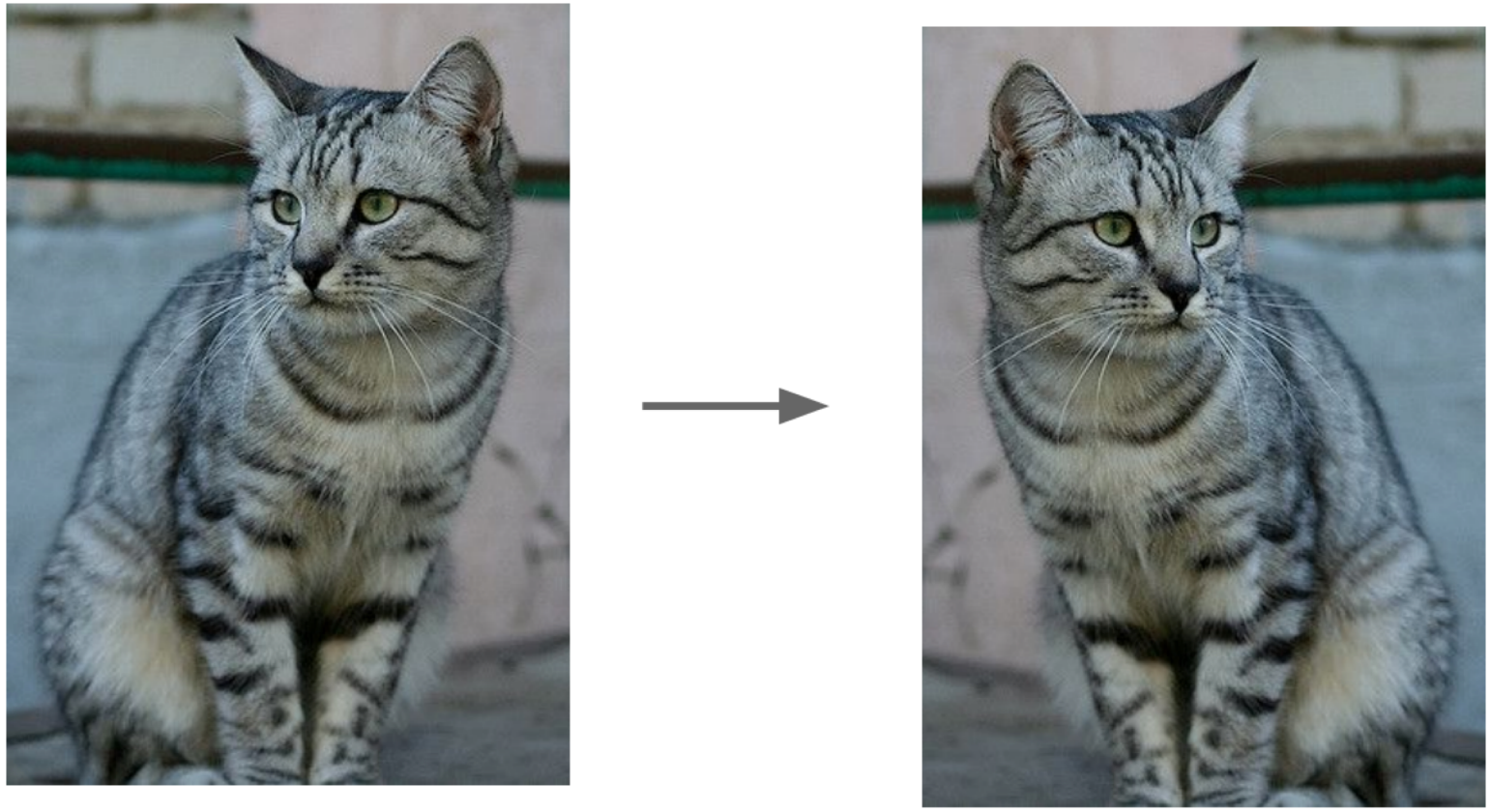
Increase size of your training data (‘augmentation’), by various transformations.

Credit: Lazebnik, CS231n, source1, source2

Data augmentation

Increase size of your training data ('augmentation'), by various transformations.

