



Introduction to Computer Vision

Lecture 5 - Deep Learning II

Prof. He Wang

Logistics

- Assignment 1: due on 3/30 11:59PM (Saturday)
 - If 1 day (0 - 24 hours) past the deadline, 15% off
 - If 2 day (24 - 48 hours) past the deadline, 30% off
 - Zero credit if more than 2 days.

Logistics

- Assignment 1: due on 4/1 11:59PM (Saturday)

论坛: 作业一

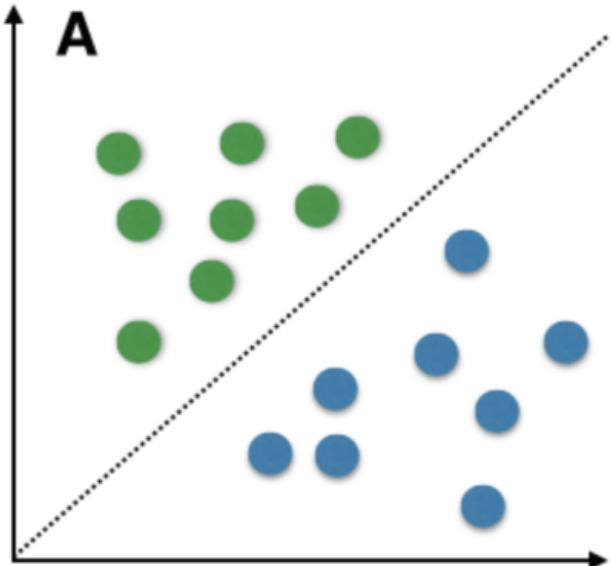
论坛由可围绕特定主题进行组织的独立讨论话题构成。话题是论坛中的对话，其中包括原始帖子及该帖子的所有回复。访问论坛时，会显示

创建话题 订阅

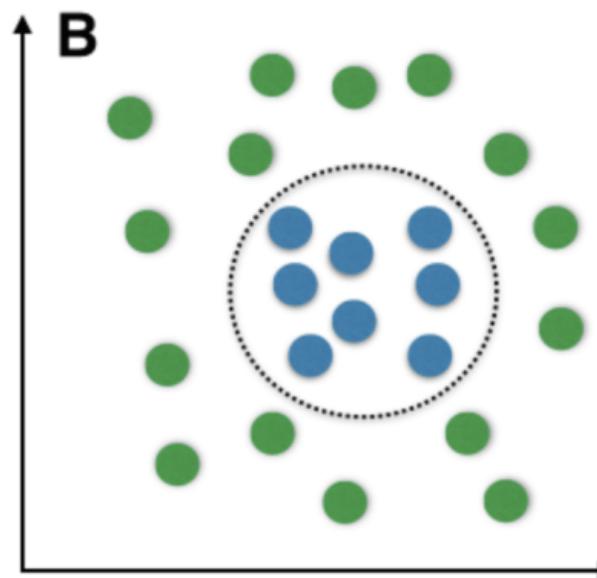
操作	日期	话题	作者	状态
<input type="checkbox"/>	24-3-20 下午2:12	调超参有什么窍门	元培学院 周俊廷	已发布
<input type="checkbox"/>	24-3-20 下午1:42	Q2(b) 中np.roll的使用	元培学院 李长烨	已发布
<input type="checkbox"/>	24-3-19 下午3:39	Q2 (b) 的输入输出	信息科学技术学院 白杨	已发布
<input type="checkbox"/>	24-3-19 下午1:26	请问是否允许在 list comprehension 用到 for	信息科学技术学院 杨涛瑜	已发布
<input type="checkbox"/>	24-3-19 上午12:13	作业重复提交	元培学院 彭莘尧	已发布
<input type="checkbox"/>	24-3-18 下午3:50	作业中hints的使用	信息科学技术学院 刘星宇	已发布
<input type="checkbox"/>	24-3-17 上午10:47	作业一中用Toeplitz矩阵做卷积	信息科学技术学院 张智琦	已发布
<input type="checkbox"/>	24-3-16 下午7:35	Task 1(a) 代码可以更加完善	元培学院角色 倪嘉怡	已发布

Problem with Single-Layer Network

- $g(\theta^T x) = 0$ is a hyperplane in the space of x
- can only handle linear separable cases

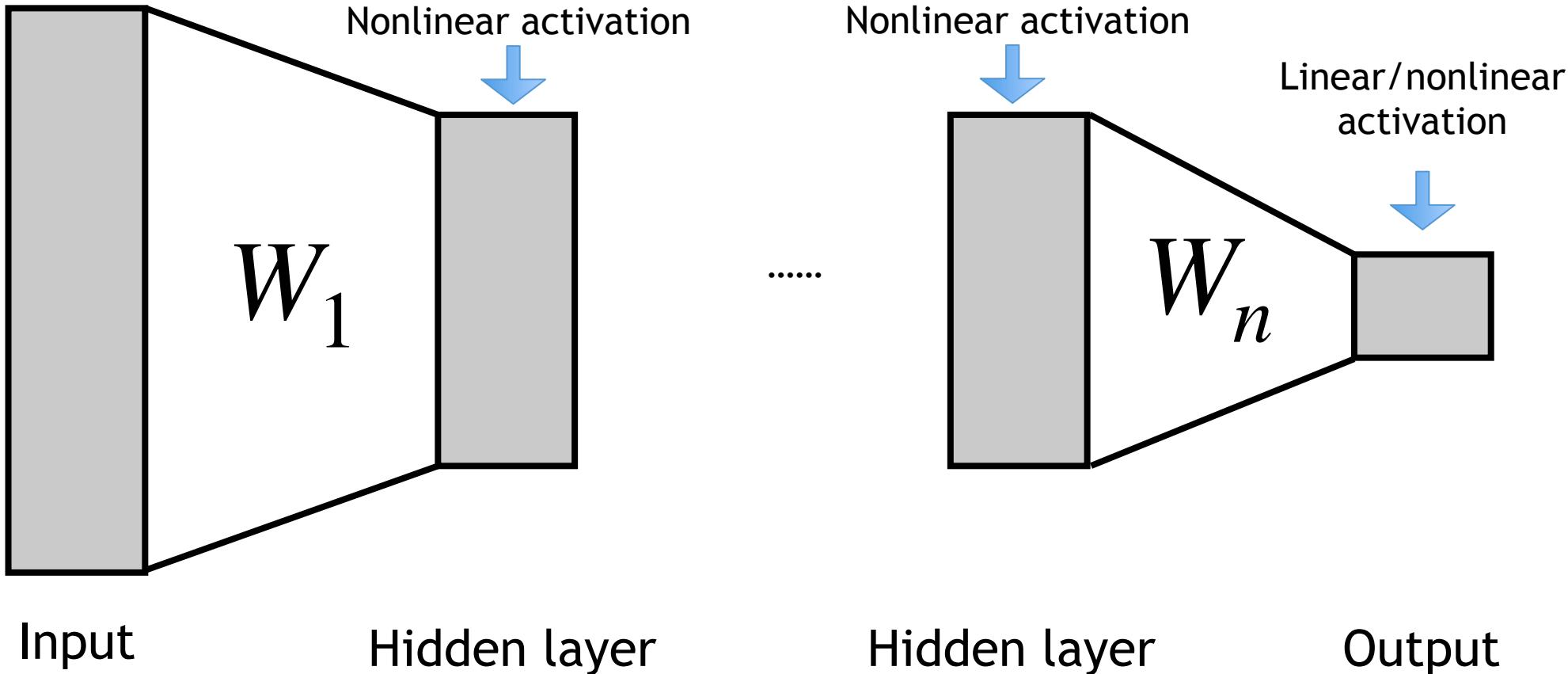


Linear separable



Linear non-separable

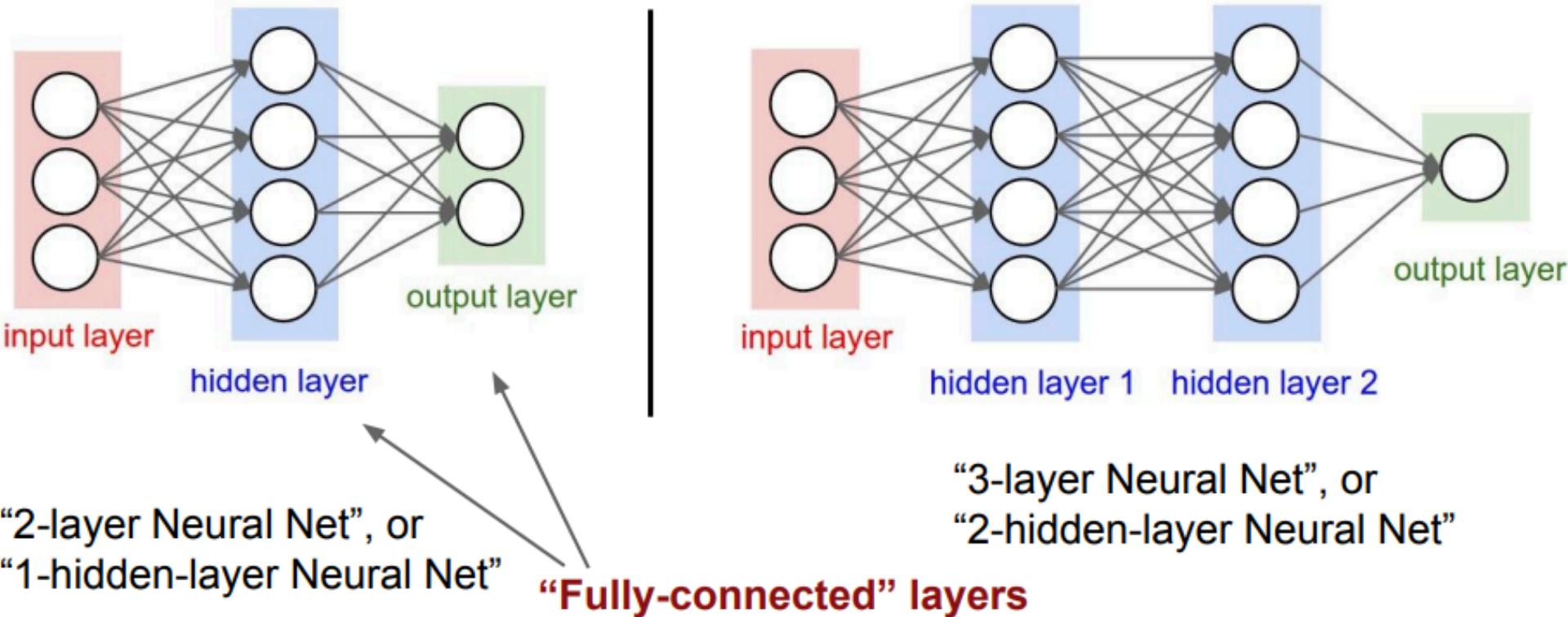
Multi-Layer Perceptron (MLP)



- MLP: Stacking linear layer and nonlinear activations.
- Through many non-linear layers, transform a linear non-separable problem to linear separable at the last layer

Review

Neural networks: Architectures



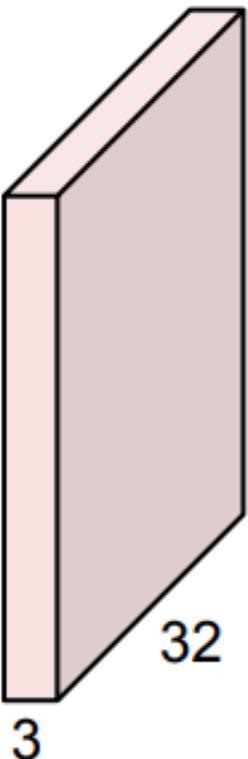
Convolutional Neural Network

Slides are borrowed from Stanford CS231N.

