Midterm Review

50 minutes is short!

This is just to help you get going with your studies.

Overview of today’s session

Summary of Course Material:

- How we “power” neural networks:
  - Loss function
  - Optimization
- How we build complex network models
  - Nonlinear Activations
  - Convolutional Layers
- How we “rein in” complexity
  - Regularization

Practice Midterm Problems

Q&A, time permitting

Overview of today's session

Summary of Course Material

- How we "power" neural networks:
  - Loss function
  - Optimization
- How we build complex network models
  - Nonlinear Activations
  - Convolutional Layers
- How we "rein in" complexity
  - Regularization

Practice Midterm Problems

Lecture 3: Loss Functions and Optimization

An optimization problem

At the end of the day, we want to train a model that performs a desired task well – and a proxy for best achieving this is minimizing a loss function

SVM/Softmax Loss

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

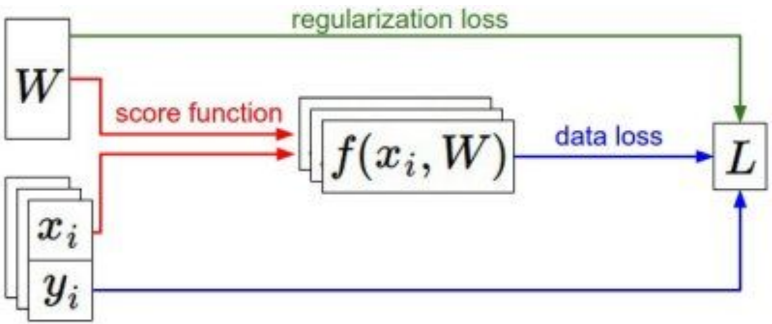
Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss



Know how to derive the SVM and Softmax gradients!

Stochastic Gradient Descent (SGD)

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

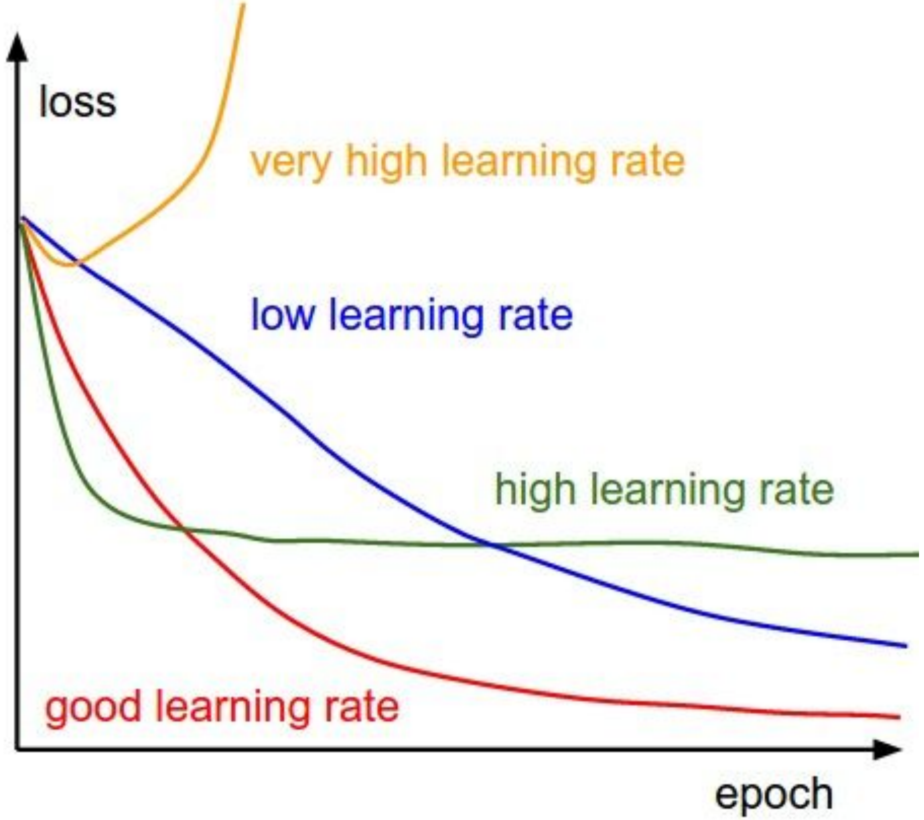
$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