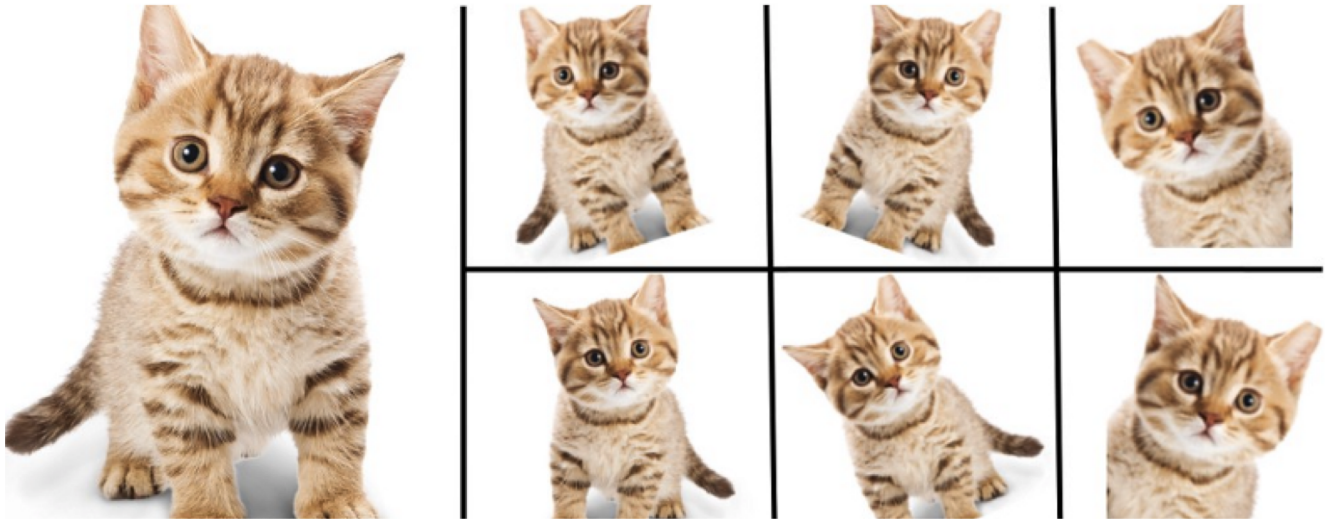
- Horizontal flips

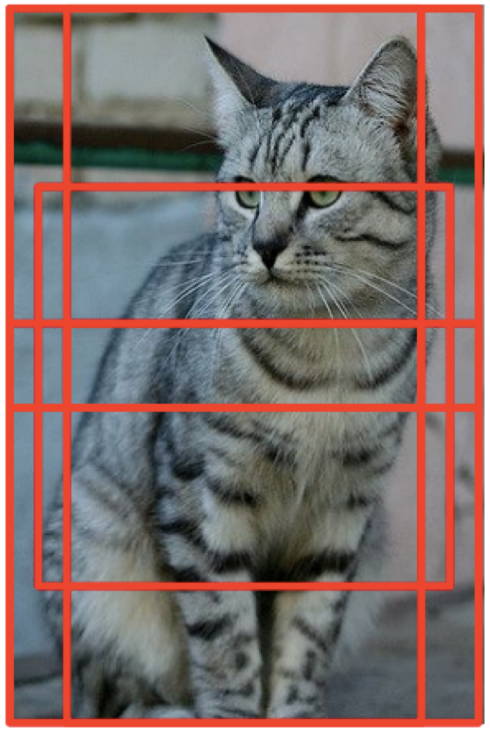


Credit: Lazebnik, CS231n, source

Data augmentation

- Increase size of your training data ('augmentation'), by various transformations.
  - Horizontal flips
  - Rotation, shearing
  - Random crops



