Different Elements in Training Convolutional Neural Networks 2

Xiaolong Wang

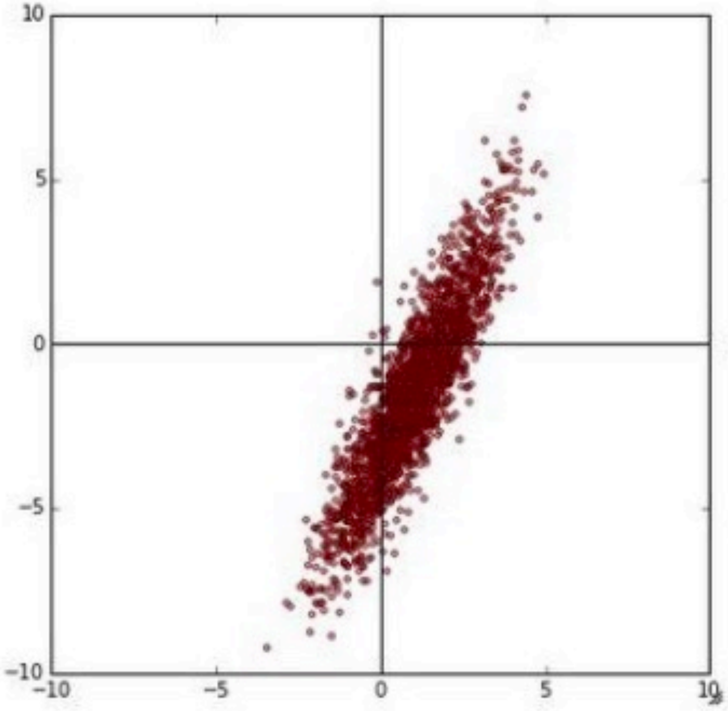
This Class

- Data Augmentation and Pre-processing
- Weight Initialization
- Batch Normalization
- Regularization in Training Deep Networks

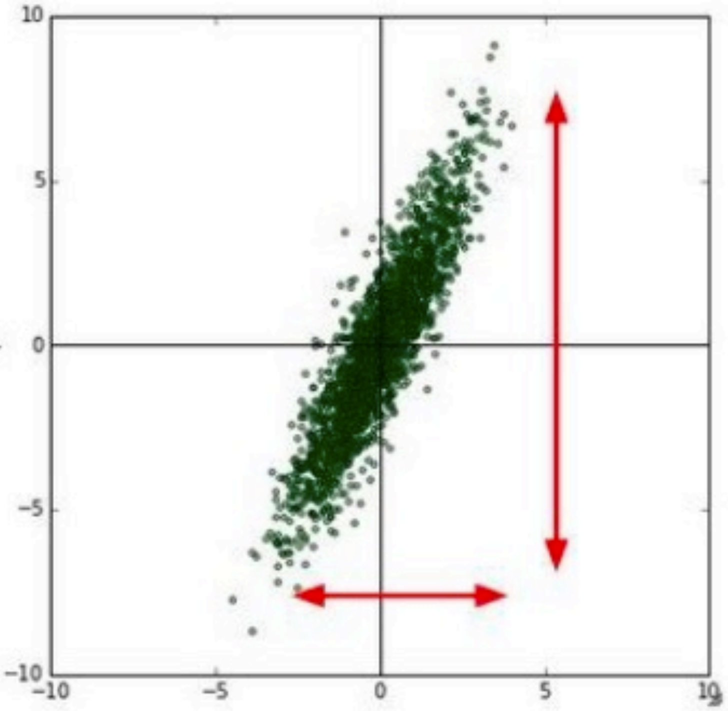
Images partially from: http://cs231n.stanford.edu/, https://slazebni.cs.illinois.edu/fall20/