Convolutional Neural Network

Convolution Layer

32x32x3 image



5x5x3 filter

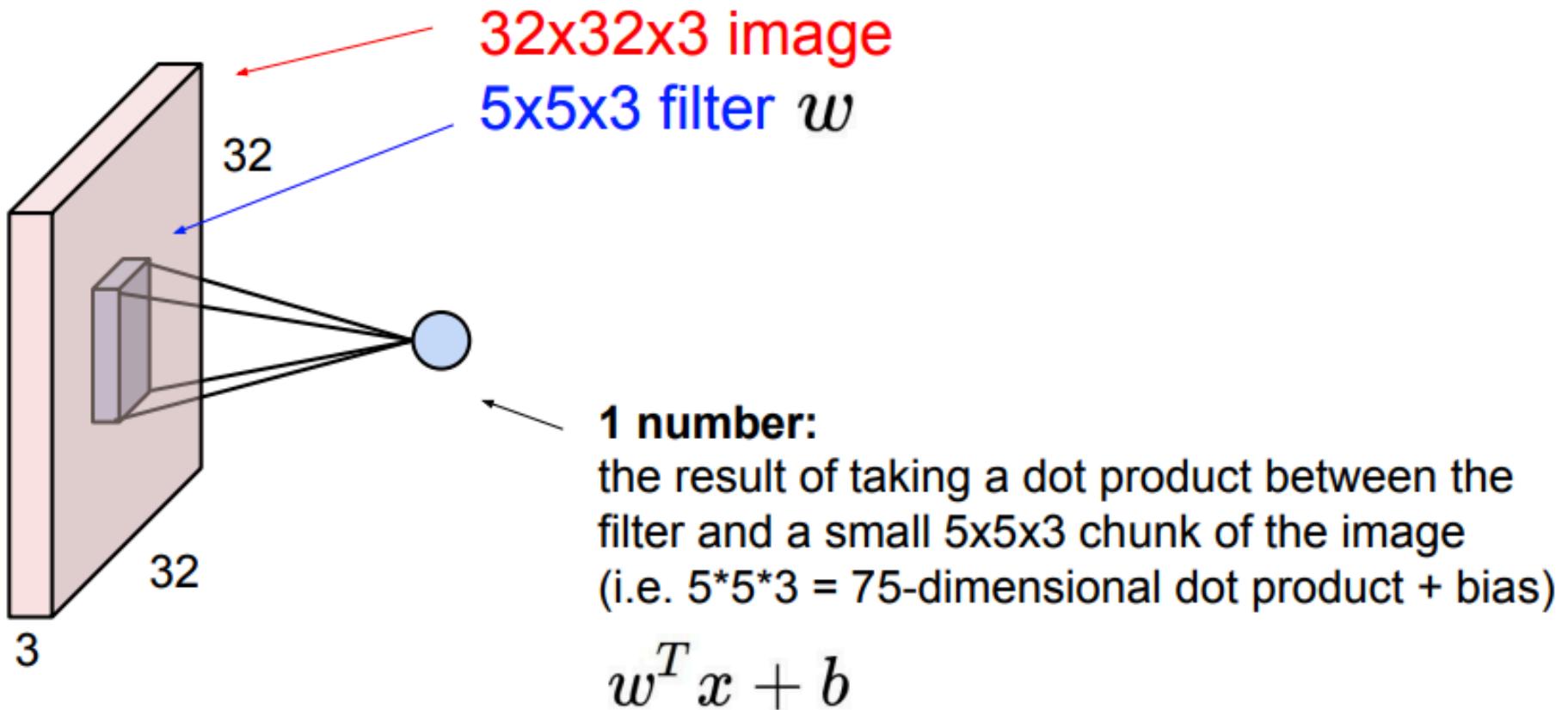


Filters always extend the full depth of the input volume

Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

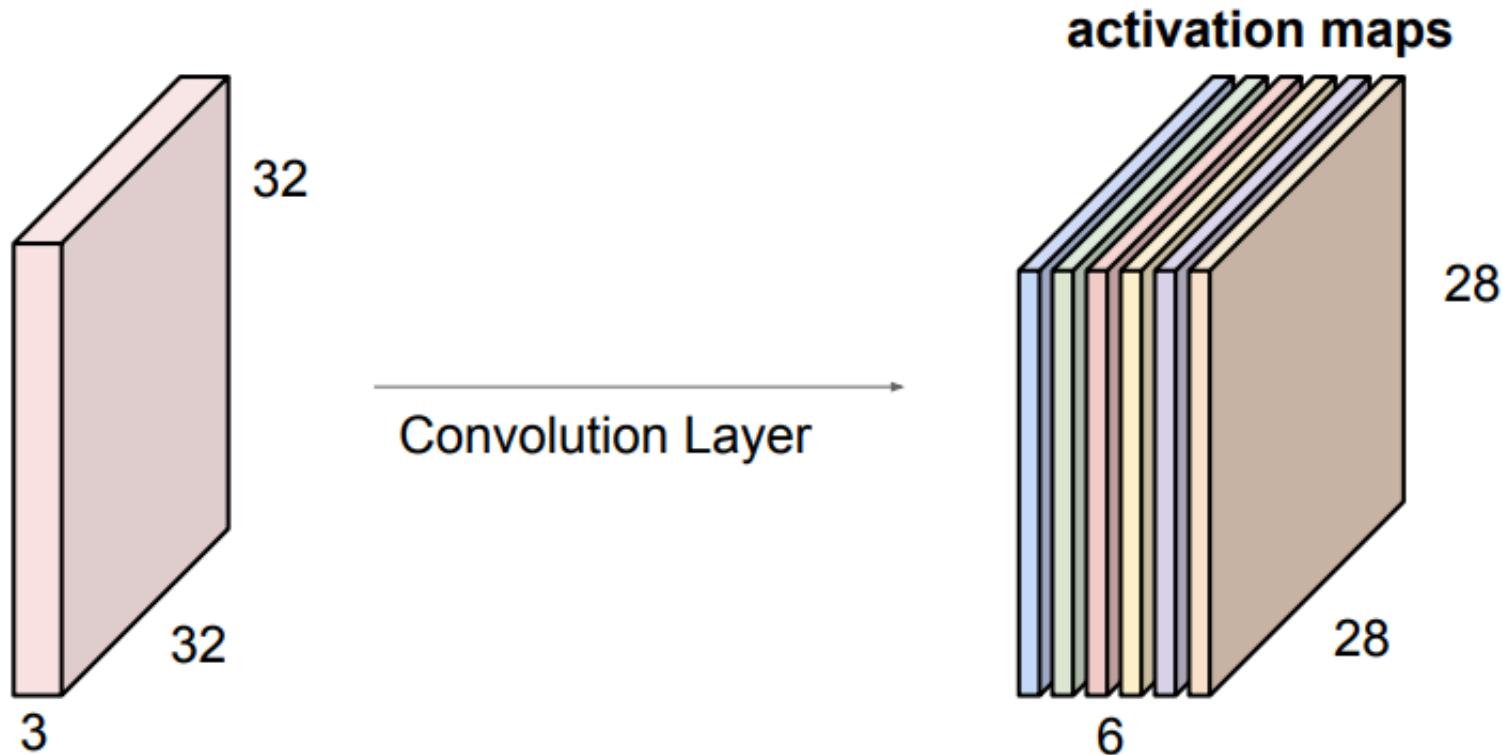
Convolutional Neural Network

Convolution Layer



Convolutional Neural Network

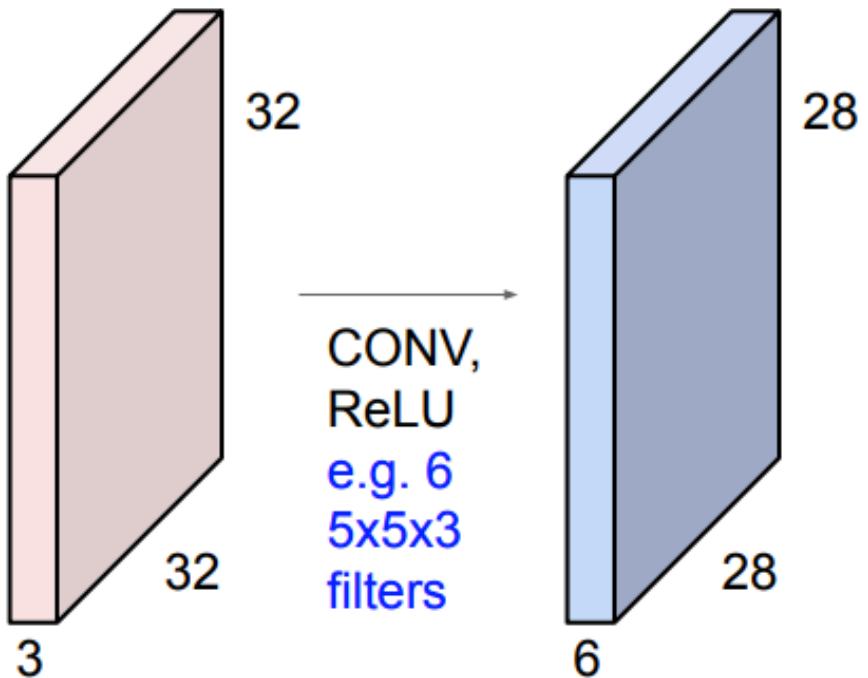
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size 28x28x6!

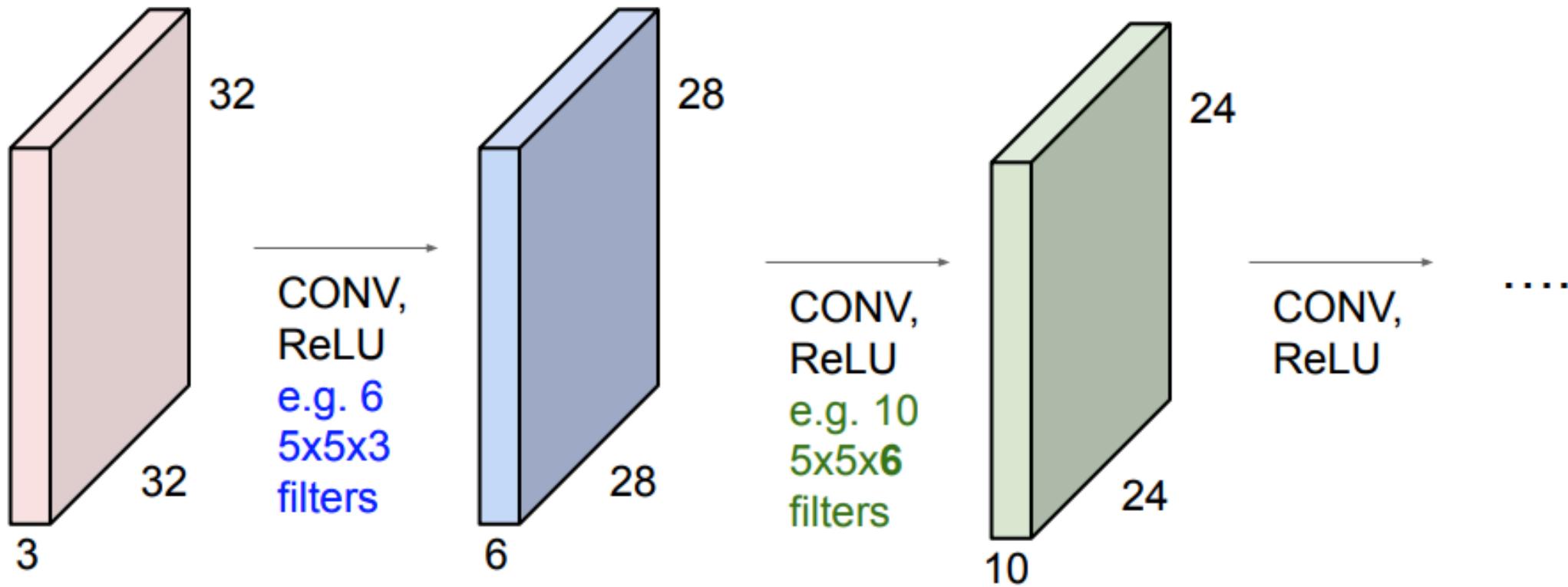
Convolutional Neural Network

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



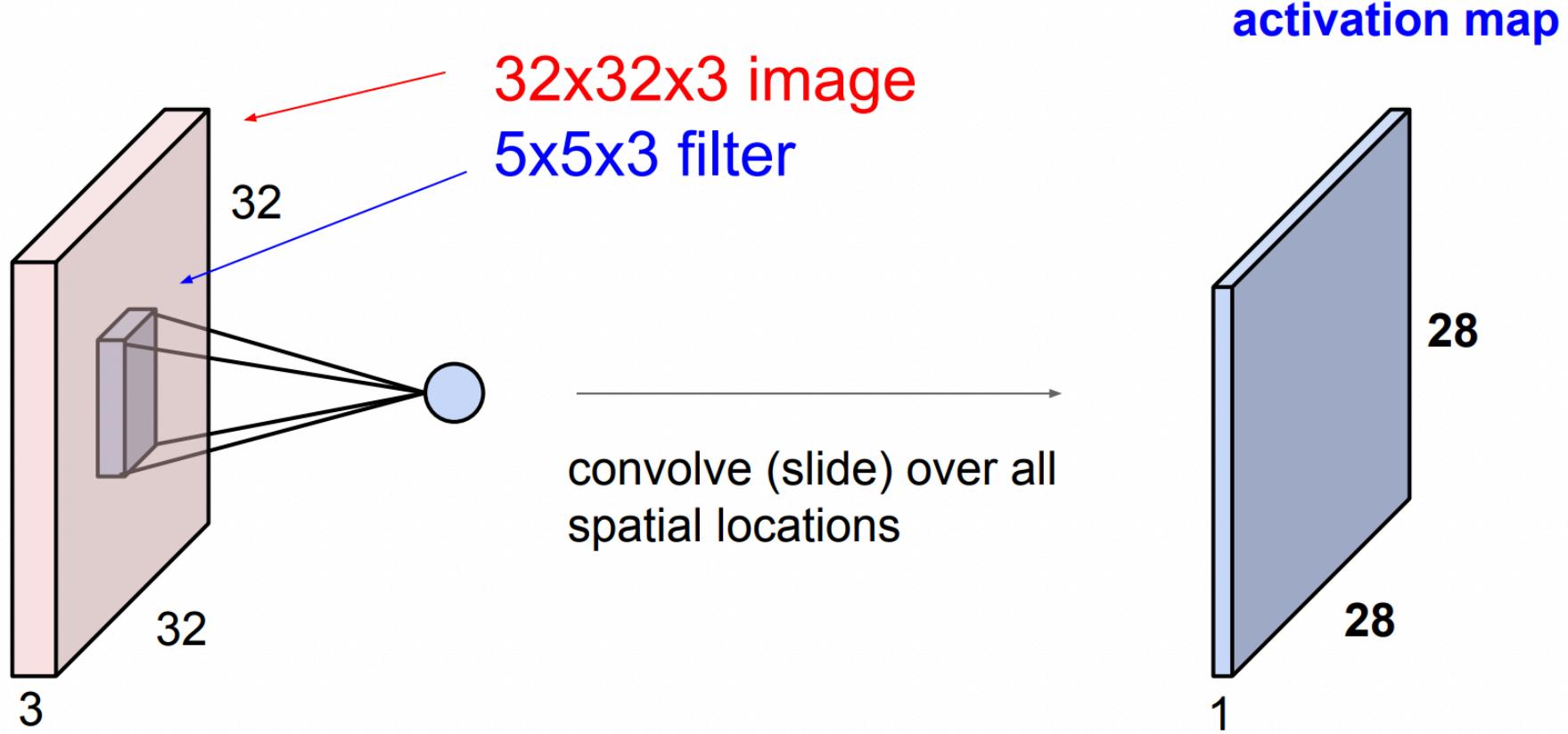
Convolutional Neural Network

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

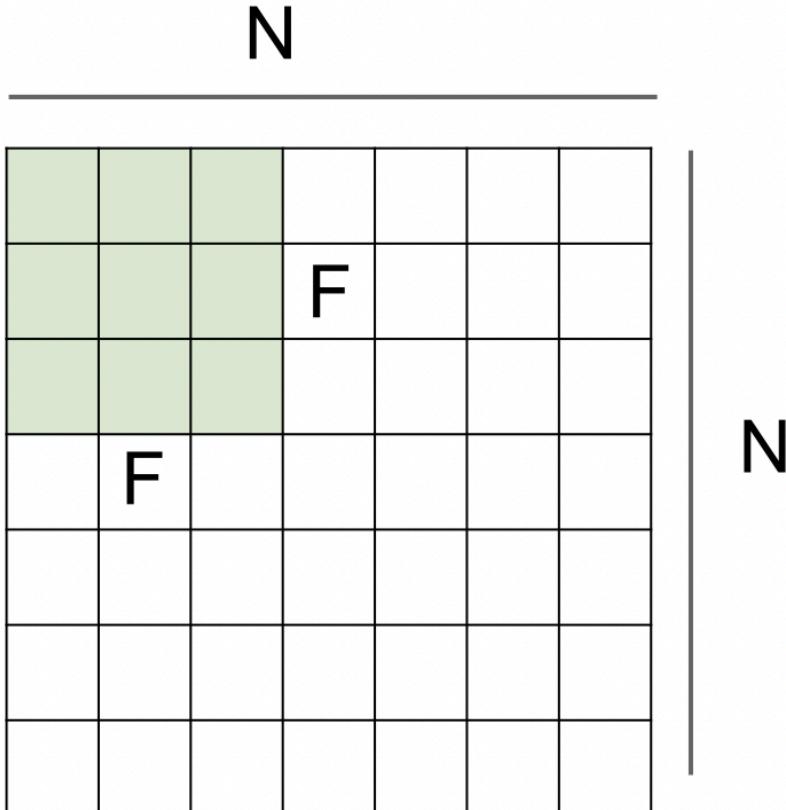


Convolutional Layer

A closer look at spatial dimensions:



Convolutional Layer



Output size:
 $(N - F) / \text{stride} + 1$

e.g. $N = 7$, $F = 3$:
stride 1 => $(7 - 3)/1 + 1 = 5$
stride 2 => $(7 - 3)/2 + 1 = 3$
stride 3 => $(7 - 3)/3 + 1 = 2.33$:\

Padding

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

Padding

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

(recall:)

$$(N + 2P - F) / \text{stride} + 1$$

Padding

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

7x7 output!