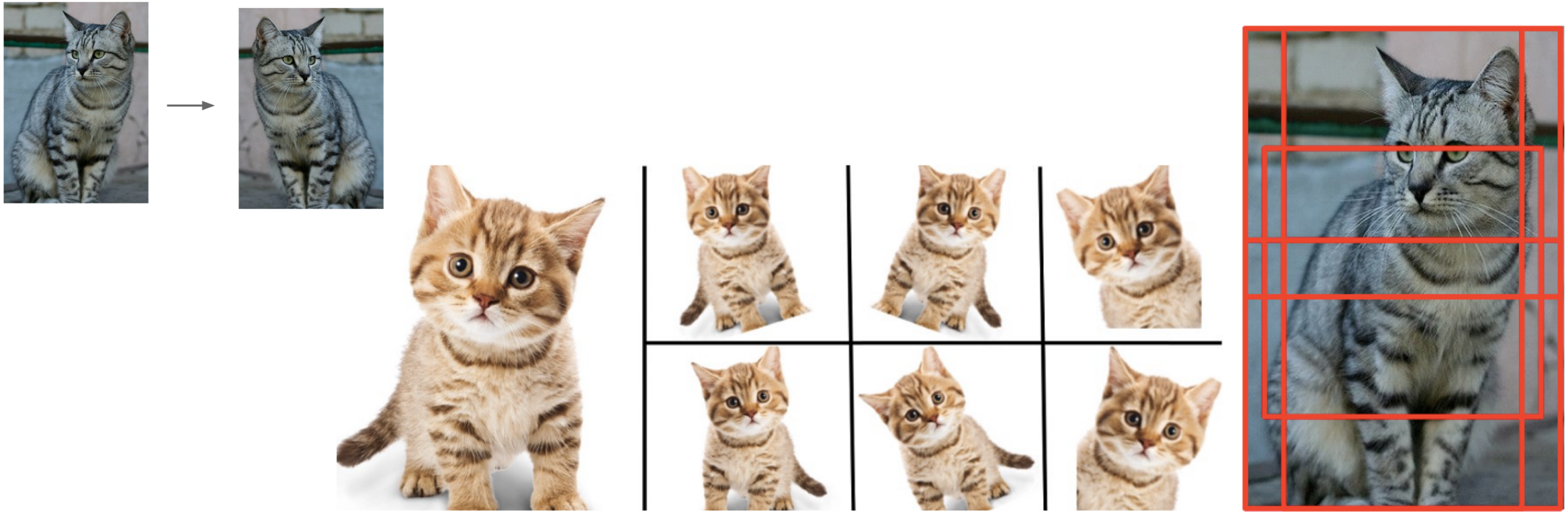


Credit: Lazebnik, CS231n, source1, source2

Data augmentation

Increase size of your training data ('augmentation'), by various transformations.

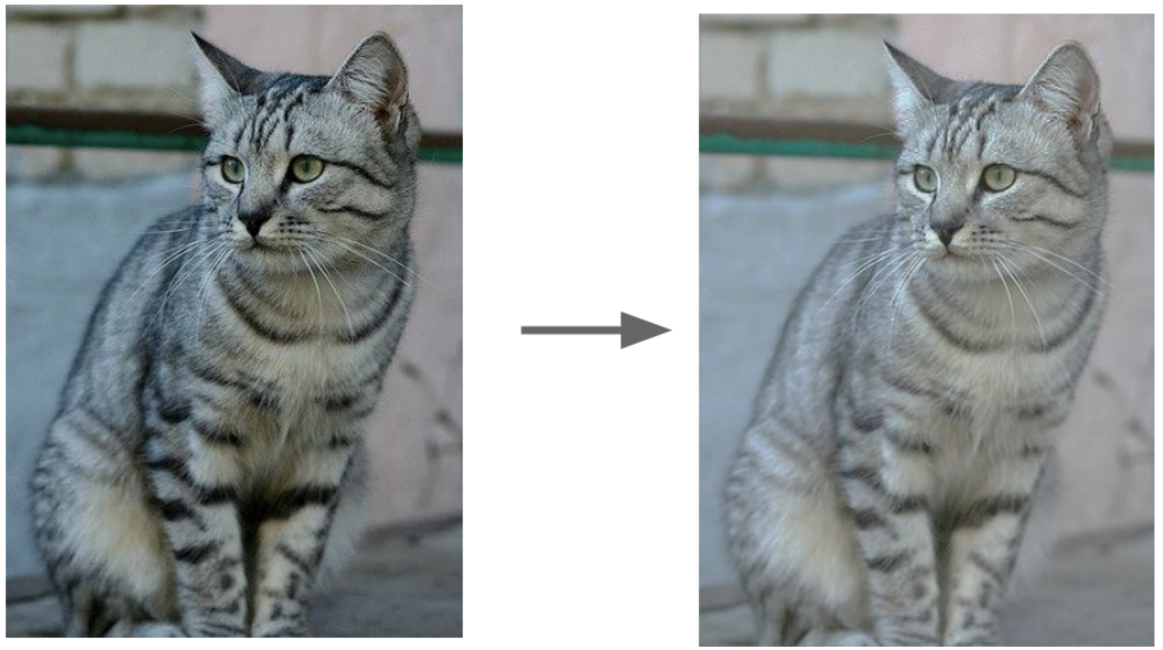
- Geometric: flipping, rotation, shearing, random crops



Credit: Lazebnik, CS231n, source1, source2

Data augmentation

- Increase size of your training data ('augmentation'), by various transformations.
  - Geometric: flipping, rotation, shearing, random crops
  - Photometric: color transformations