Data Augmentation and Pre-processing

Data Pre-Processing

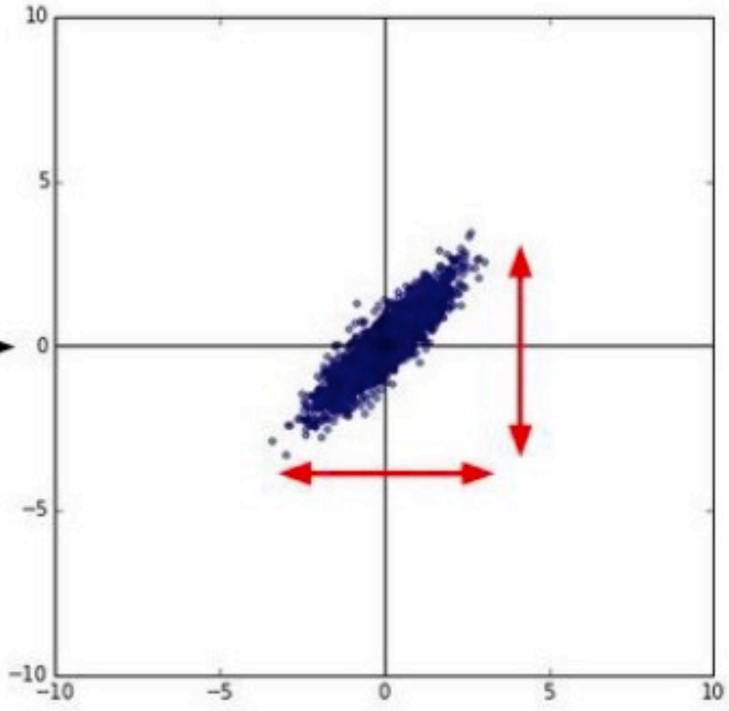


Input data



Zero-centered data

$$X = X - mean(X)$$



normalized data

$$X = \frac{X}{std(X)}$$

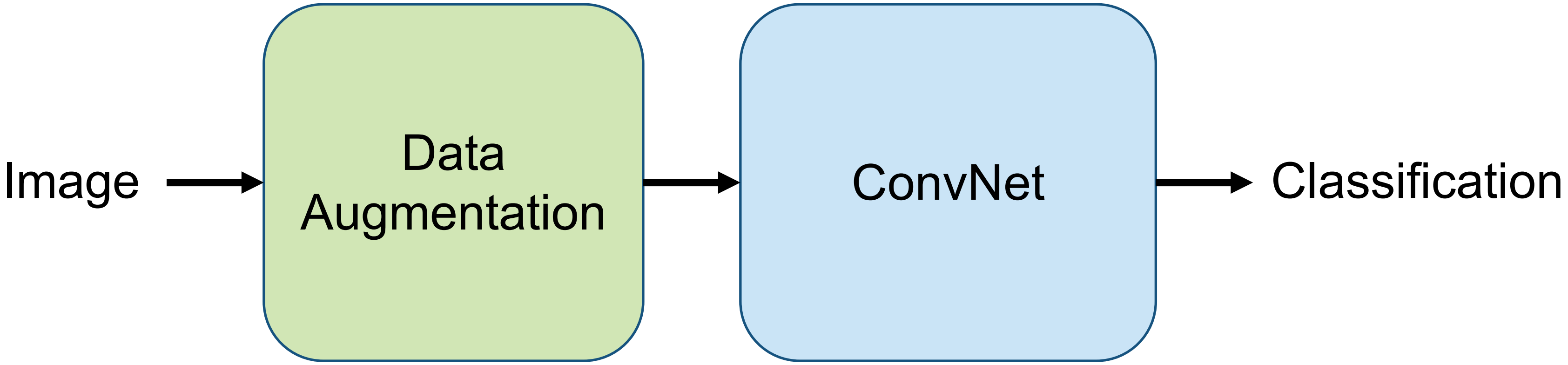
Data Pre-Processing

- Subtract mean and divide the std is optional if we have batch normalization (will introduce later)

- Should maintain the same input process for both training and testing

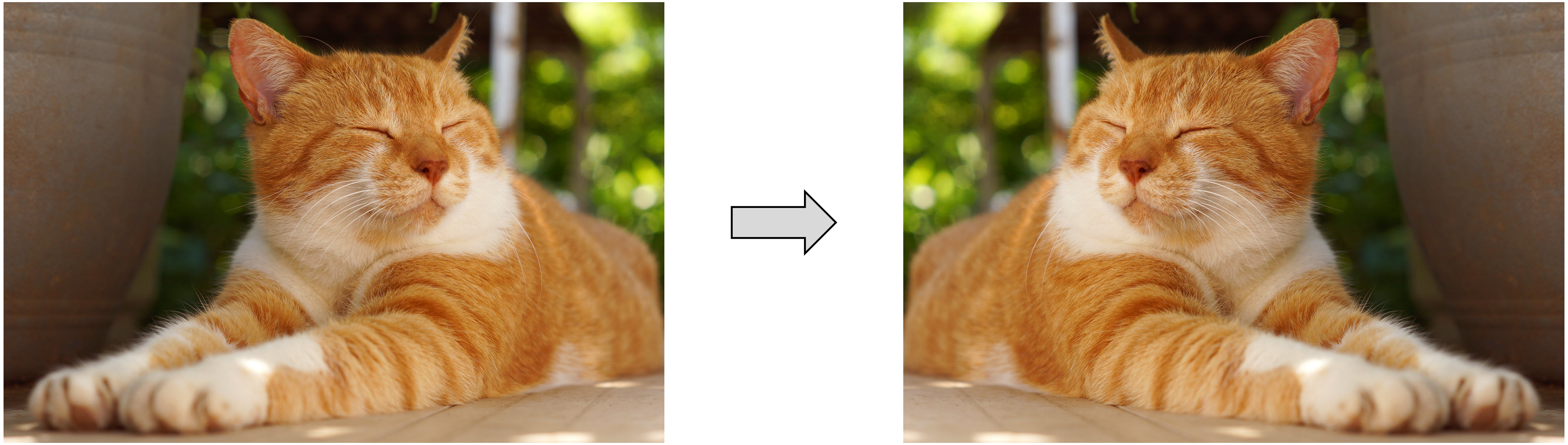
Data Augmentation

- Data augmentation is a free way to increase training data
- Prevent overfitting
- Improve performance