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with $(F-1)/2$. (will preserve size spatially)

e.g. $F = 3 \Rightarrow$ zero pad with 1

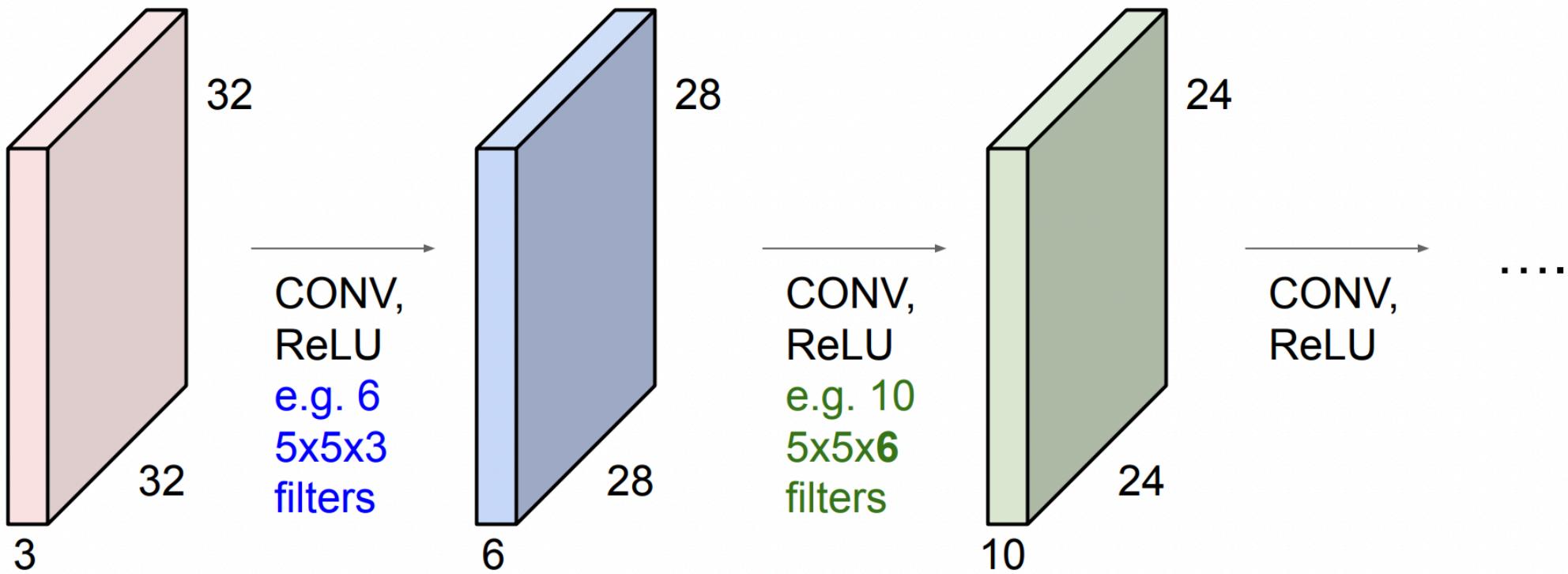
$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Convolutional Layer

Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



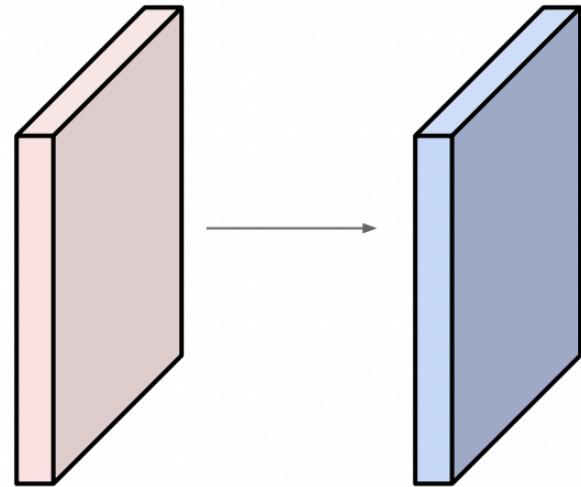
Convolutional Layer

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?

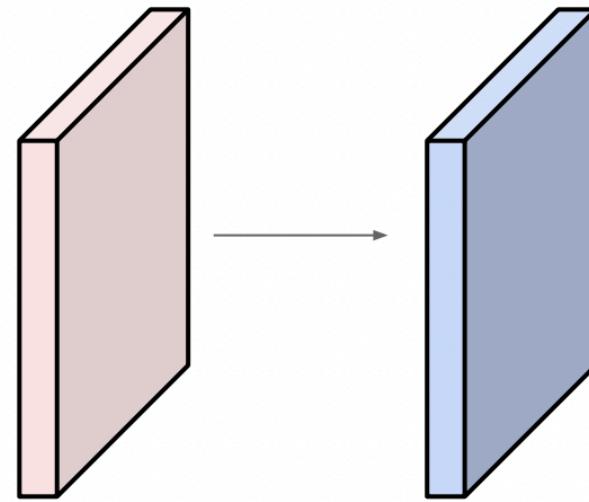


Convolutional Layer

Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride **1**, pad **2**



Output volume size:

$(32+2*2-5)/1+1 = 32$ spatially, so

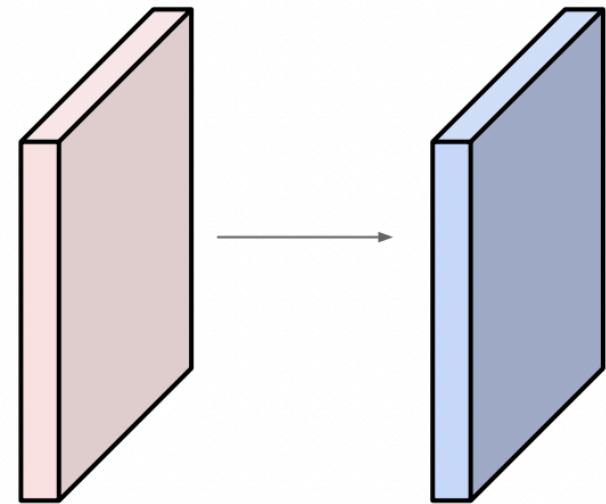
32x32x10

Convolutional Layer

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



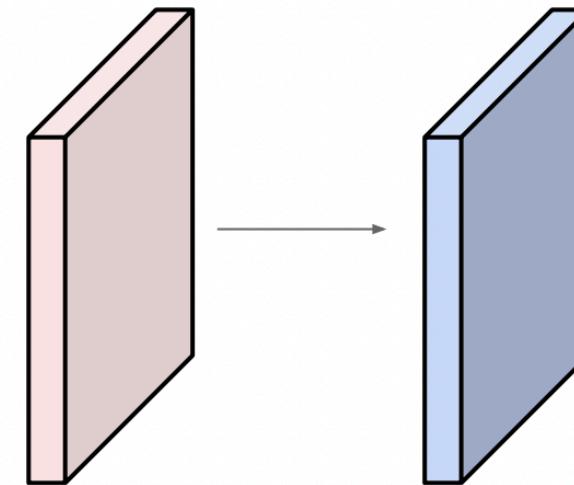
Number of parameters in this layer?

Convolutional Layer

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

each filter has $5*5*3 + 1 = 76$ params (+1 for bias)

$$\Rightarrow 76 * 10 = 760$$

Convolutional Layer

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$

where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Convolutional Layer

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$

where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

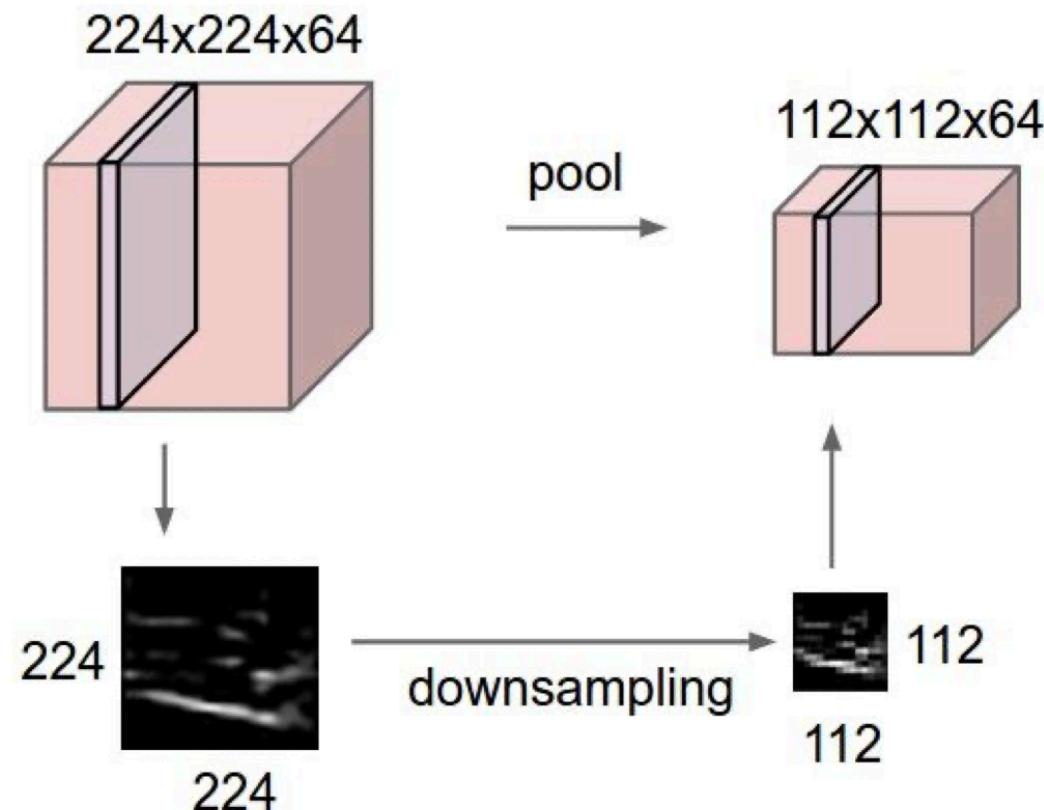
Common settings:

$K = (\text{powers of 2, e.g. } 32, 64, 128, 512)$

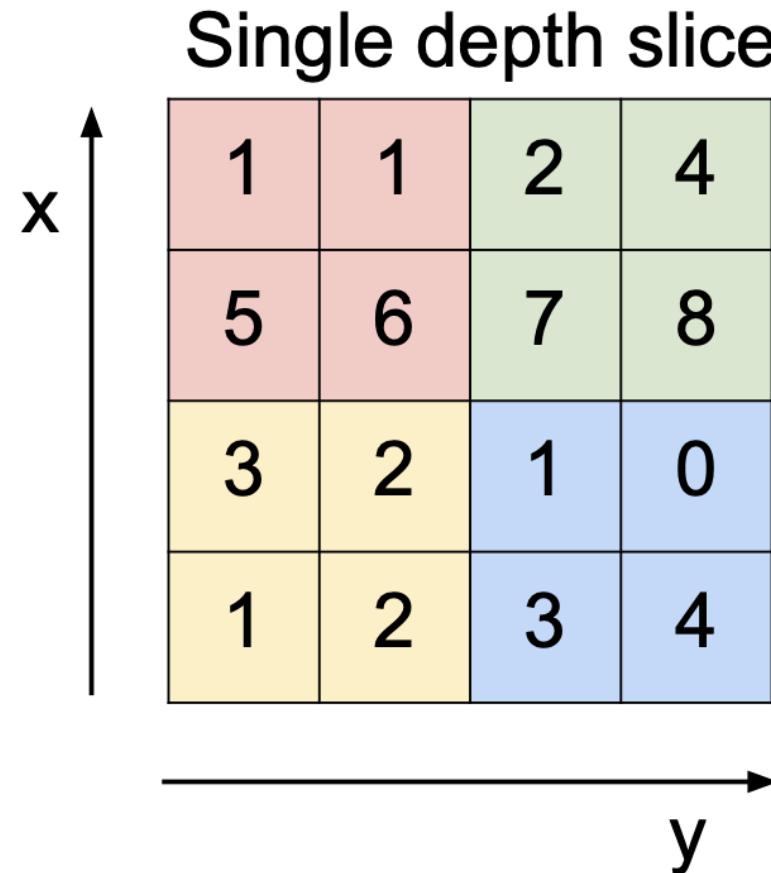
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ? \text{ (whatever fits)}$
- $F = 1, S = 1, P = 0$

Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling



max pool with 2x2 filters
and stride 2

→

6	8
3	4

Pooling

- Average pooling
- Max pooling
- Sum pooling (rarely)

Pooling Layer: Summary

- Let's assume input is $W_1 \times H_1 \times C$
- 2×2 pooling will result in
 - $W_2 = \lceil W_1 / 2 \rceil$
 - $H_2 = \lceil H_1 / 2 \rceil$
- Number of parameters: 0

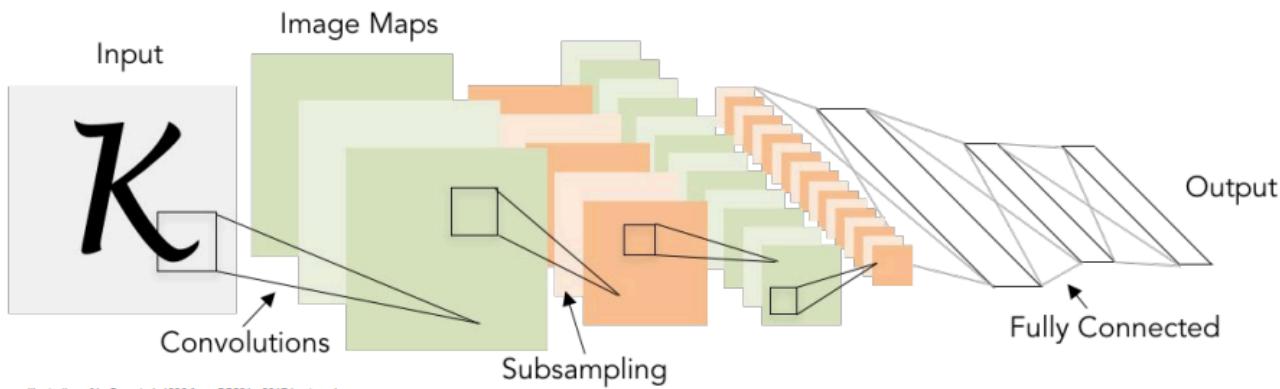
Summary of CNN-based Classification Network

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like

$[(\text{CONV-RELU})^*N - \text{POOL?}]^*M - (\text{FC-RELU})^*K, \text{SOFTMAX}$

where N is usually up to ~5, M is large, $0 \leq K \leq 2$.