Credit: Lazebnik, CS231n, source

Data augmentation

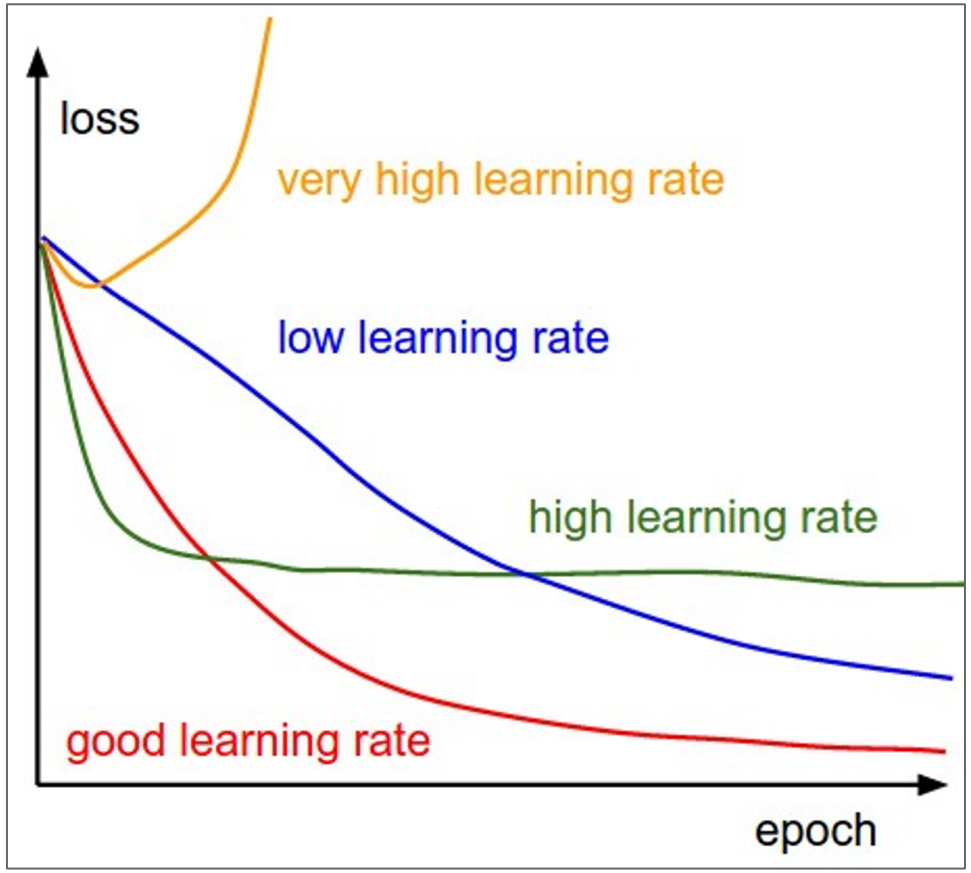
- Increase size of your training data ('augmentation'), by various transformations.
  - Geometric: flipping, rotation, shearing, random crops
  - Photometric: color transformations
  - Other: add noise, compression artifacts, lens distortions, etc.

Credit: Lazebnik, CS231n, source

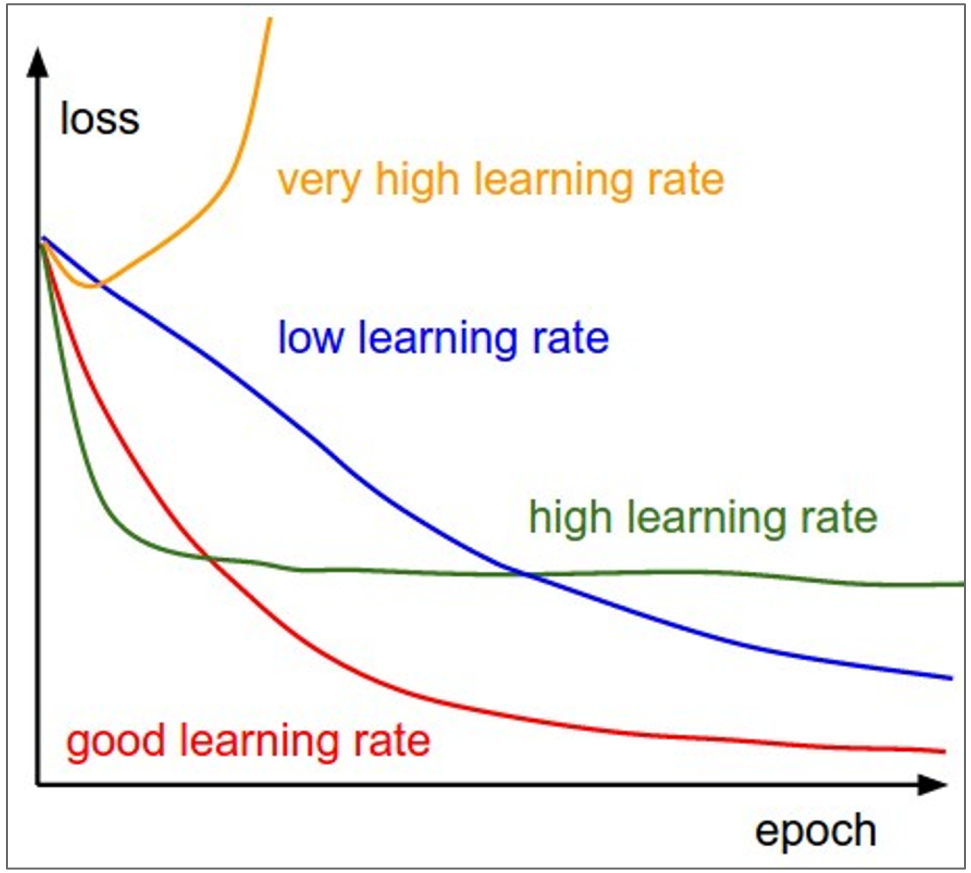
Regularization#4: Early Stopping

Learning Rate Schedules

SGD,SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.