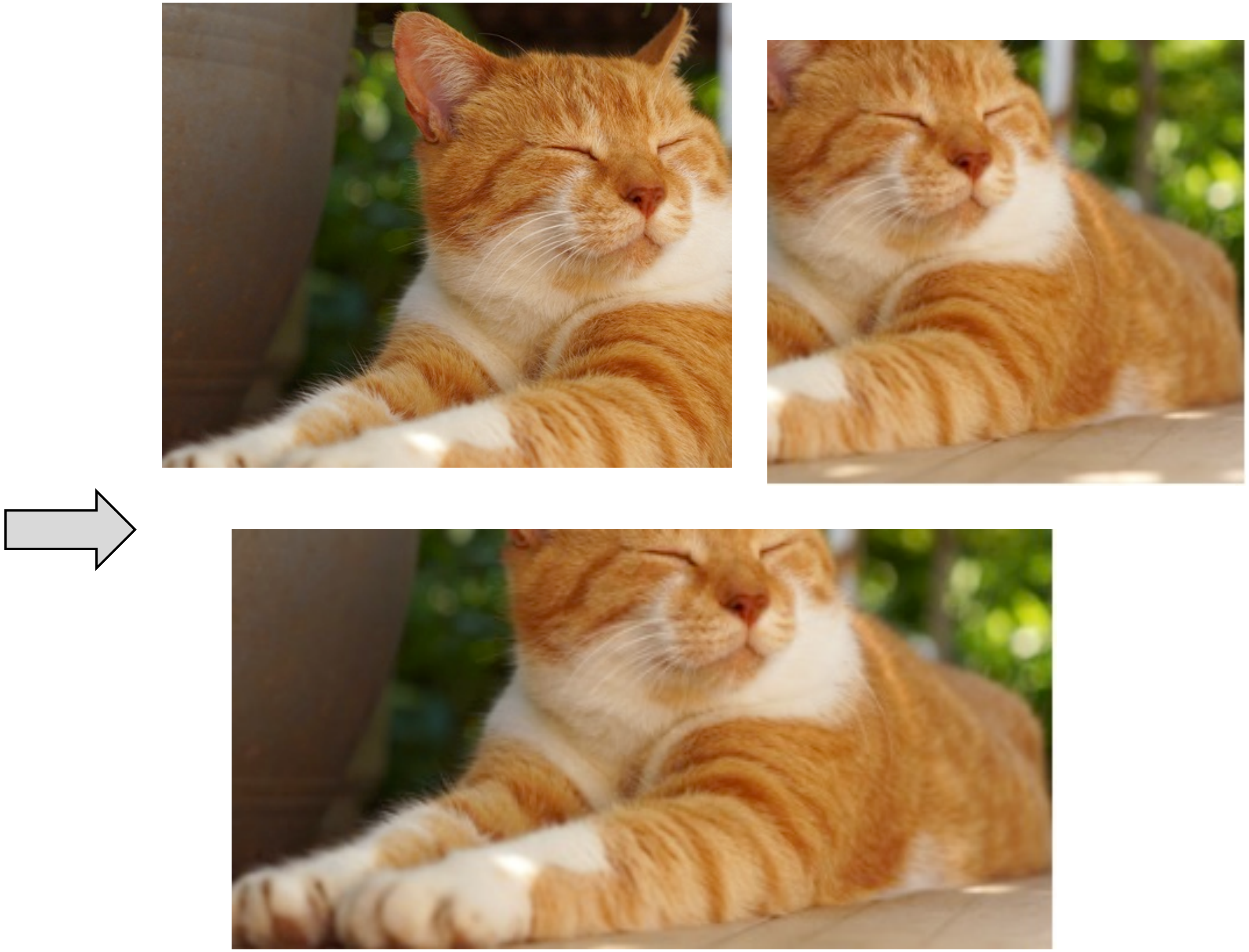
Data Augmentation for Classification

- Horizontal Flip (useful)



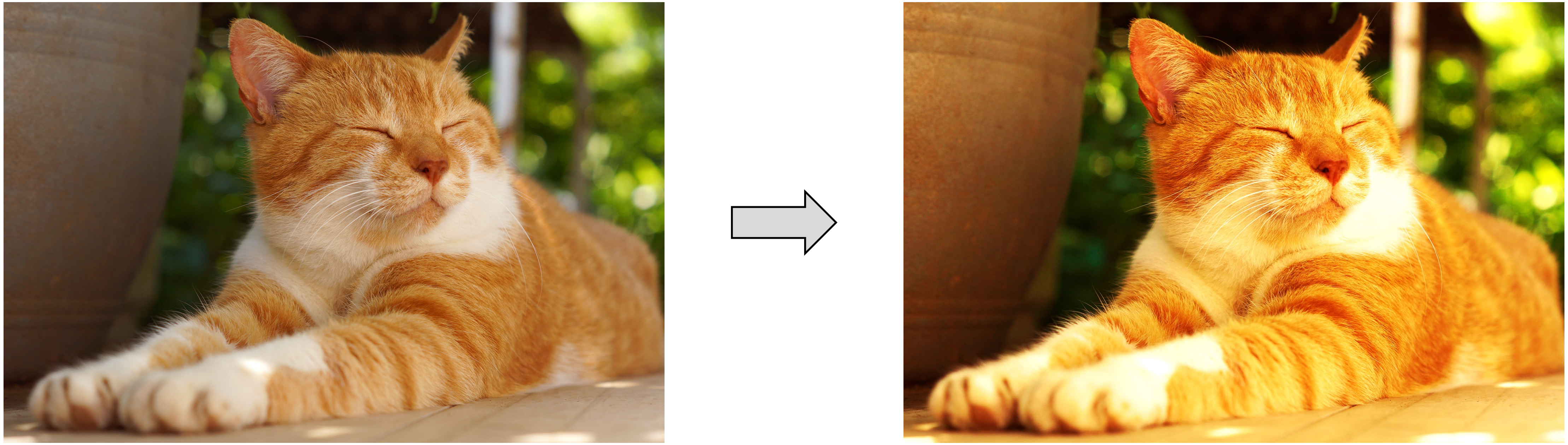
Data Augmentation for Classification

- Random Crop (critical)