- but recent advances such as ResNet/GoogLeNet have challenged this paradigm



A Comparison between MLP and CNN

- Input: $W_1 \times H_1 \times C$
- Output: $W_2 \times H_2 \times K$
- Fully connected layer (FC)
 - Densely connected
 - Parameters: $W_1 W_2 H_1 H_2 C K$
- CNN layer
 - Filter size: F
 - Parameters: $F^2 C K$

Toeplitz Matrix for 1D convolution

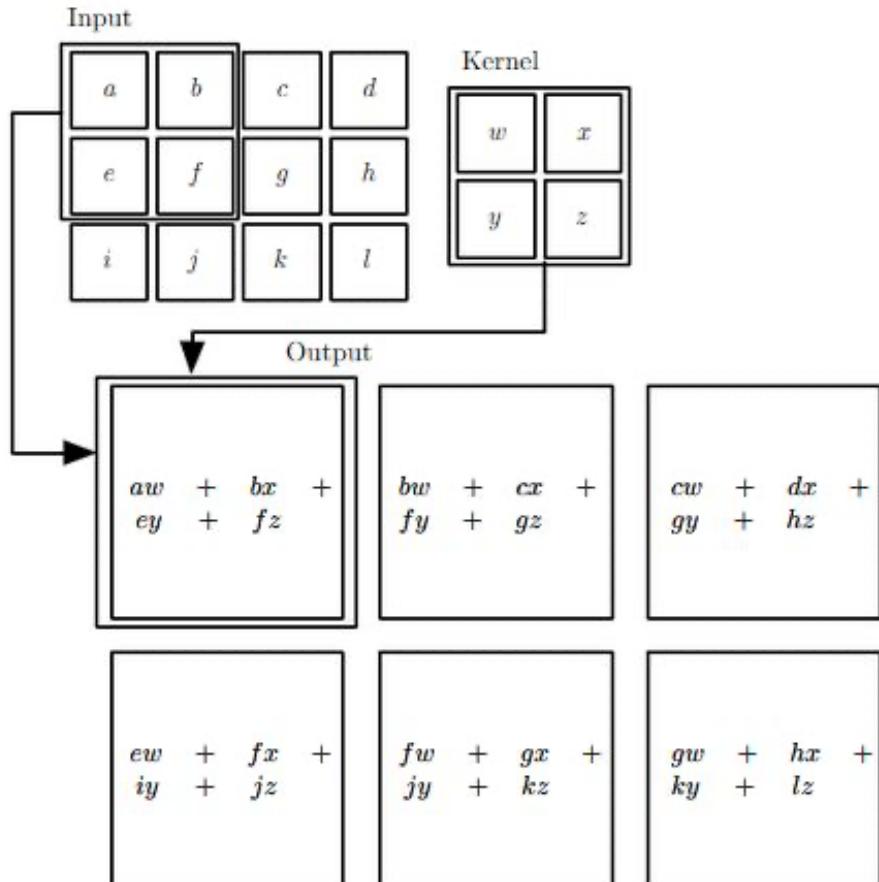
For 1D convolution, $h \in \mathbb{R}^m$, $x \in \mathbb{R}^n$, the convolution operation $y = h * m$ can be constructed as a matrix multiplication:

$$y = h * x = \begin{bmatrix} h_1 & 0 & \cdots & 0 & 0 \\ h_2 & h_1 & & \vdots & \vdots \\ h_3 & h_2 & \cdots & 0 & 0 \\ \vdots & h_3 & \cdots & h_1 & 0 \\ h_{m-1} & \vdots & \ddots & h_2 & h_1 \\ h_m & h_{m-1} & & \vdots & h_2 \\ 0 & h_m & \ddots & h_{m-2} & \vdots \\ 0 & 0 & \cdots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & & h_m & h_{m-1} \\ 0 & 0 & 0 & \cdots & h_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

- A sparse matrix
- A lot of equal constraints

a **Toeplitz** matrix or diagonal-constant matrix, named after Otto Toeplitz, is a matrix in which each descending diagonal from left to right is constant.

Matrix for 2D convolution

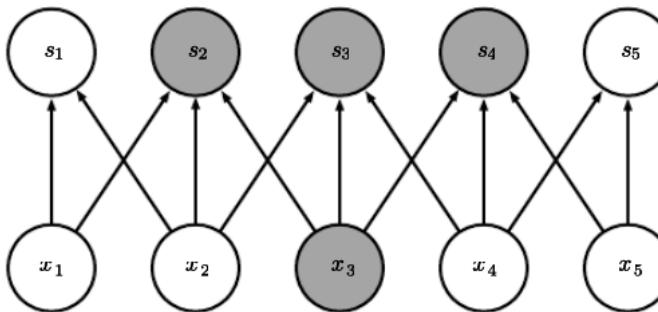


A very sparse
doubly block circulant matrix

Sparse Connectivity

- Convolutional networks, however, typically have sparse interactions (also referred to as sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input.

CNN
1D Conv with filter size=3



FC

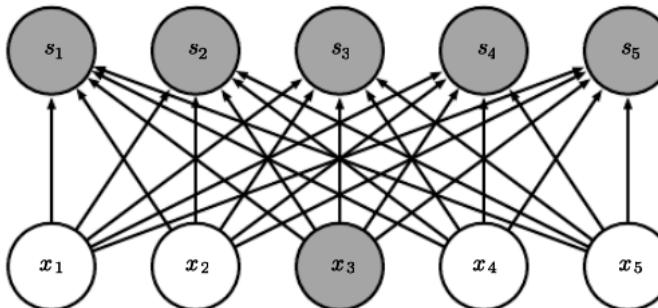
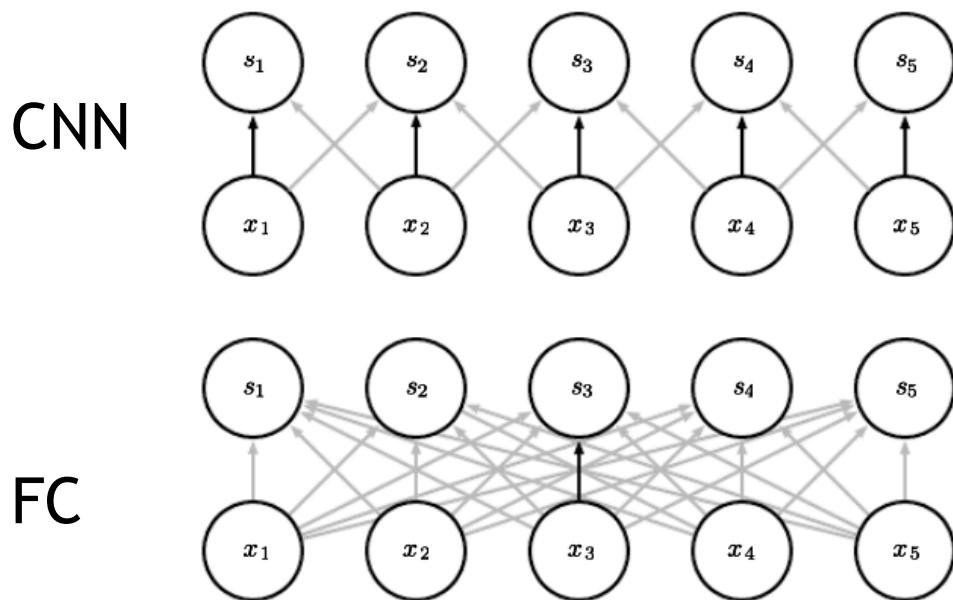


Figure from Deep learning (MIT press, 2016).

Parameter Sharing

- refers to using the same parameter for more than one function in a model.



Parameter sharing. Black arrows indicate the connections that use a particular parameter in two different models.

Why CNN?

- Which is more expressive?
 - CNN
 - FC

Why CNN?

- Which is more expressive?
 - CNN
 - FC is a super set of CNN (without sparse and parameter sharing constraints.)
- What is the problem of FC/MLP?

Why CNN?

- Which is more expressive?
 - CNN
 - FC is a super set of CNN (without sparse and parameter sharing constraints.)
- What is the problem of FC/MLP?
 - FC needs too many parameters.
 - What else?

Image Perturbations



Applying a small translation

Image Perturbations



Applying a small rotation



The behavior of Fully Connect Layer

- FC will change dramatically even if you just shift one row or one column. Same happens for a 5° rotation.
- A huge problem during network training or optimization!
- What about CNN?

Equivariance

General definition of equivariance:

$$S_A[\phi(X)] = \phi(T_A(X))$$

Here A is an operation, T_A and S_A are their transformation function in the space of X and $\phi(X)$.

Equivariance

General definition of equivariance:

$$S_A[\phi(X)] = \phi(T_A(X))$$

Here A is an operation, T_A and S_A are their transformation function in the space of X and $\phi(X)$.

Invariance is a special case of equivariance,
When $S_A = I$, $\phi(X) = \phi(T_A(X))$.

Parameter Sharing = Equivariance with Translation

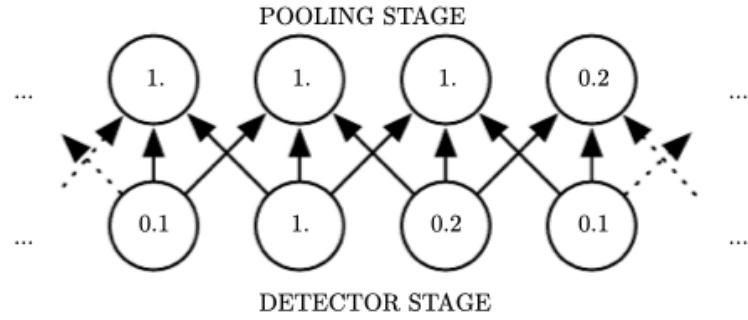
- Ignoring the values at the boundary, 2D Conv is equivariant with translation.
- We use the same way to process all local regions due to parameter sharing.
- For example, when processing images, it is useful to detect edges in the first layer of a convolutional network. **The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image.**

CNN

- Convolution is not naturally equivariant with some other transformations
 - changes in the scale
 - rotation of an image.
- Other mechanisms are necessary for handling these kinds of transformations.

Pooling

- Invariance to local translation can be a useful property if we care more about whether some feature is present than exactly where it is.



Max pooling introduces invariance.

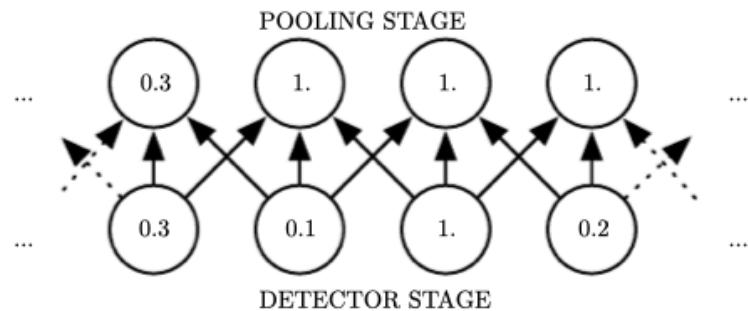
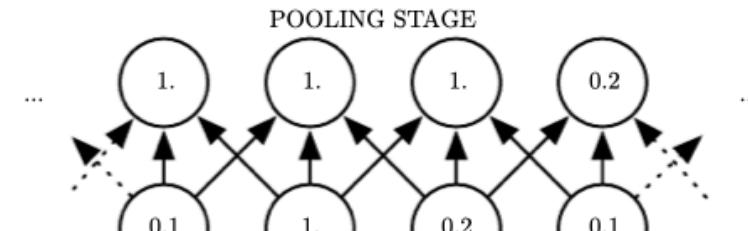


Figure from Deep learning (MIT press, 2016).

Pooling

- Applying pooling induces invariance to small translations and rotations.



Max pooling introduces invariance.

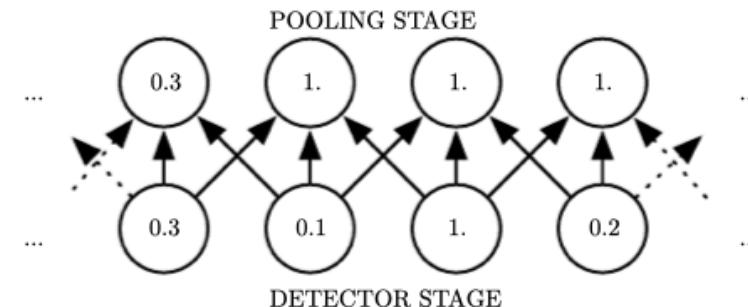


Figure from Deep learning (MIT press, 2016).

Convolution and Pooling as a Prior

- CNN stacks Conv layer and pooling layer.
- Conv layer + pooling layer is invariant with small translation and rotation.
- Thus CNN is much easier to optimize than a FC for an image.
- Loss landscape:



CNN Training

To Train a CNN

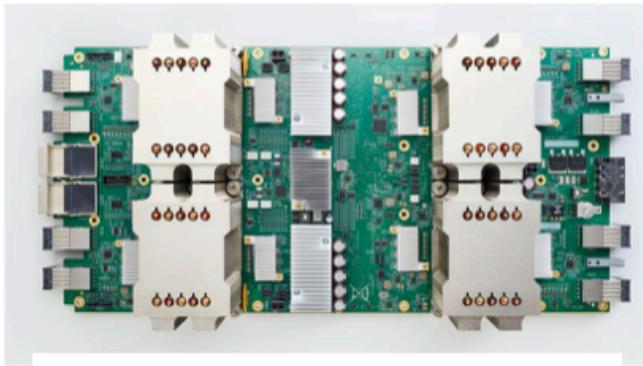
Mini-batch SGD

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph
(network), get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

To Train a CNN

Hardware + Software



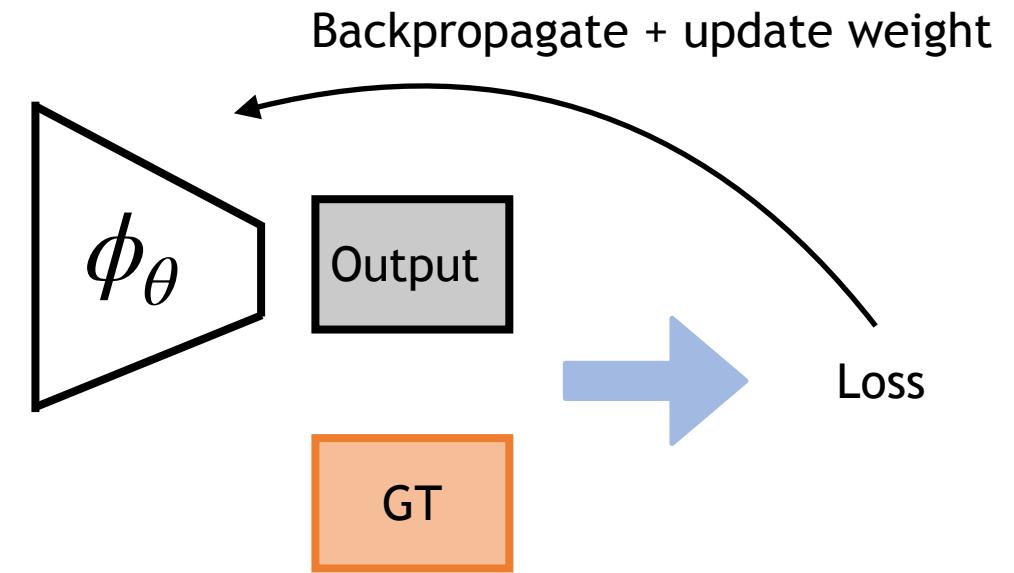
PyTorch



TensorFlow

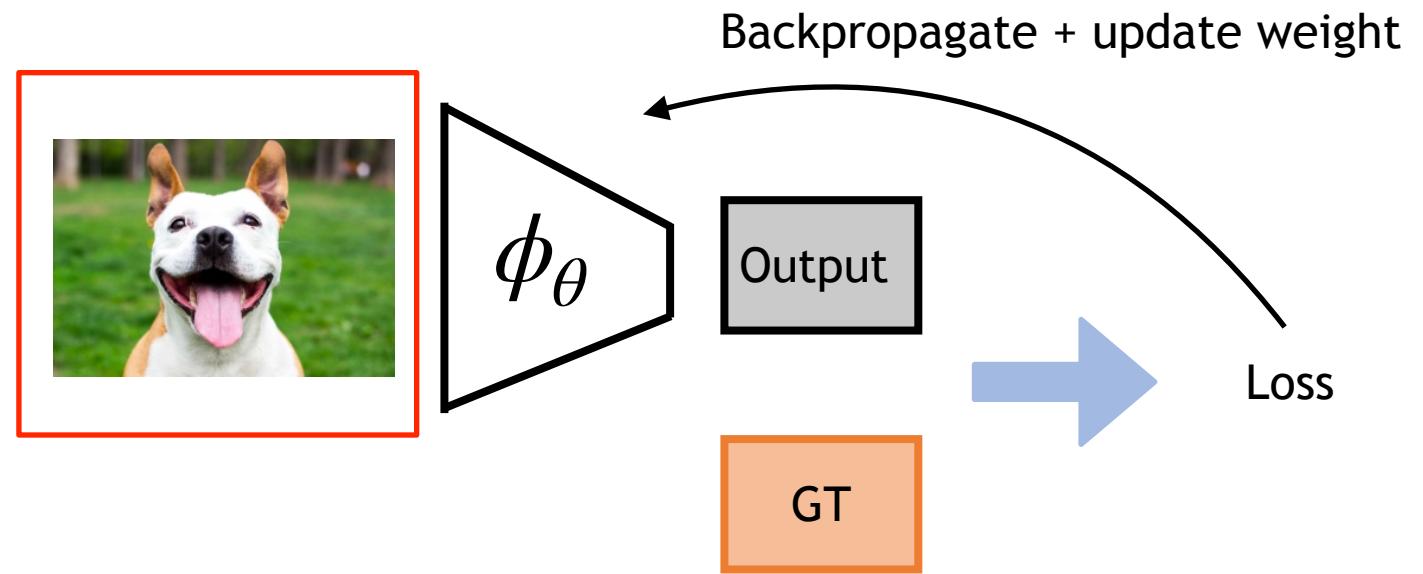
Outline

- Data preparation
- Weight Initialization
- Set a loss function
- Start optimization
 - optimizer?
 - learning rate?