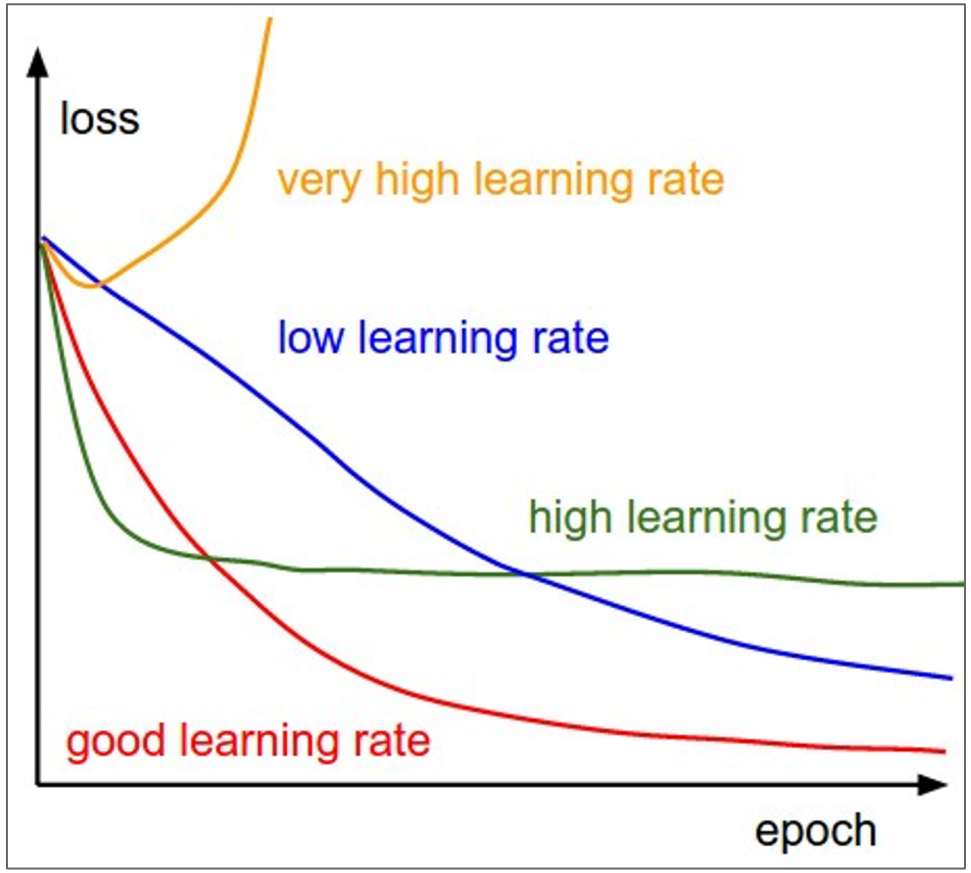


SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Q: Which one of these learning rates is best to use?

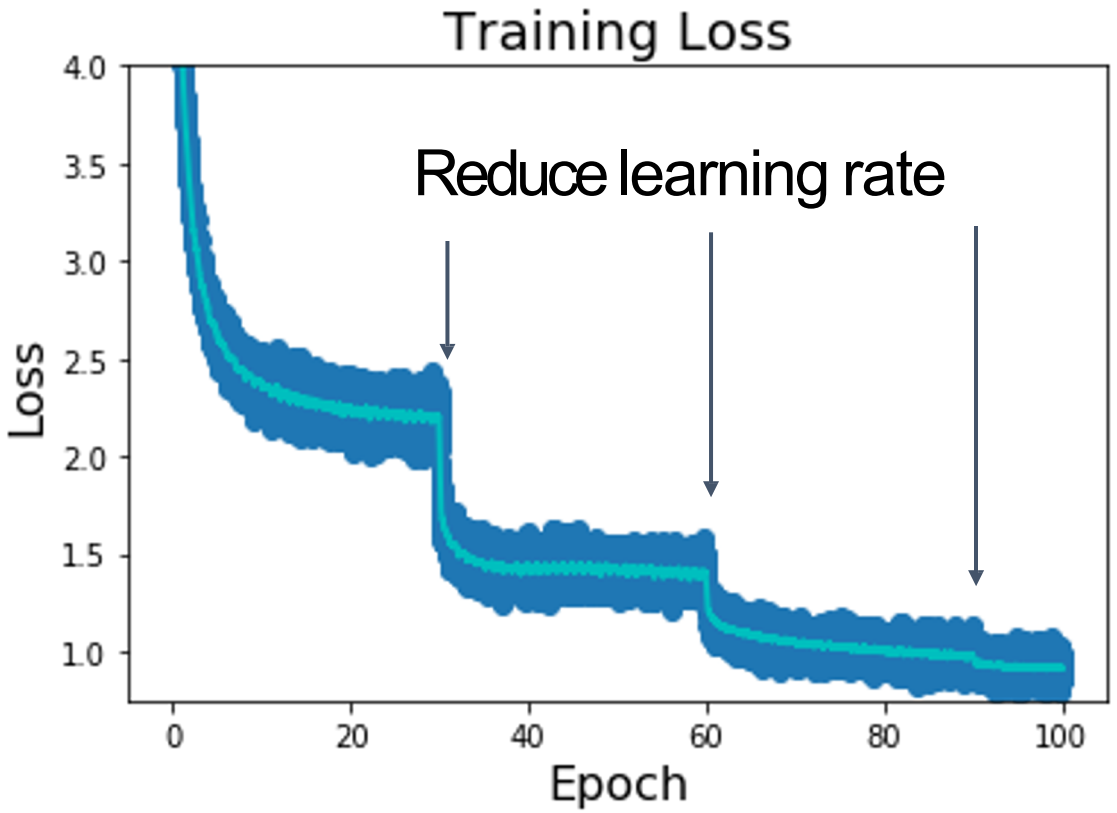
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