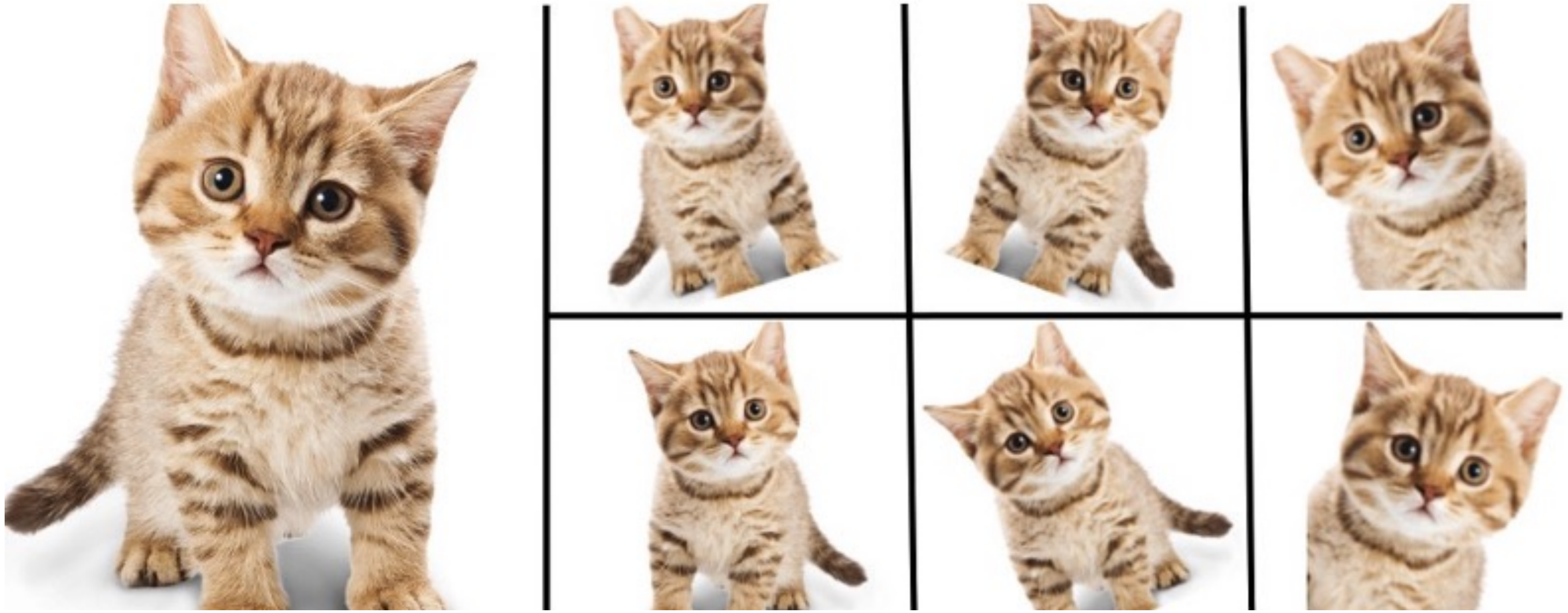
Data Augmentation for Classification

- Color augmentation, brightness, contrast (can ignore)



Data Augmentation for Classification

- Rotation (sometimes useful, especially for pose estimation)



Data Augmentation for Classification

- Training:
  - Pick a random L in range [256, 480]
  - Resize the image, the short side is resized to length L, maintaining the original aspect ratio
  - Randomly crop an [224, 224] patch out of the image

- Testing:
  - Resize the image, the short side is resized to length 256
  - Crop an [224, 224] patch from the center of the image

Weight Initialization

Gaussian Initialization

- Gaussian initialization with zero mean and 1e-2 standard deviation

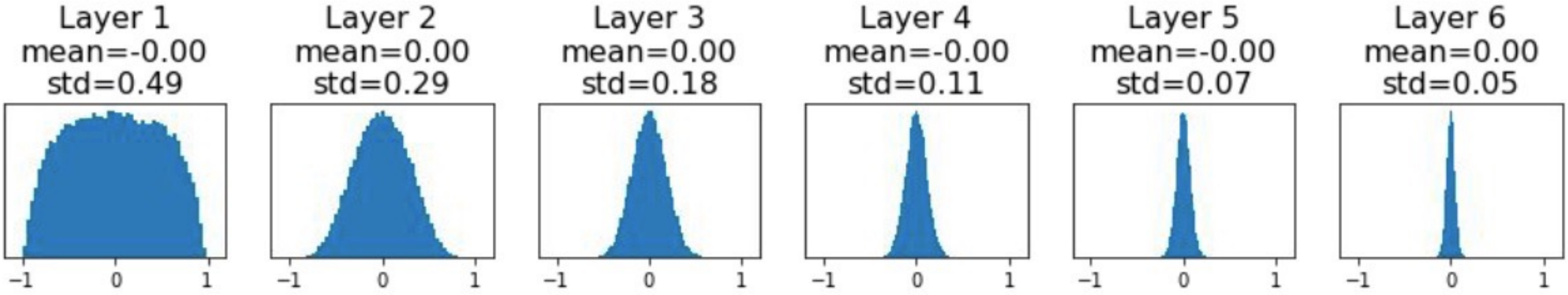
```
W = 0.01 * np.random.randn(Din, Dout)
```

- np.random.randn samples from a gaussian distribution with zero mean and 1 std

Gaussian Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

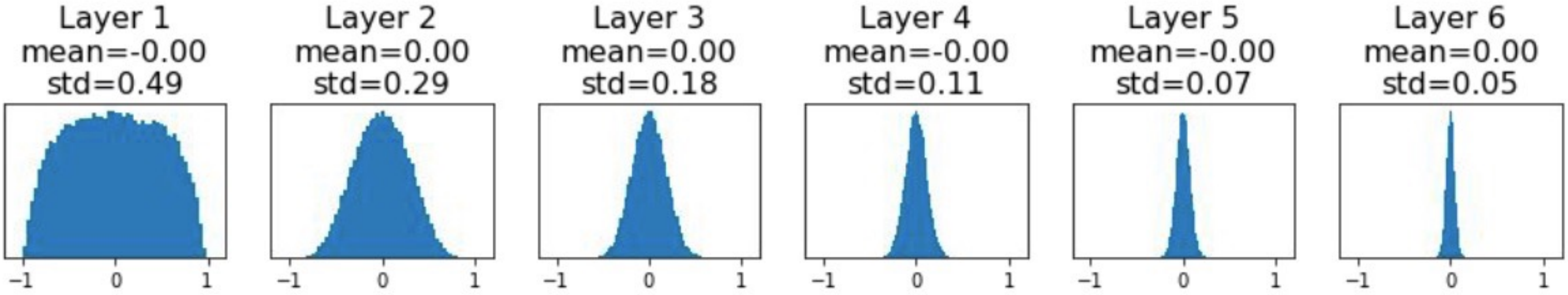
Forward pass for a 6-layer net with hidden size 4096



Gaussian Initialization

- The magnitude of the activations become smaller and smaller for higher layers

- We want the magnitude to be maintained over the layers



Why is it important to maintain the magnitude of activations?

$$\frac{\partial e}{\partial W_k} = \frac{\partial e}{\partial h_k} \boxed{\frac{\partial h_k}{\partial W_k}}$$ activations

$$\frac{\partial e}{\partial h_{k-1}} = \frac{\partial e}{\partial h_k} \frac{\partial h_k}{\partial h_{k-1}}$$