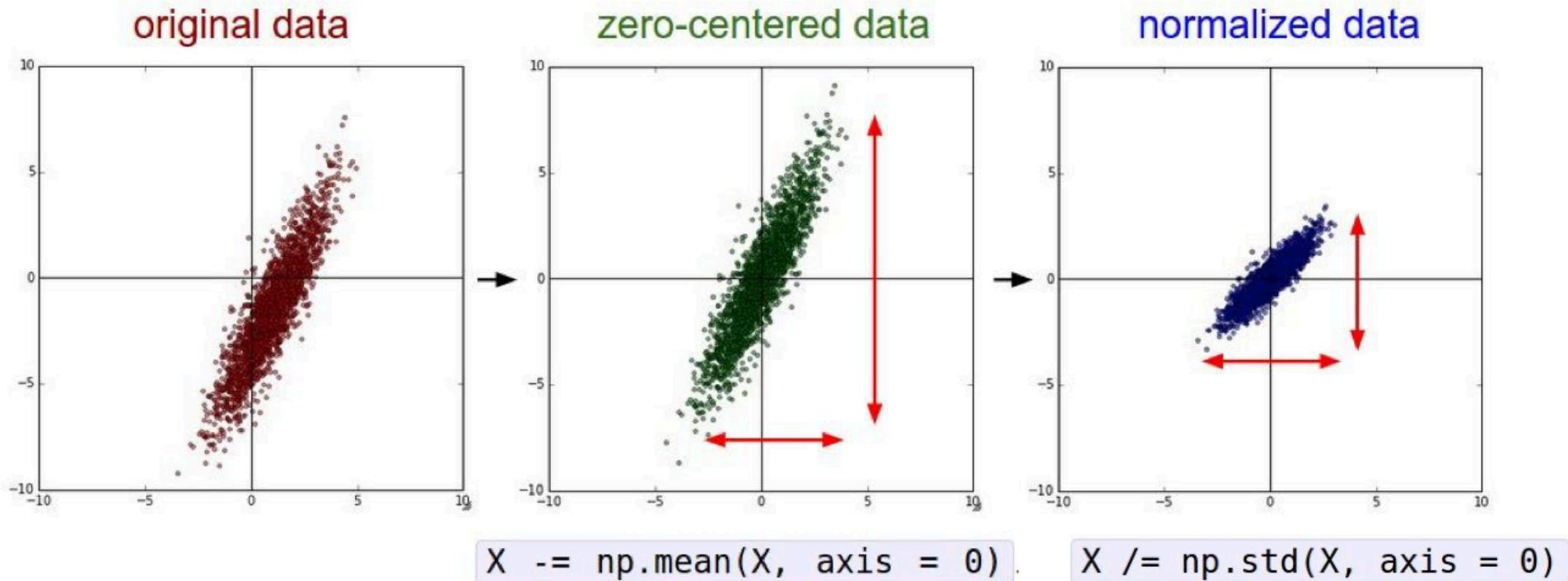


Outline

- Data preparation
- Weight Initialization
- Set a loss function
- Start optimization
 - optimizer?
 - learning rate?



Data Preprocessing

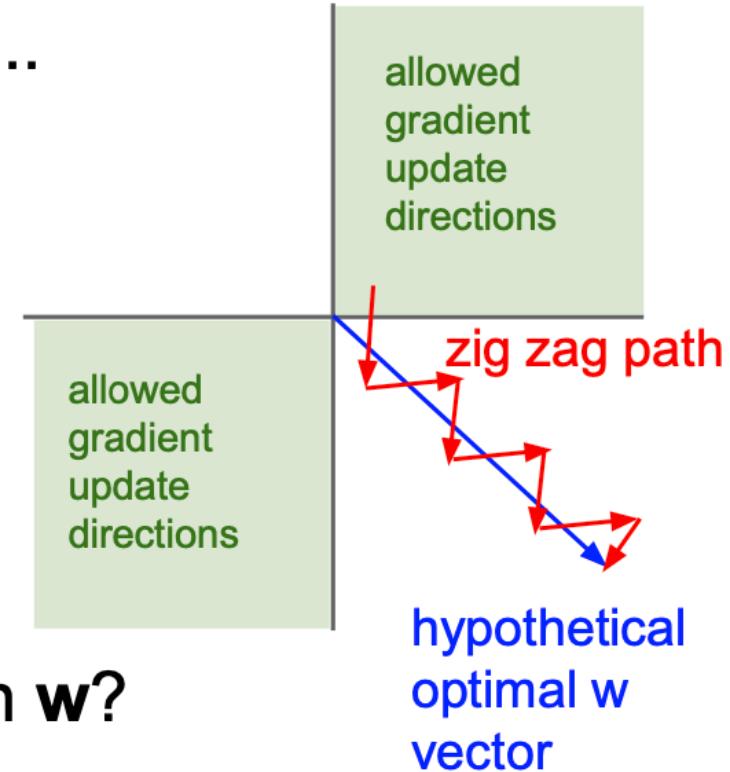


(Assume X [NxD] is data matrix,
each example in a row)

Data Preprocessing

Remember: Consider what happens when the input to a neuron is always positive...

$$f \left(\sum_i w_i x_i + b \right)$$



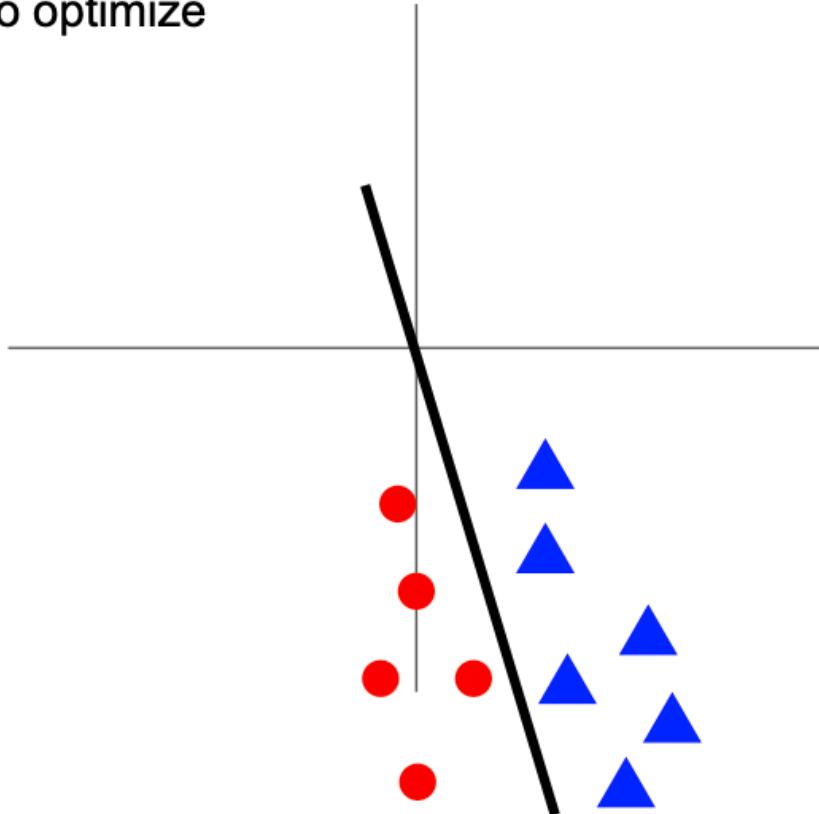
What can we say about the gradients on w ?

Always all positive or all negative :(

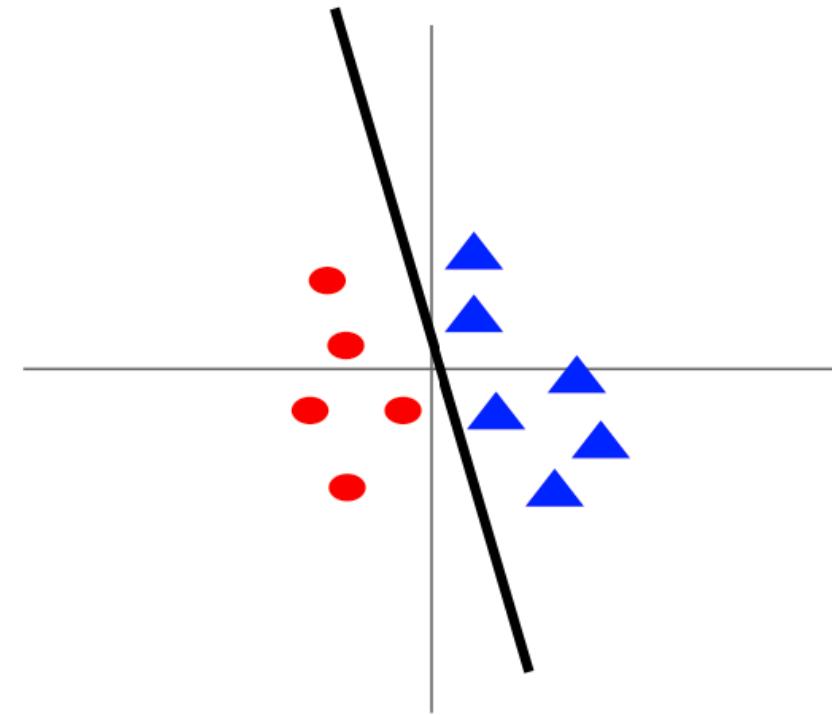
(this is also why you want zero-mean data!)

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



Summary of Data Preprocessing

e.g. consider CIFAR-10 example with [32,32,3] images

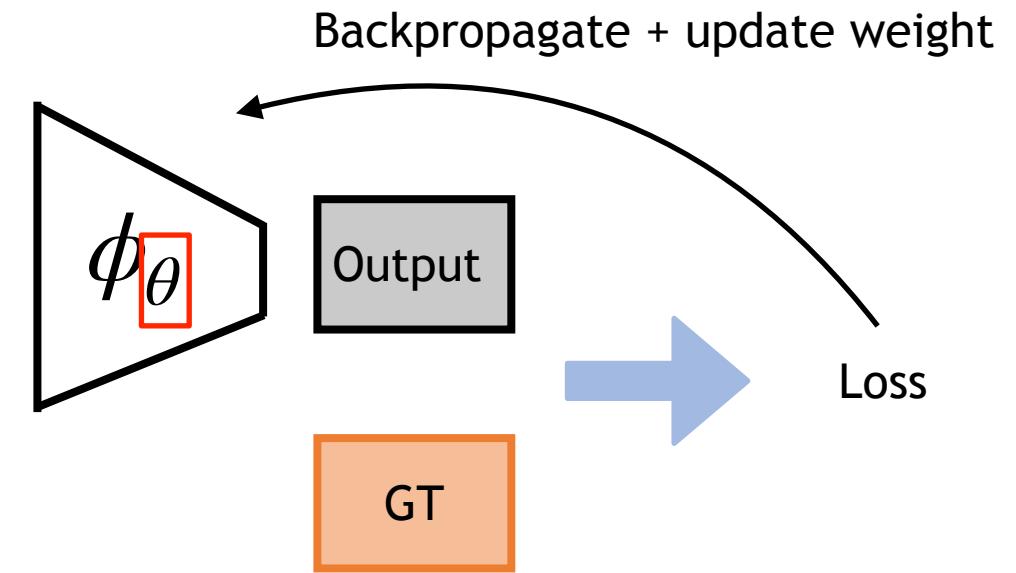
- Subtract per-channel mean (e.g. VGGNet)
(mean along each channel = 3 numbers)
- Subtract per-channel mean and
Divide by per-channel std (e.g. ResNet)
(mean along each channel = 3 numbers)

Outline

- Data preparation
- **Weight Initialization**
- Set a loss function
- Start optimization
 - optimizer?
 - learning rate?

Outline

- Data preparation
- Weight Initialization
- Set a loss function
- Start optimization
 - optimizer?
 - learning rate?



Weight Initialization

- First idea: **Small random numbers**
(gaussian with zero mean and 1e-2 standard deviation)

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

Works ~okay for small networks, but problems with deeper networks.

Weight Initialization

```
dims = [4096] * 7      Forward pass for a 6-layer  
hs = []                  net with hidden size 4096  
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)
```

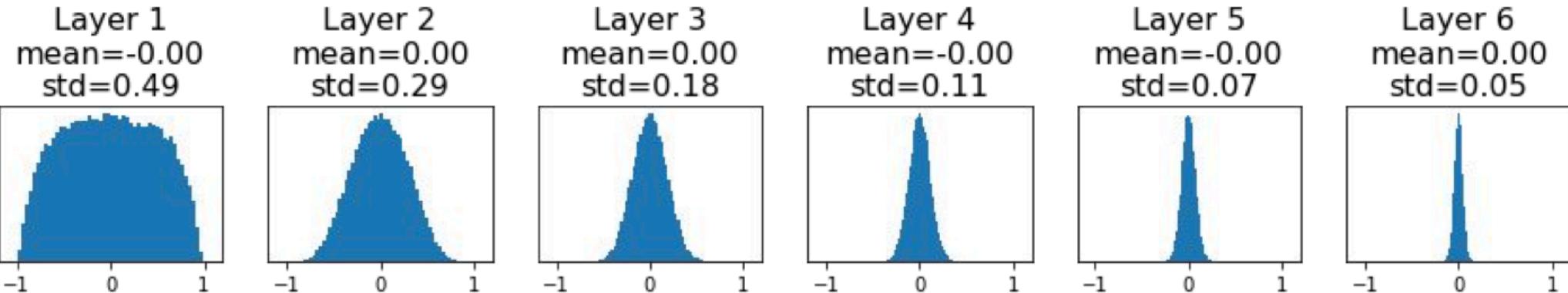
What will happen to the activations for the last layer?

Weight Initialization: Activation Statistics

```
dims = [4096] * 7      Forward pass for a 6-layer  
hs = []                net with hidden size 4096  
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)
```

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?