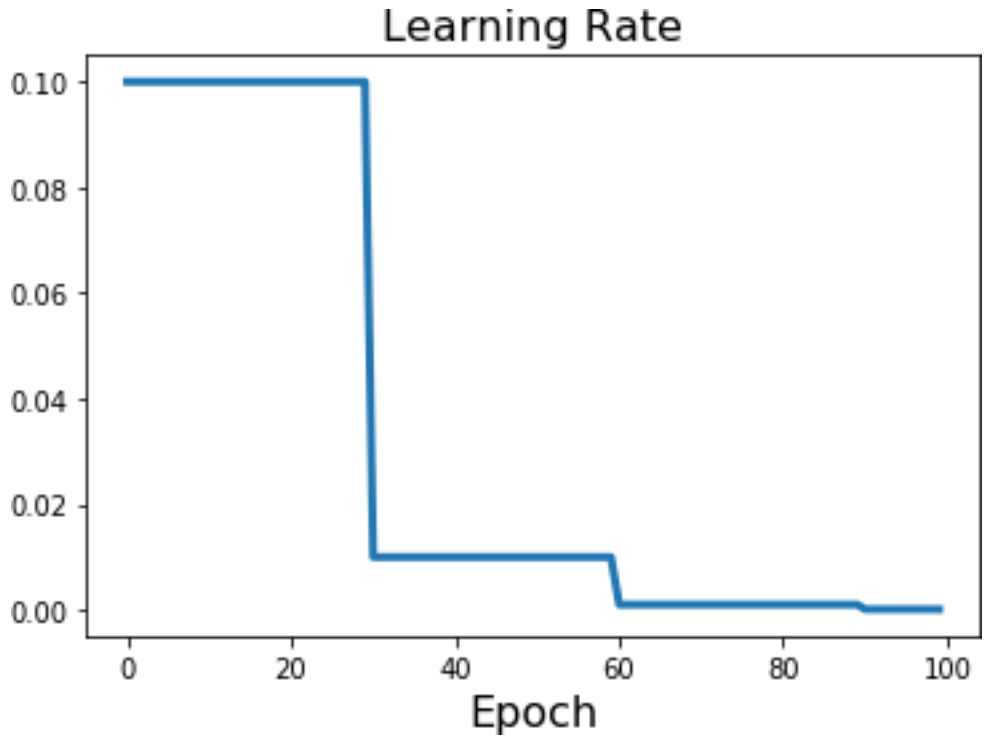
Q: Which one of these learning rates is best to use?