Compute $\frac{\partial h_k}{\partial W_k}$

Layer k

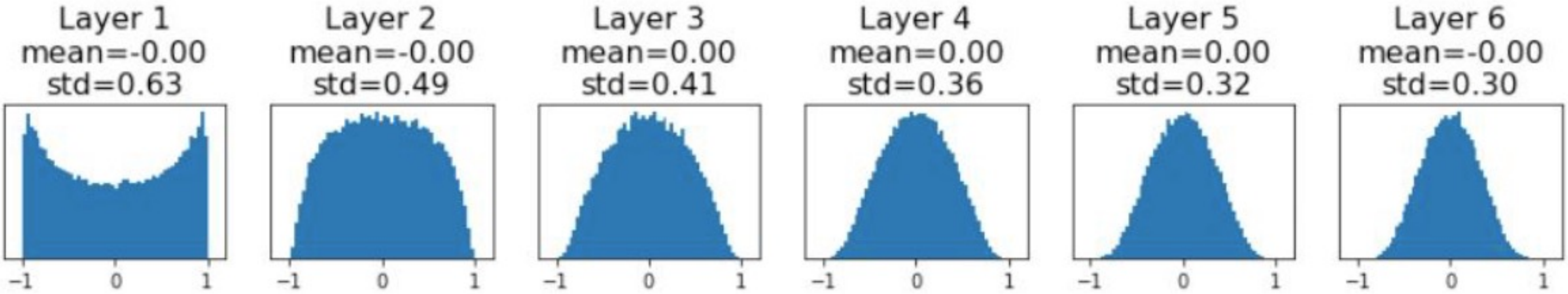
Compute $\frac{\partial h_k}{\partial h_{k-1}}$

$$\frac{\partial e}{\partial h_k}$$

Xavier Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)



Batch Normalization

Batch Normalization

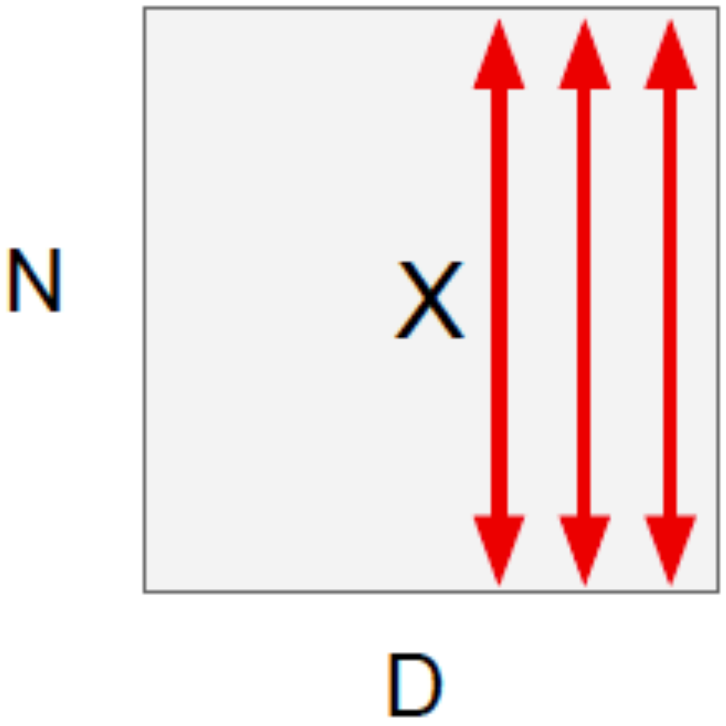
- Explicitly enforce each layer to have zero-mean and unit-variance outputs

- A basic version of batch norm:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

Batch Normalization for FC layer

Input: $x \in \mathbb{R}^{N \times D}$



$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Compute mean for each channel $\mu \in \mathbb{R}^{D}$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2$$

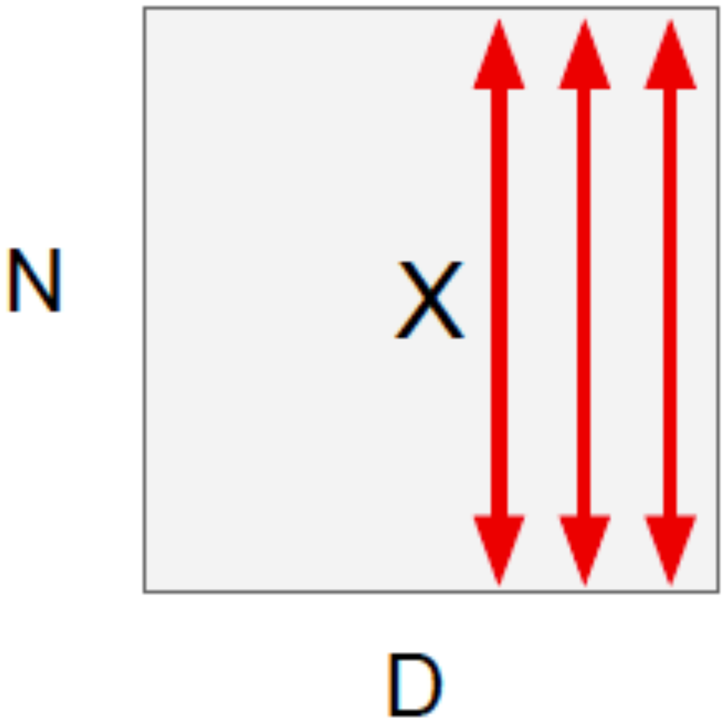
Compute variance for each channel $\sigma^2 \in \mathbb{R}^{D}$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalize $x \in \mathbb{R}^{N \times D}$

Batch Normalization for FC layer

Input: $x \in \mathbb{R}^{N \times D}$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Compute mean for each channel $\mu \in \mathbb{R}^{D}$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}\left(x_{i,j} - \mu_j\right)^2$$

Compute variance for each channel $\sigma^2 \in \mathbb{R}^{D}$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