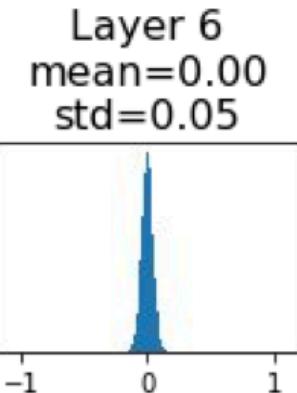
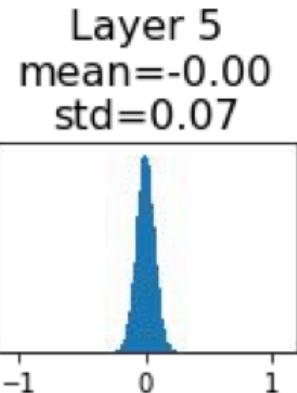
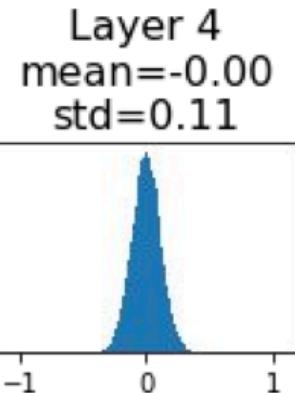
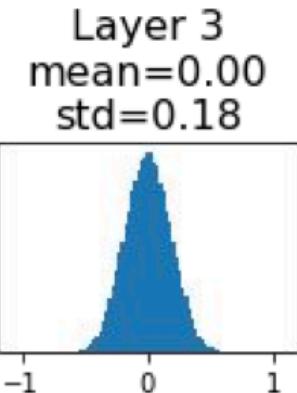
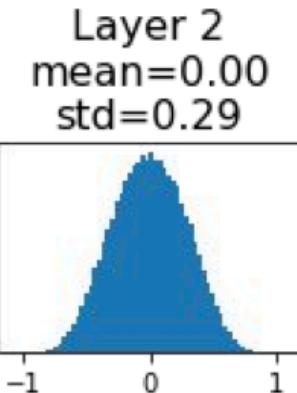
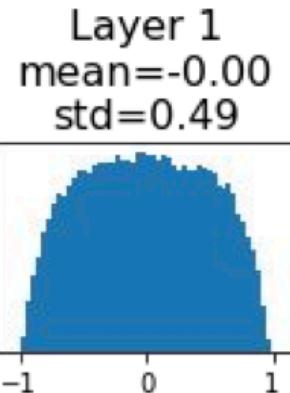
Weight Initialization: Activation Statistics

```
dims = [4096] * 7      Forward pass for a 6-layer  
hs = []                net with hidden size 4096  
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)
```

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?

A: All zero, no learning =(



Weight Initialization: Activation Statistics

```
dims = [4096] * 7    Increase std of initial  
hs = []                weights from 0.01 to 0.05  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = 0.05 * np.random.randn(Din, Dout)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

What will happen to the activations for the last layer?

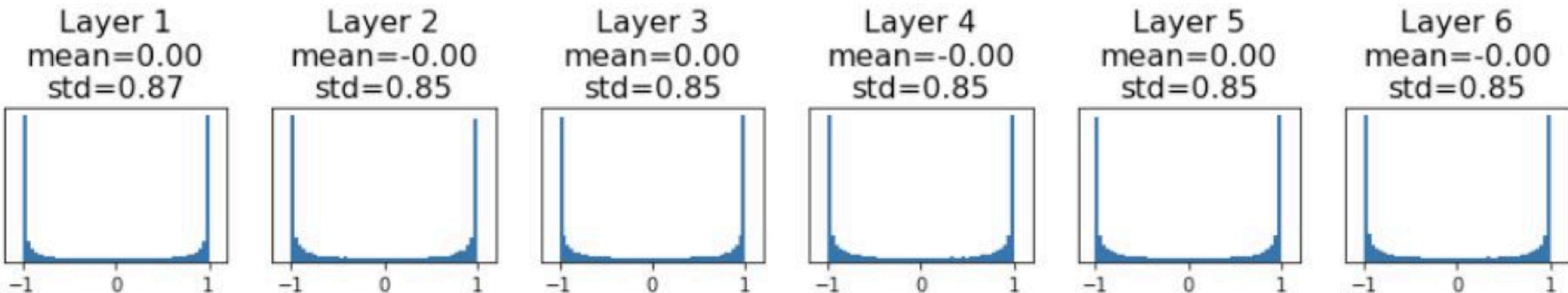
Weight Initialization: Activation Statistics

```
dims = [4096] * 7      Increase std of initial  
hs = []                  weights from 0.01 to 0.05  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = 0.05 * np.random.randn(Din, Dout)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

All activations saturate

Q: What do the gradients look like?

A: Local gradients all zero, no learning =(



Weight Initialization: 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:
 $\text{std} = 1/\sqrt{D_{\text{in}}}$

Weight Initialization: Xavier Initialization

```
dims = [4096] * 7          "Xavier" initialization:  
hs = []                      std = 1/sqrt(Din)  
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)
```

“Just right”: Activations are nicely scaled for all layers!

For conv layers, Din is filter_size² * input_channels

Let: $y = x_1 w_1 + x_2 w_2 + \dots + x_{Din} w_{Din}$

Assume: $\text{Var}(x_1) = \text{Var}(x_2) = \dots = \text{Var}(x_{Din})$

We want: $\text{Var}(y) = \text{Var}(x_i)$

$$\begin{aligned}\text{Var}(y) &= \text{Var}(x_1 w_1 + x_2 w_2 + \dots + x_{Din} w_{Din}) \\ &= \text{Din} \text{Var}(x_i w_i) \\ &= \text{Din} \text{Var}(x_i) \text{Var}(w_i)\end{aligned}$$

[Assume all x_i, w_i are iid]

So, $\text{Var}(y) = \text{Var}(x_i)$ only when $\text{Var}(w_i) = 1/\text{Din}$

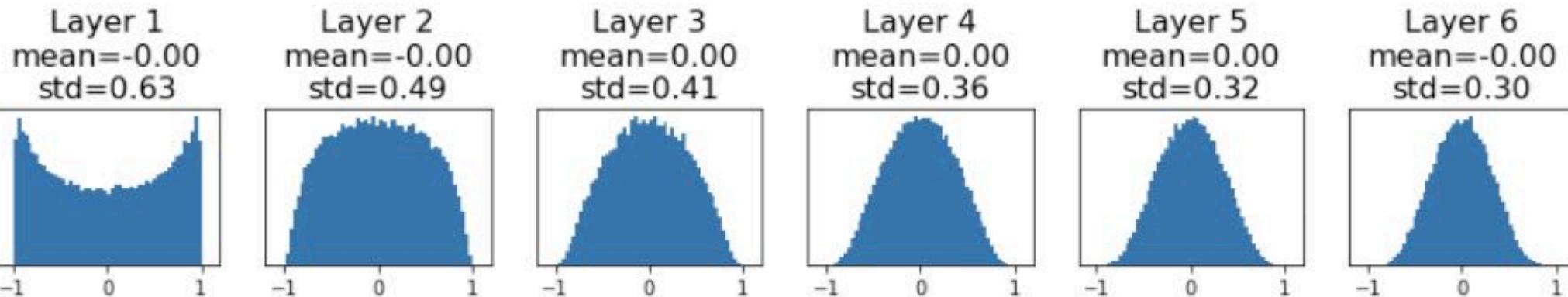
Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

Slide credit: Stanford CS231N

Weight Initialization: Xavier Initialization

```
dims = [4096] * 7          "Xavier" initialization:  
hs = []                      std = 1/sqrt(Din)  
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)
```

“Just right”: Activations are nicely scaled for all layers!



Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

Weight Initialization: Xavier Initialization

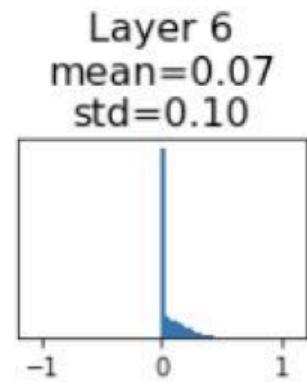
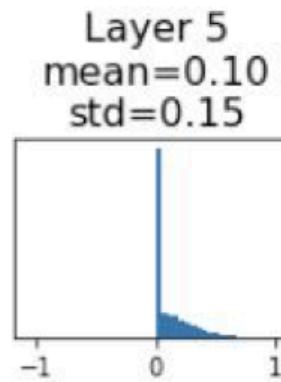
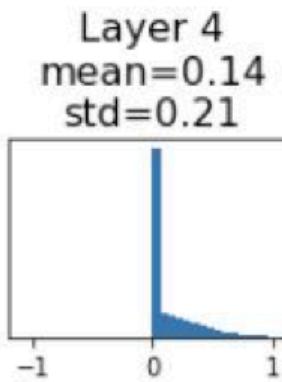
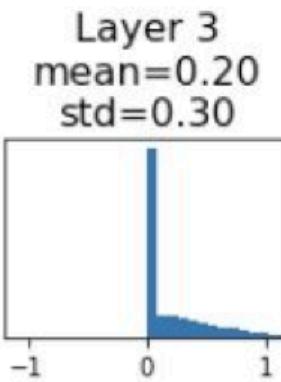
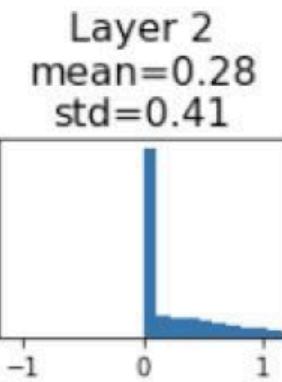
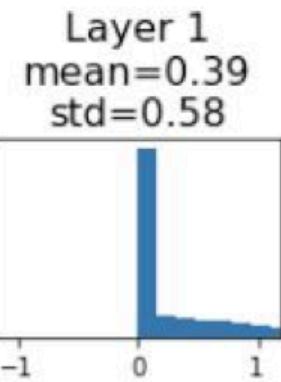
```
dims = [4096] * 7      Change from tanh to ReLU
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Weight Initialization: Xavier Initialization

```
dims = [4096] * 7      Change from tanh to ReLU
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.maximum(0, x.dot(W))
    hs.append(x)
```

Xavier assumes zero centered activation function

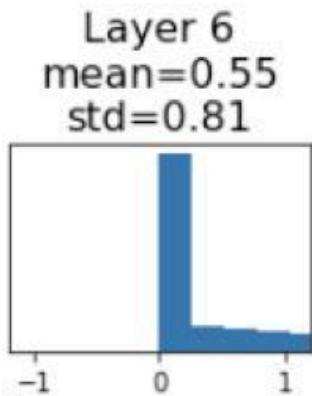
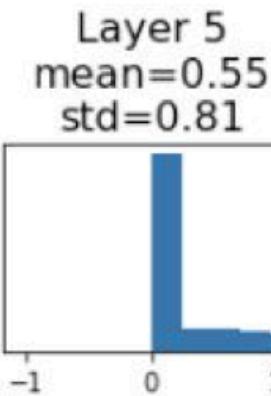
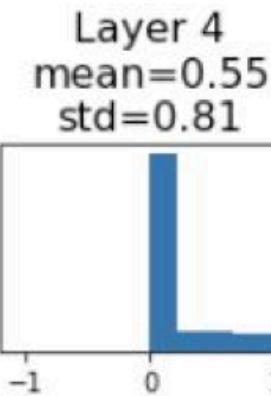
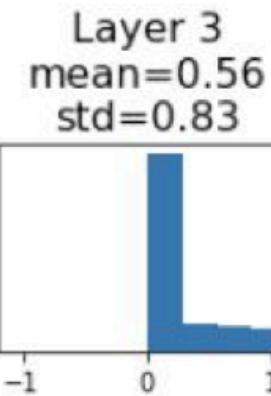
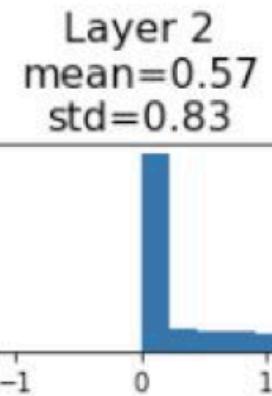
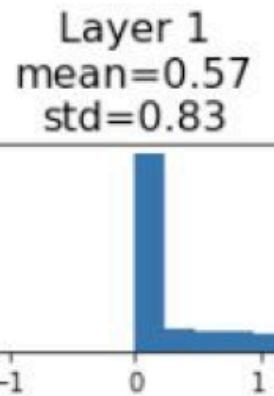
Activations collapse to zero again, no learning =(



Weight Initialization: He Initialization

```
dims = [4096] * 7  ReLU correction: std = sqrt(2 / Din)
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(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

“Just right”: Activations are nicely scaled for all layers!



He et al, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, ICCV 2015

Slide credit: Stanford CS231N

Initialization is still an Active Research Area

Understanding the difficulty of training deep feedforward neural networks

by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

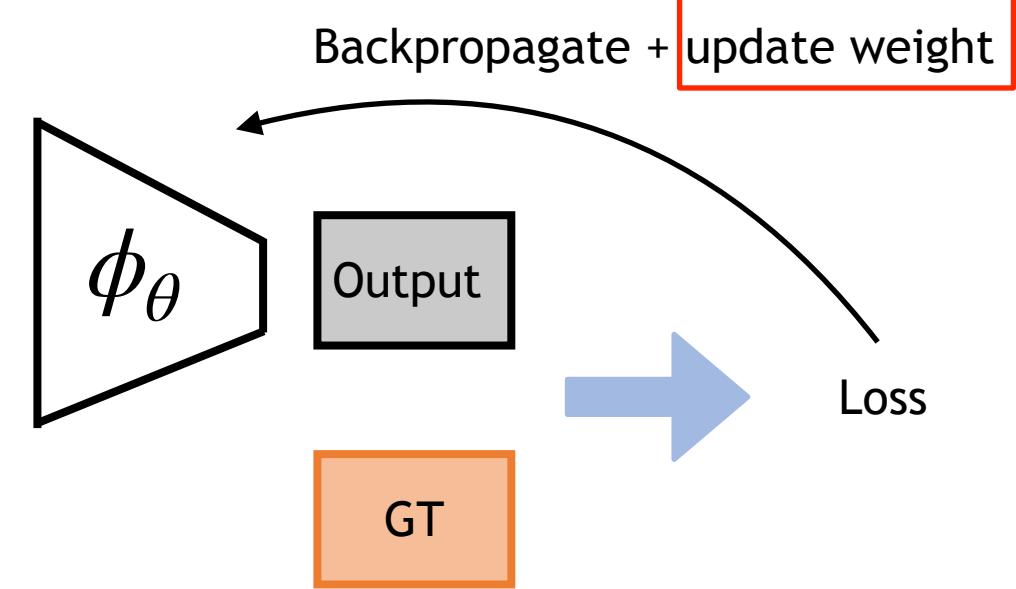
All you need is a good init, Mishkin and Matas, 2015

Fixup Initialization: Residual Learning Without Normalization, Zhang et al, 2019

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, Frankle and Carbin, 2019

Outline

- Data preparation
- Weight Initialization
- Set a loss function
- Start optimization
 - optimizer?
 - learning rate?