A: All of them! Start with large learning rate and decay over time

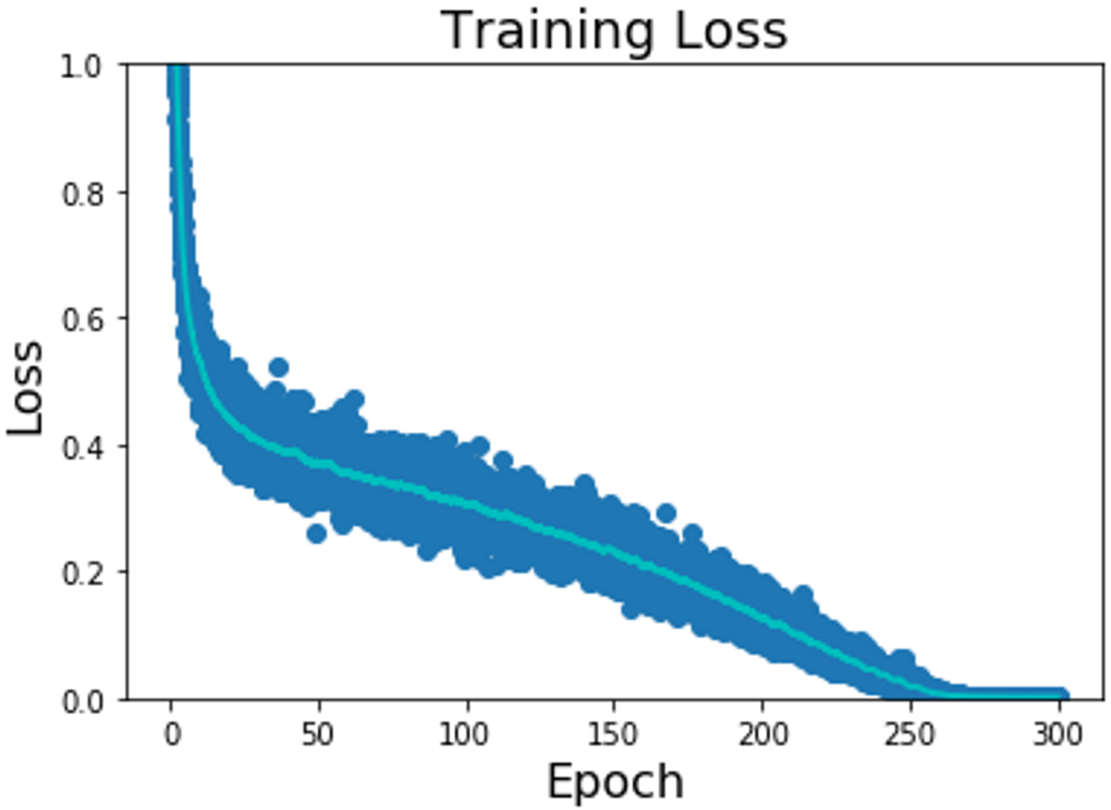
Learning Rate Decay: Step



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

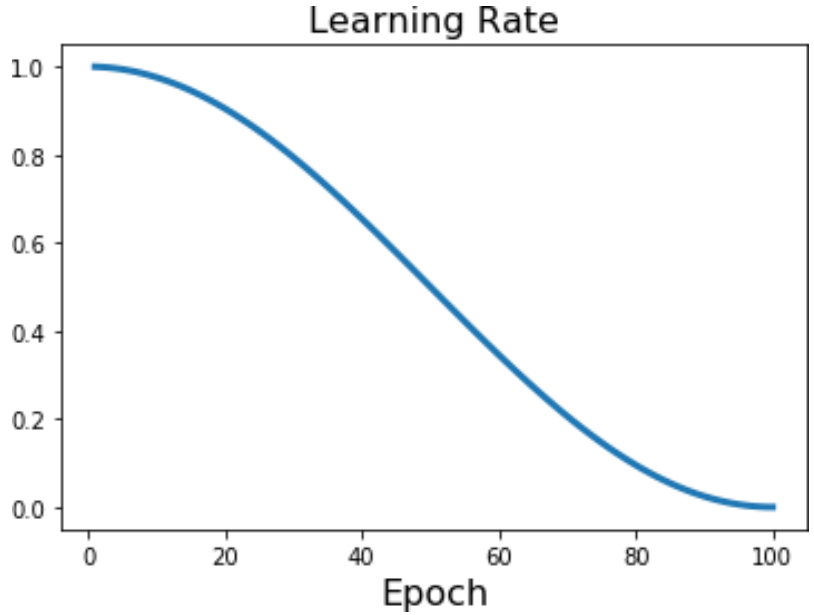


Learning Rate Decay: Cosine



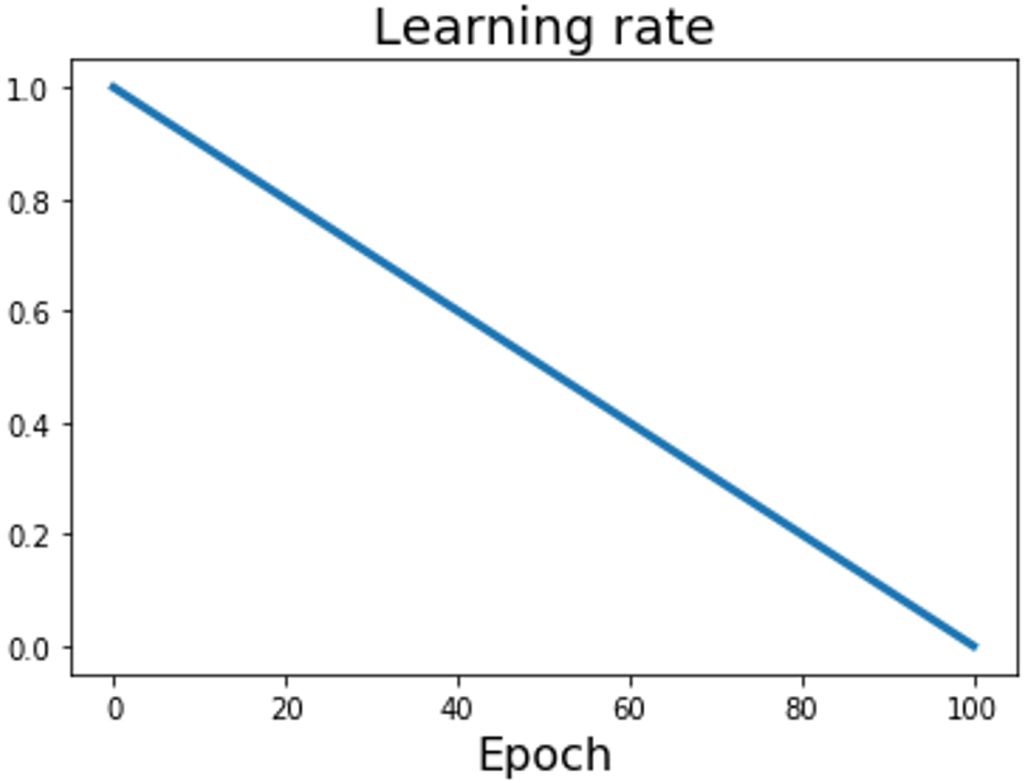
Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$$



Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017
Radford et al, "Improving Language Understanding by Generative Pre-Training", 2018
Feichtenhofer et al, "SlowFast Networks for Video Recognition", ICCV 2019