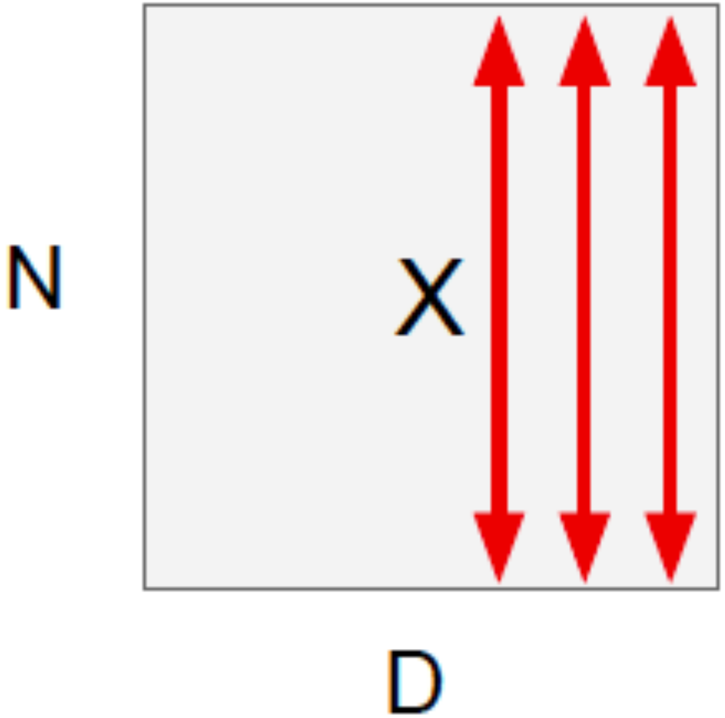
Normalize $x \in \mathbb{R}^{N \times D}$

$$y_{i,j} = \gamma_j \, \hat{x}_{i,j} + \beta_j$$

Scale with learnable parameters $\gamma \in \mathbb{R}^{D}, \beta \in \mathbb{R}^{D}$

During Test Time

Input: $x \in \mathbb{R}^{N \times D}$



$$\cancel{\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}}$$

A running average of μ during training

$$\cancel{\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2}$$

A running average of σ^2 during training

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

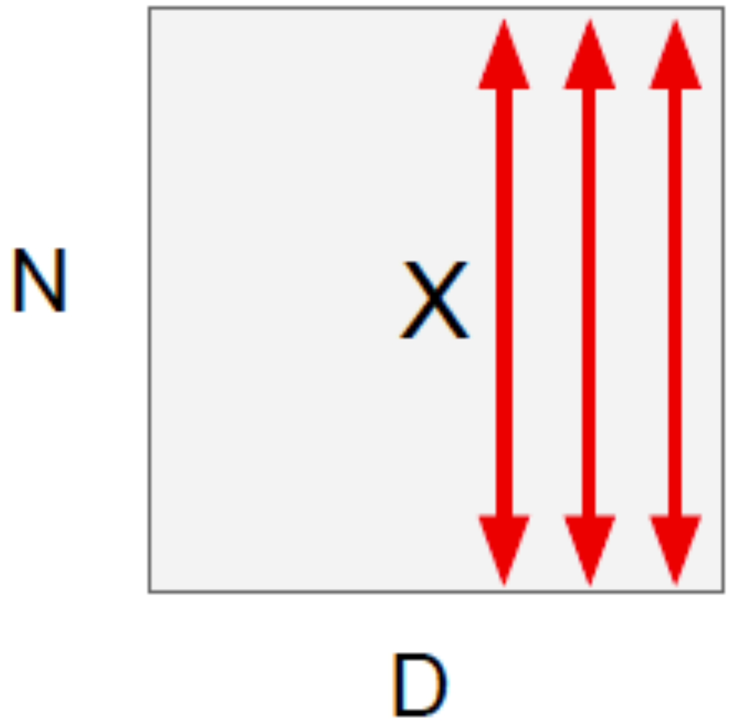
Normalize $x \in \mathbb{R}^{N \times D}$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Scale with learnable parameters $\gamma \in \mathbb{R}^D, \beta \in \mathbb{R}^D$

During Test Time

Input: $x \in \mathbb{R}^{N \times D}$



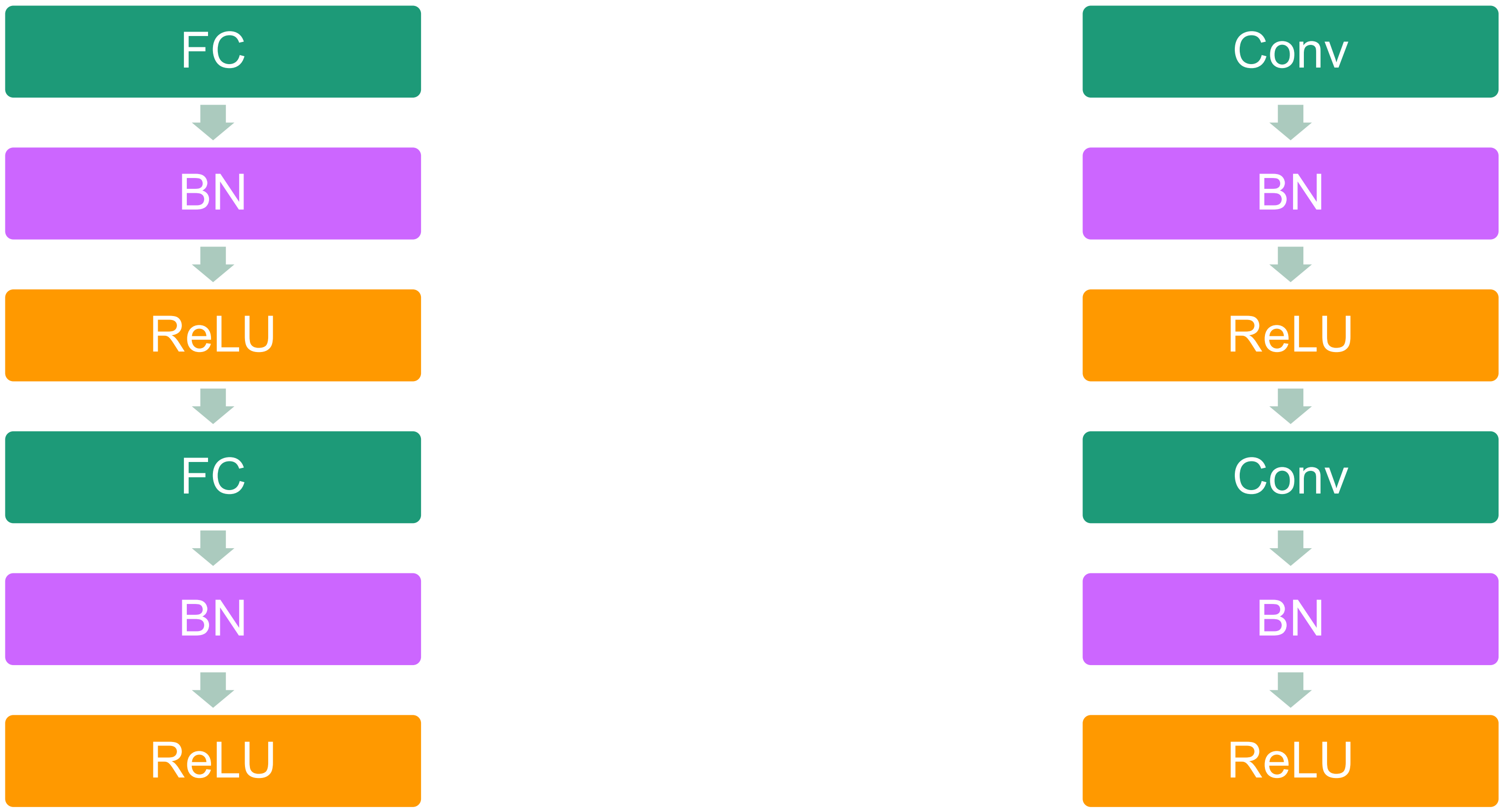
A running average of μ during training:

$$\hat{\mu}_t = \alpha \hat{\mu}_{t-1} + (1 - \alpha)\mu_{t-1}$$

A running average of σ^2 during training:

$$\hat{\sigma}_t^2 = \alpha \hat{\sigma}_{t-1}^2 + (1 - \alpha)\sigma_{t-1}^2$$

Batch Normalization in Deep Networks

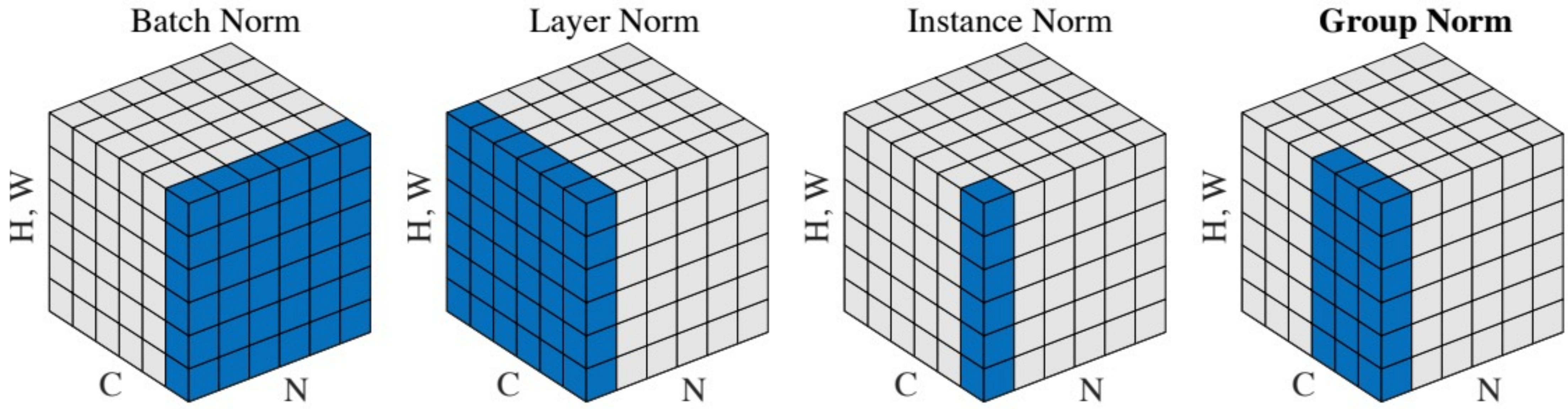


Batch Normalization for ConvNets

MLPs	ConvNets
$\mathbf{x}: N \times D$	$\mathbf{x}: N \times C \times H \times W$
Normalize	Normalize
$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times D$	$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times C \times 1 \times 1$
$\gamma, \beta: 1 \times D$	$\gamma, \beta: 1 \times C \times 1 \times 1$
$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$	$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

Other Normalization layers

- Layer normalization (Ba et al., 2016)
- Instance normalization (Ulyanov et al., 2017)
- Group normalization (Wu and He, 2018)



Y. Wu and K. He, Group Normalization, ECCV 2018

Regularization

Prevent overfitting: L2 regularization

- Adding regularization in training objective, L2 regularization:

$$\hat{L}(W) \;=\; \underbrace{\frac{\lambda}{2}||W||^2}_{\text{L2 regularization}} \;+\; \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(W, x_i, y_i)}_{\text{Loss from data}}$$

$$W \leftarrow W - \underline{\alpha\,(\lambda\, W} + \nabla_W \frac{1}{n}\sum_{i=1}^{n} L(W, x_i, y_i)\,)$$

Prevent overfitting: L2 regularization

$$W \leftarrow W - \underline{\alpha\,(\lambda\, W} + \nabla_W \frac{1}{n}\sum_{i=1}^{n} L(W, x_i, y_i)\,)$$