SGD

Update rule:

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

```
while True:  
    dx = compute_gradient(x)  
    x -= learning_rate * dx
```

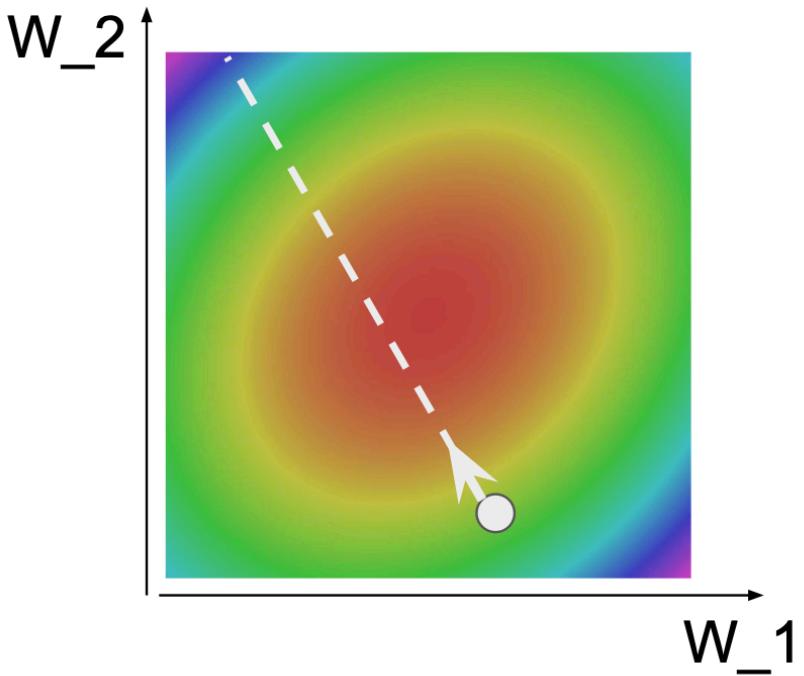
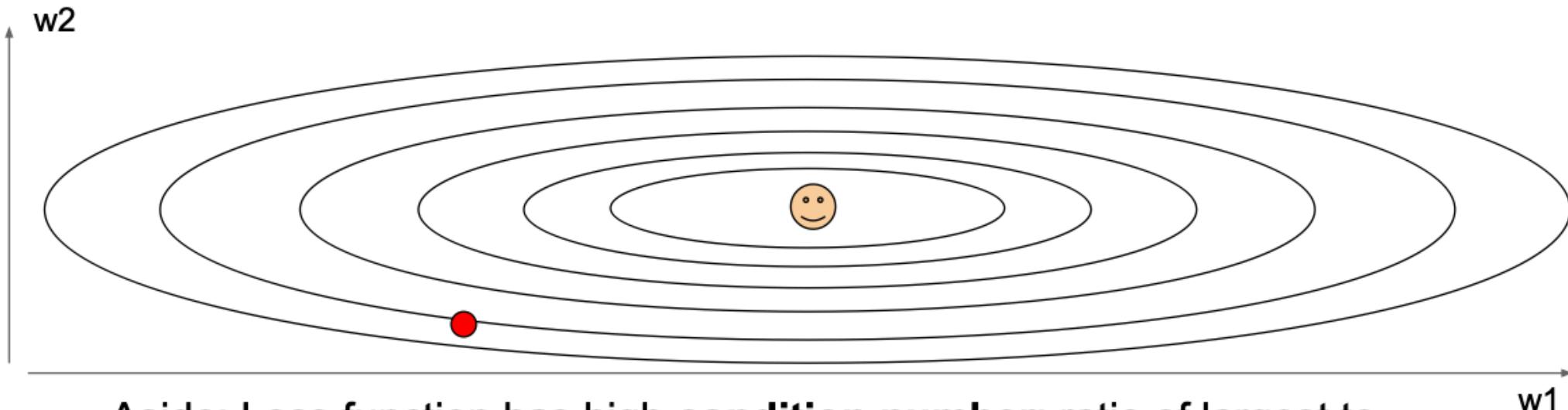


Figure credit: Stanford CS231N

Problems with SGD: #1

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?



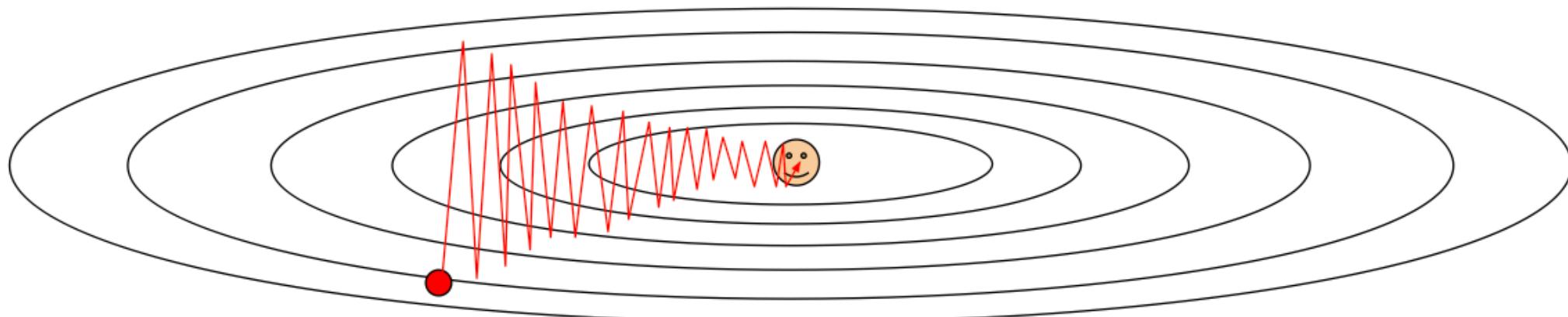
Aside: Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Problems with SGD: #1

What if loss changes quickly in one direction and slowly in another?

What does gradient descent do?

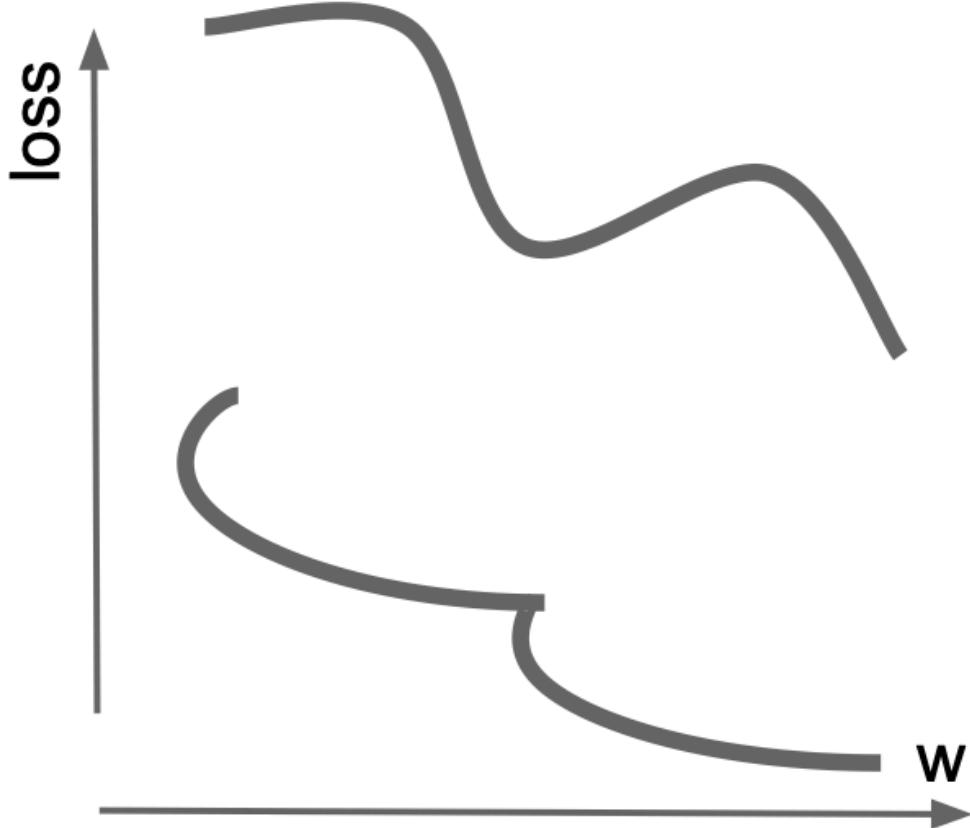
Very slow progress along shallow dimension, jitter along steep direction



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Problems with SGD: #2

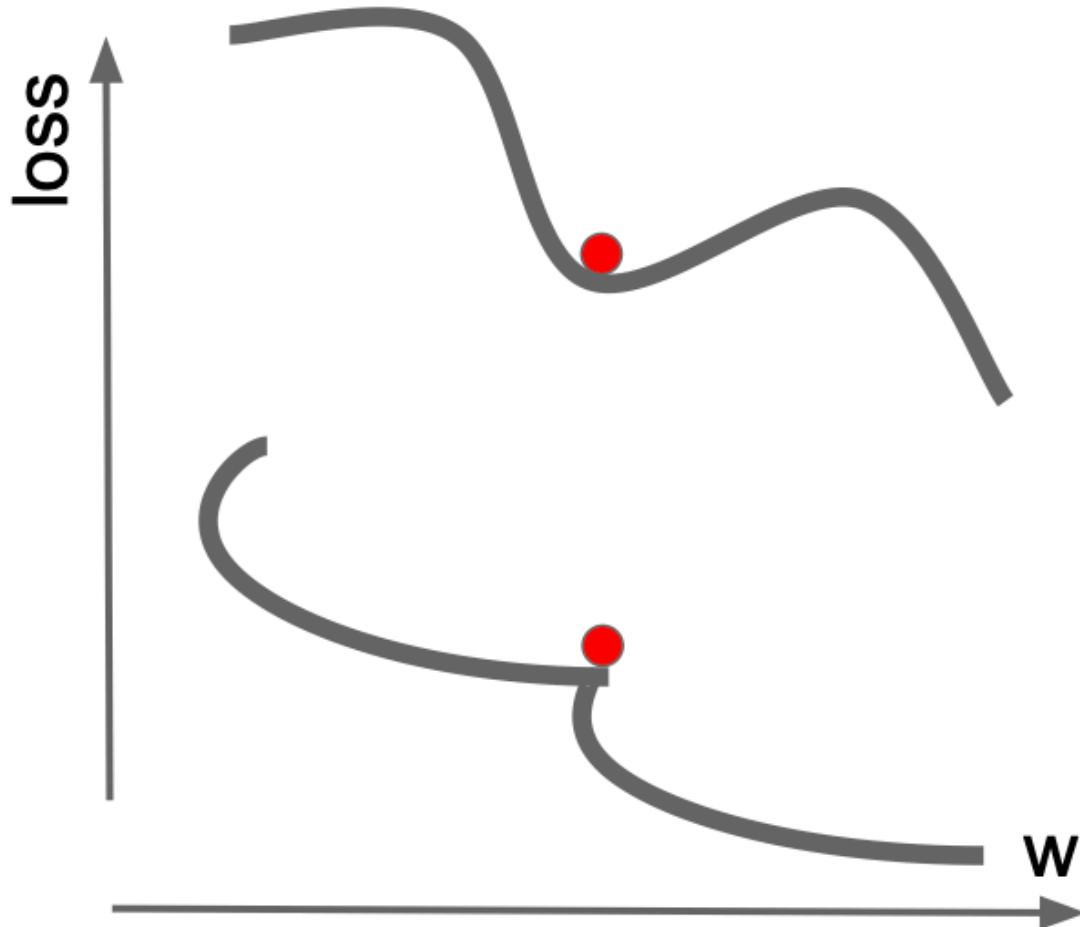
What if the loss
function has a
**local minima or
saddle point?**



Problems with SGD: #2

What if the loss function has a local minima or saddle point?

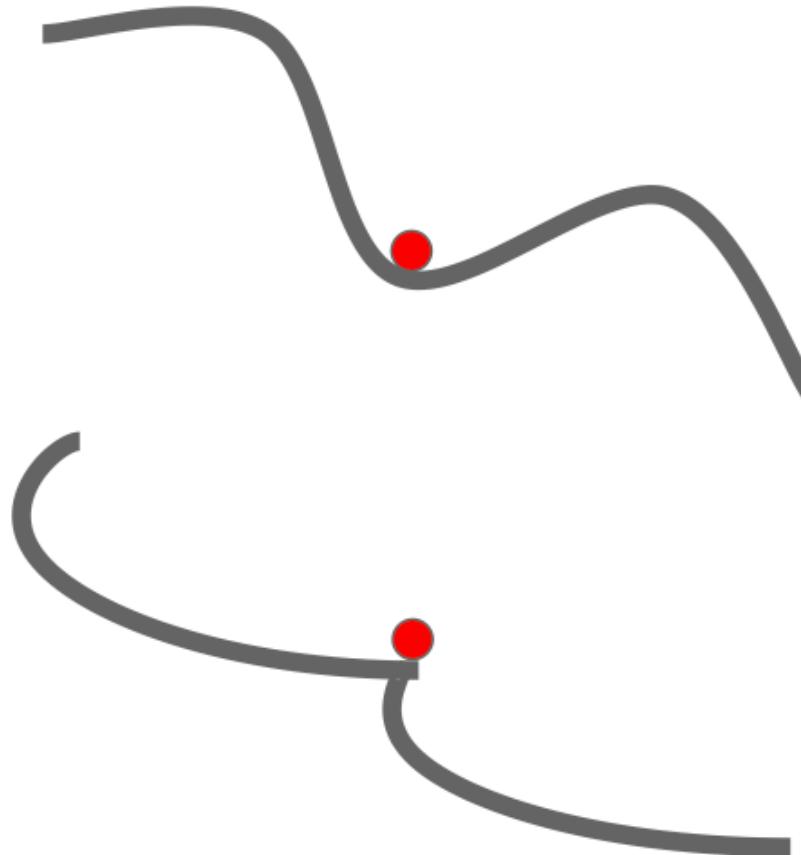
Zero gradient, gradient descent gets stuck



Problems with SGD: #2

What if the loss
function has a
local minima or
saddle point?

Saddle points much
more common in
high dimension



Dauphin et al, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", NIPS 2014

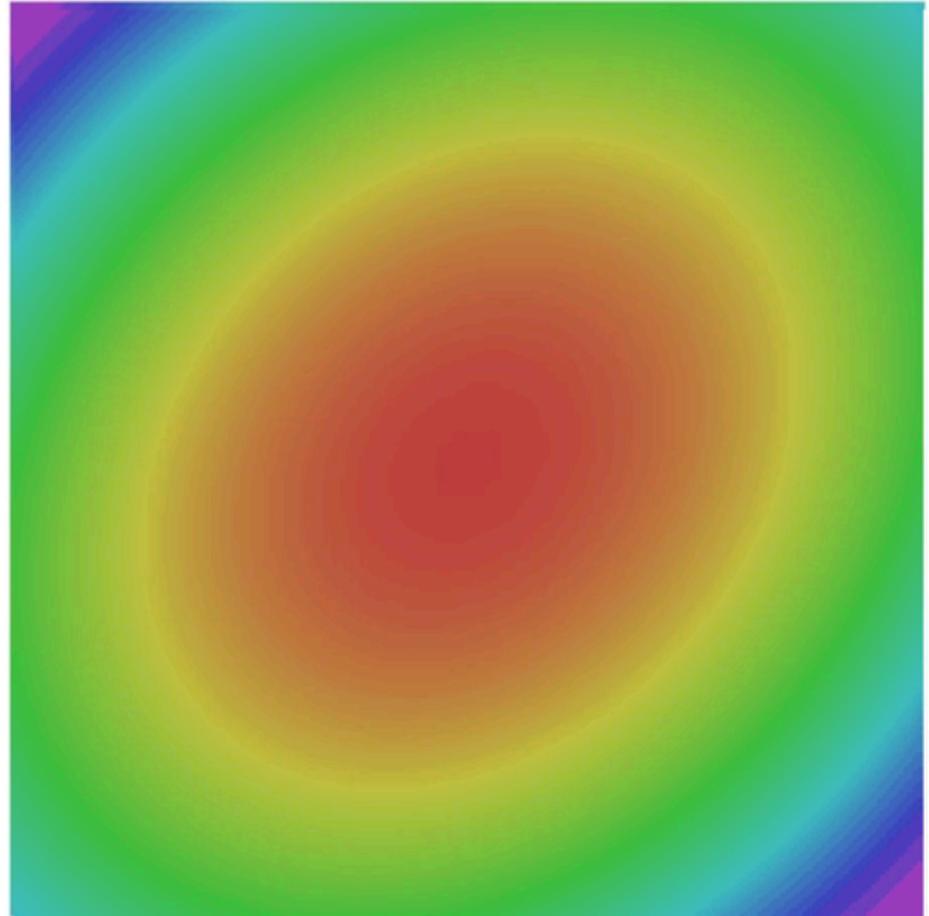
Slide credit: Stanford CS231N

Problems with SGD: #3

Our gradients come from minibatches so they can be noisy!