Radosavovic et al, "On Network Design Spaces for Visual Recognition", ICCV 2019
Child at al, "Generating Long Sequences with Sparse Transformers", arXiv 2019

Learning Rate Decay: Linear

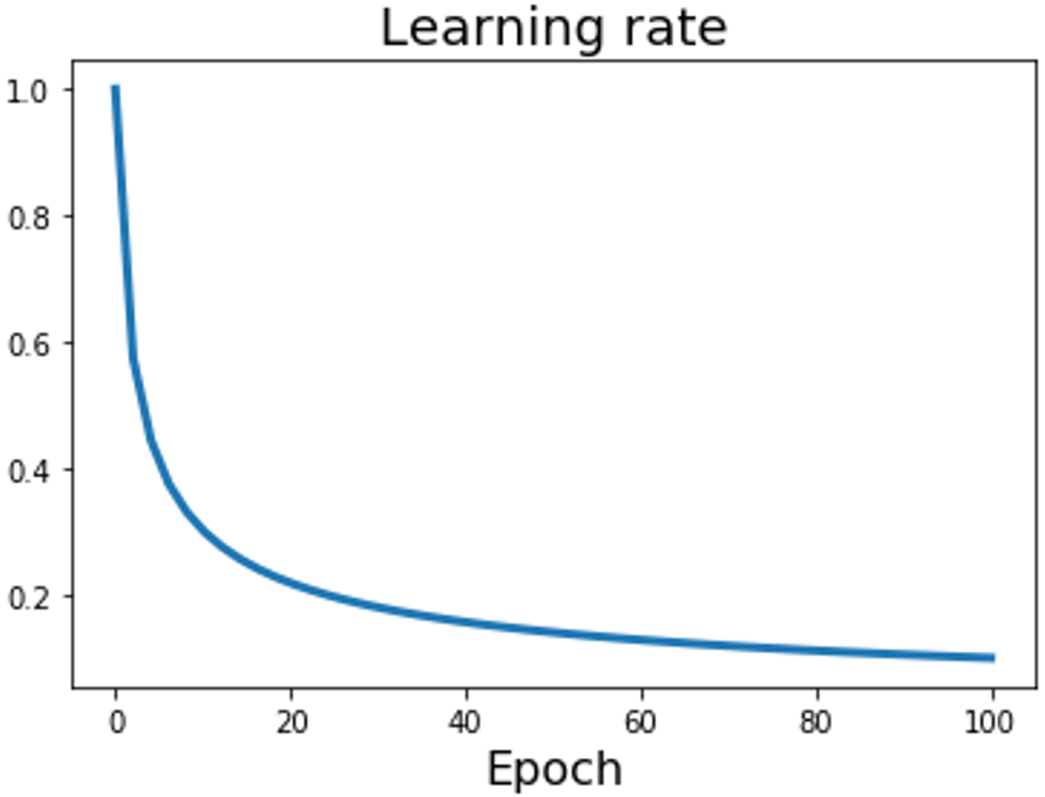


Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$

Linear: $\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$

Learning Rate Decay: Inverse Sqrt



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$

Linear: $\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$

Inverse sqrt: $\alpha_t = \alpha_0/\sqrt{t}$

Any Question ?