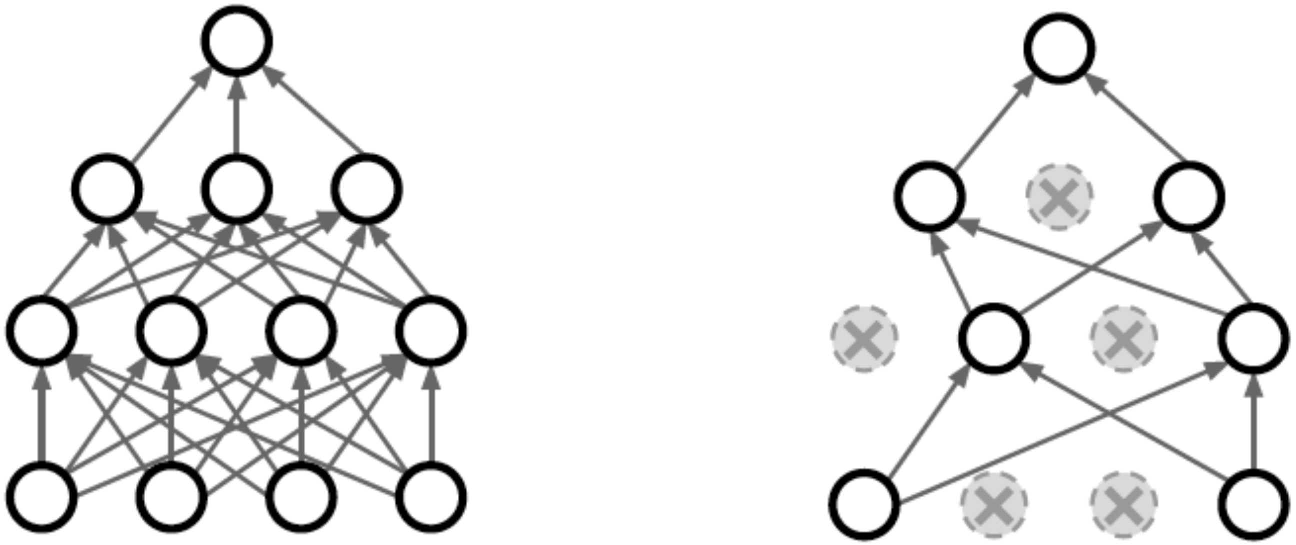
Gradients from
L2 regularization

Also called weight decay

We usually set $\lambda = 0.00005$

Dropout

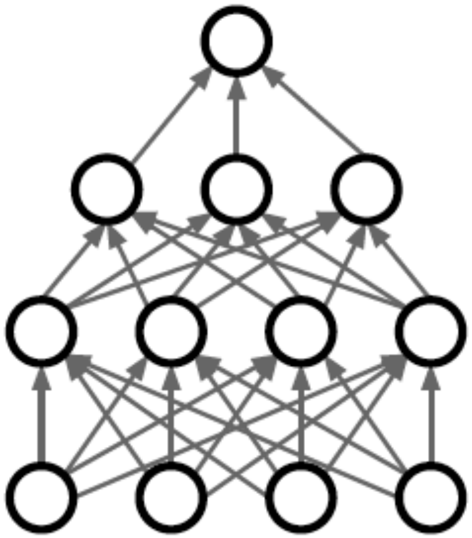
- At training time, in each forward pass, turn off some neurons with probability *p*
- Usually set *p* = 0.5

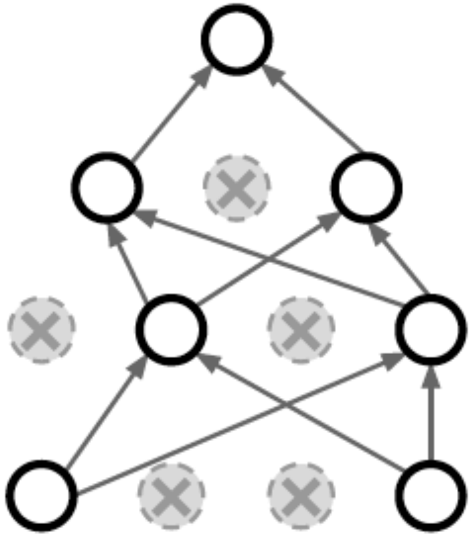


N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR 2014

Dropout

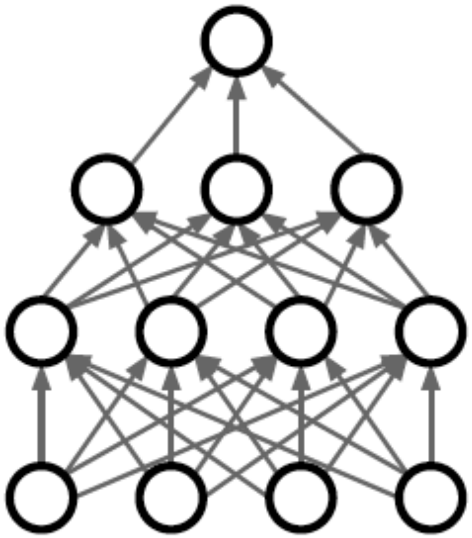
- During test time, do not apply dropout but multiply all the output by *p* to maintain the same magnitude of activations

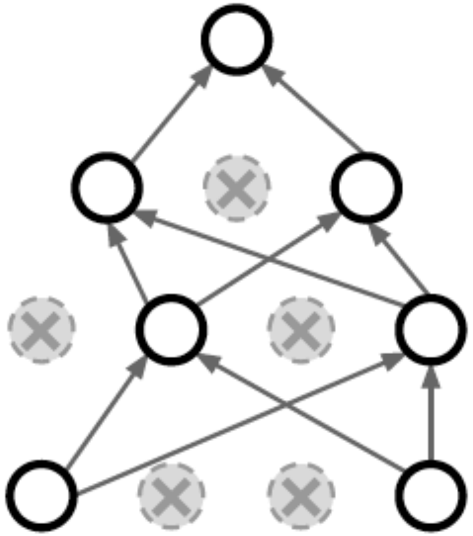




Why Dropout

- Increase robustness to noise
- Implicitly training multiple different networks, and test with multiple network ensamble





Dropout

- Not used a lot currently in training
- Less useful when the dataset is large and applying data augmentation
- Still useful when training with video dataset/task since there is less data than image datasets

This Class

- Data Augmentation and Pre-processing
- Weight Initialization
- Batch Normalization
- Regularization in Training Deep Networks

Next Class

Convolutional Neural Networks architectures