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

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



SGD + Momentum

continue moving in the general direction as the previous iterations

SGD

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

```
while True:  
    dx = compute_gradient(x)  
    x -= learning_rate * dx
```

SGD+Momentum

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

```
vx = 0  
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x -= learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Slide credit: Stanford CS231N

Adam

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that
first and second moment
estimates start at zero

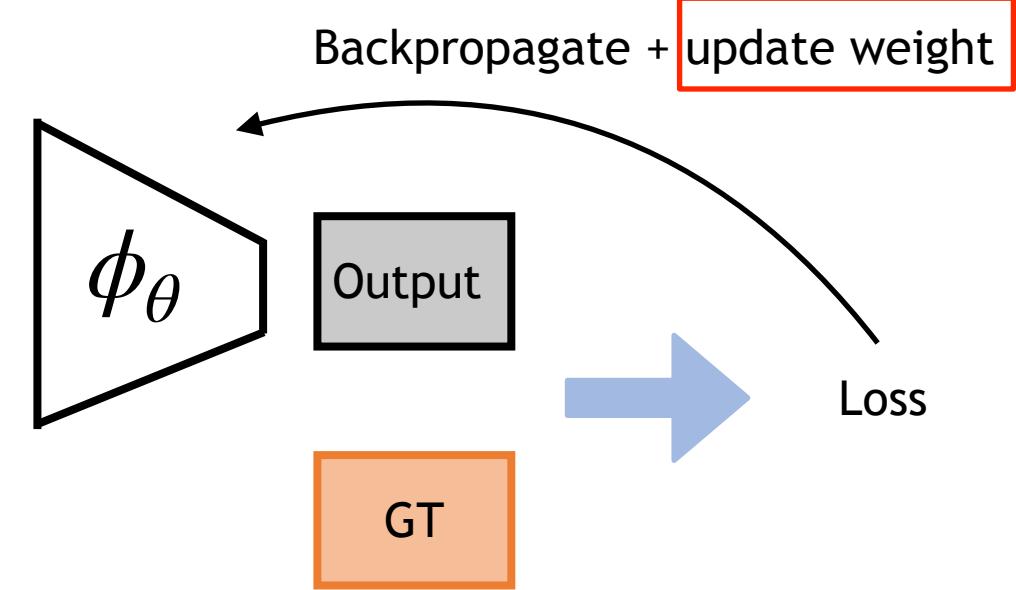
Adam with **beta1 = 0.9**,
beta2 = 0.999, and **learning_rate = 1e-3 or 5e-4**
is a great starting point for many models!

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

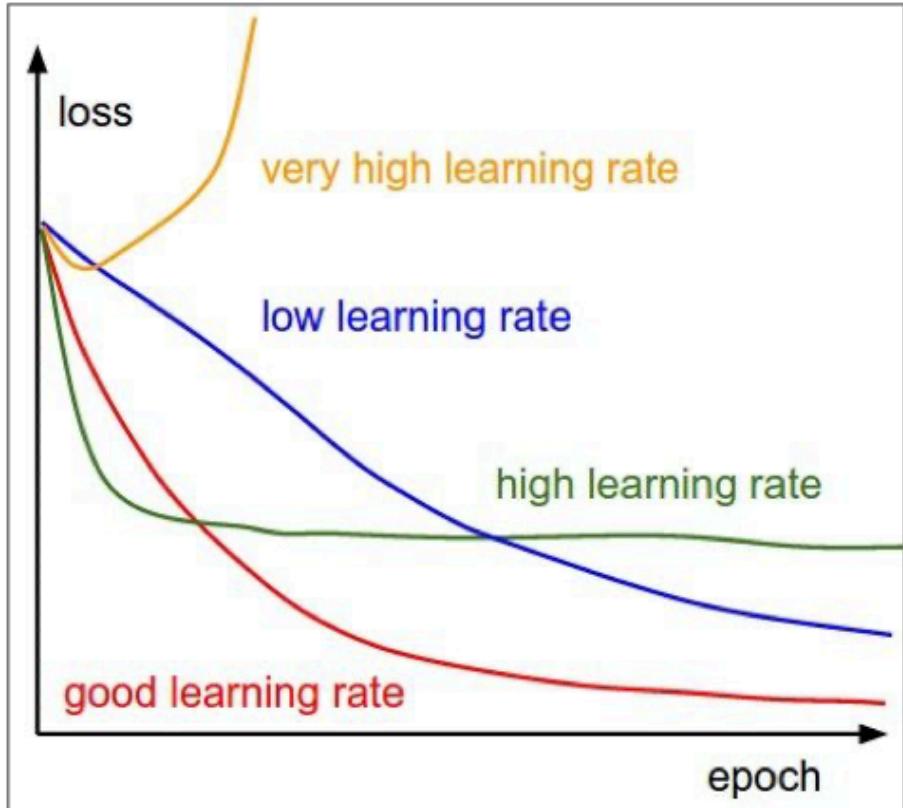
Slide credit: Stanford CS231N

Outline

- Data preparation
- Weight Initialization
- Set a loss function
- Start optimization
 - optimizer?
 - learning rate?



Loss Curves for Different Learning Rates



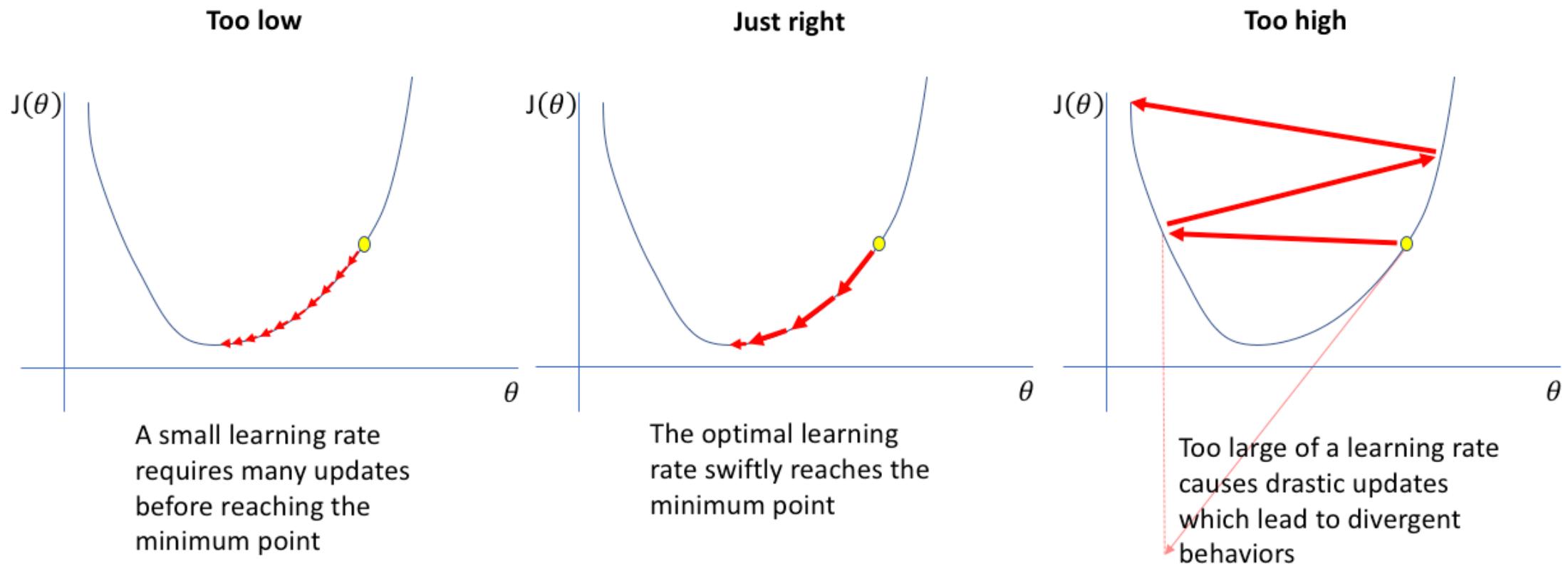
- An appropriate learning rate for classification: $1e-6 \sim 1e-3$
- Low learning rate: undershoot
- High learning rate: overshoot

Figure credit: Stanford CS231N.

Iteration and Epoch

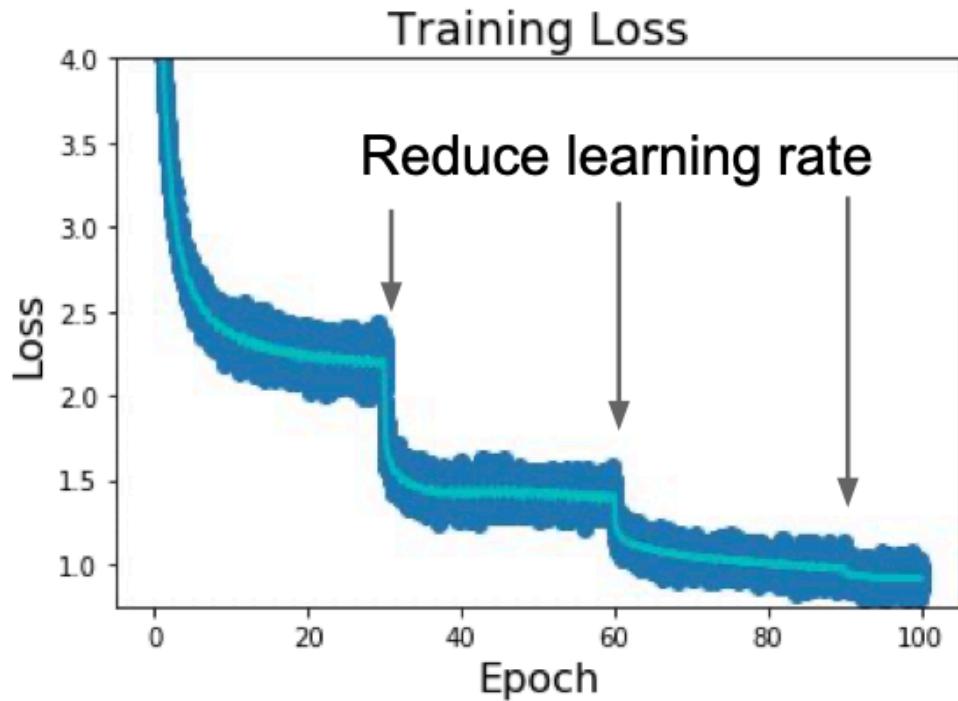
- Iteration:
 - One batch (whose size is called batch size)
 - A gradient descent step
- Epoch
 - Contains many iterations that go over the training data for one complete pass
 - After a epoch, plot train curve, evaluate on val, save model...

Learning Rate



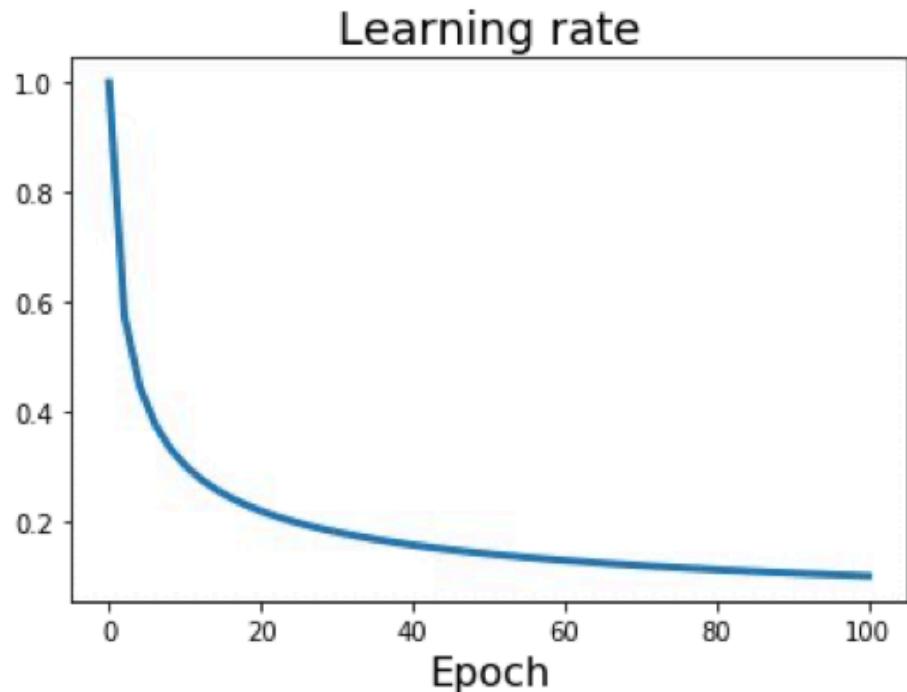
Learning Rate Schedule

Idea: high learning rate at the beginning, decay it later



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 Schedule



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

Linear: $\alpha_t = \alpha_0(1 - t/T)$

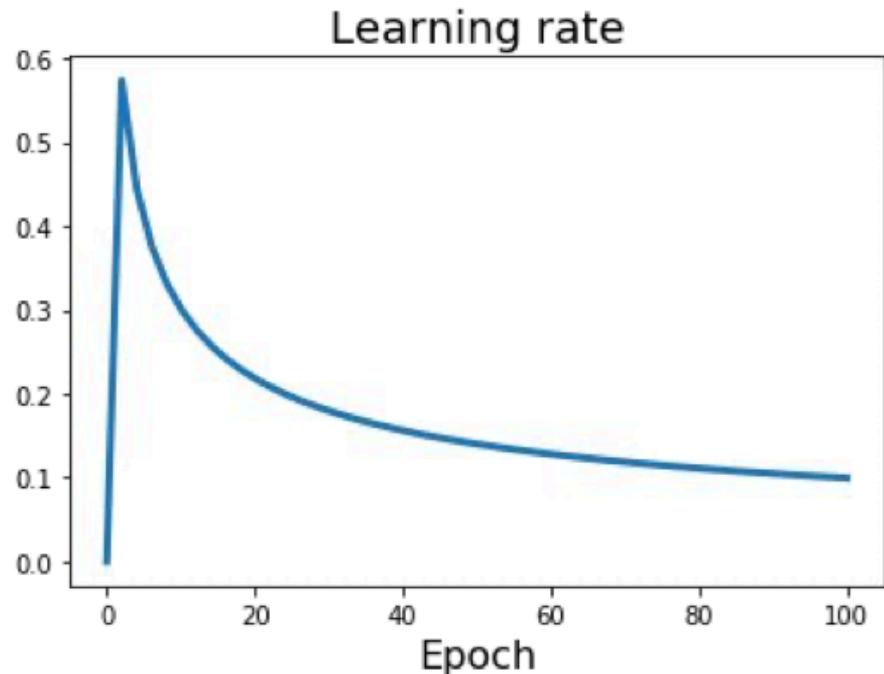
Inverse sqrt: $\alpha_t = \alpha_0 / \sqrt{t}$

α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs

Learning Rate Schedule: Linear Warmup



High initial learning rates can make loss explode; linearly increasing learning rate from 0 over the first ~5000 iterations can prevent this

Batch Size and Learning Rate

- An empirical rule of thumb: if you increase the batch size by N, also scale the initial learning rate by N.
- Why? Suggested reading: visualizing learning rate vs. batch size.

Summary of Learning Rate Schedule

- Adam is a good default choice working okay with constant learning rate.
- SGD + Momentum can outperform Adam but may require more tuning of LR and schedule
 - Try cosine schedule: very few hyper parameters.
- If you are new to a dataset, use Adam with constant learning rate until you see it converges and then modify the learning rate.



Introduction to Computer Vision

Next week: Lecture 6,
Deep Learning III