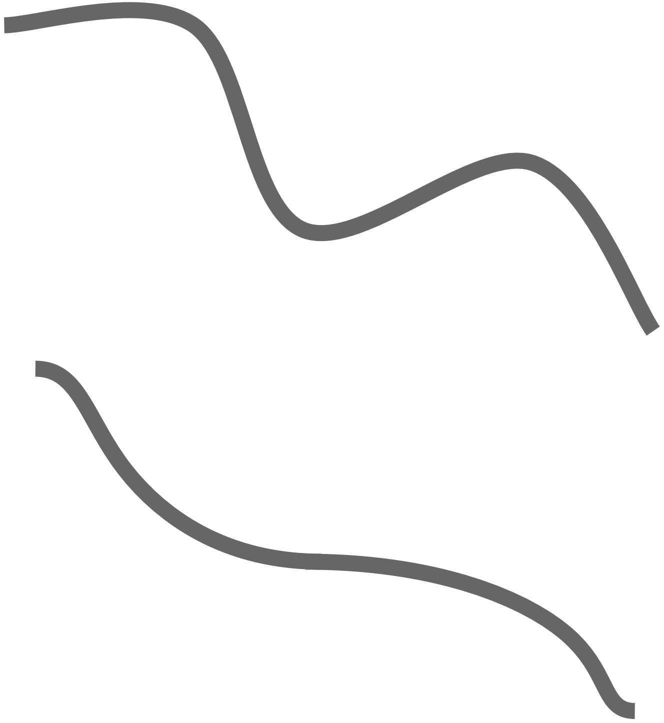
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Learning Rate Loss Curves



Optimization: Problems with SGD

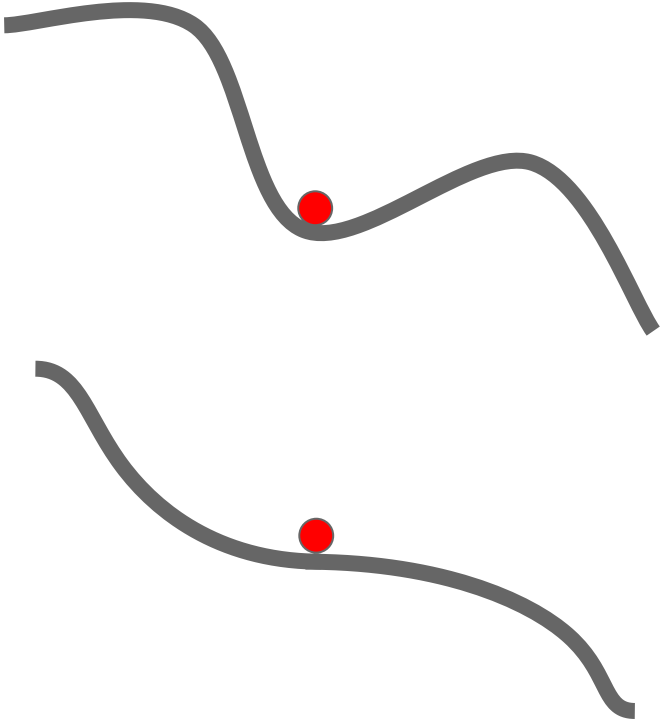
What if the loss function has a **local minima** or **saddle point**?



Optimization: Problems with SGD

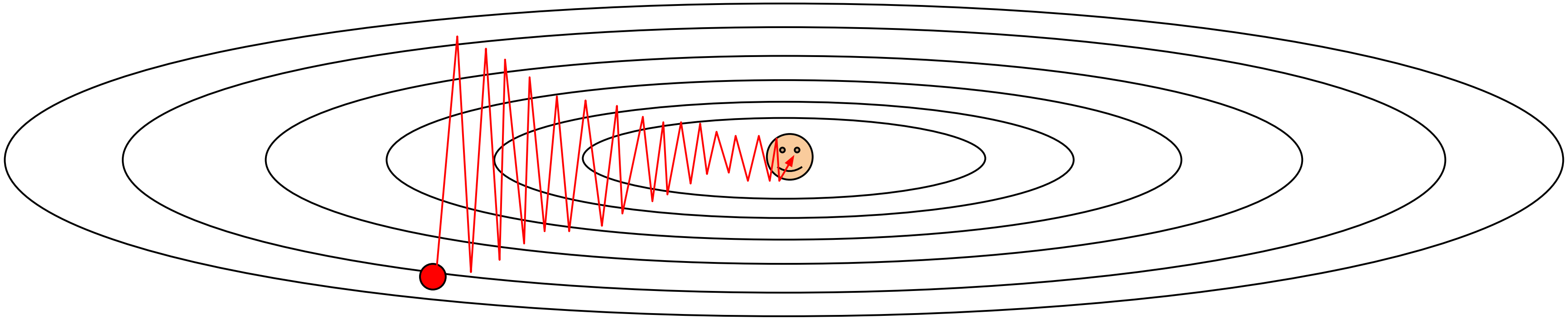
What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck



Optimization: Problems with SGD

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



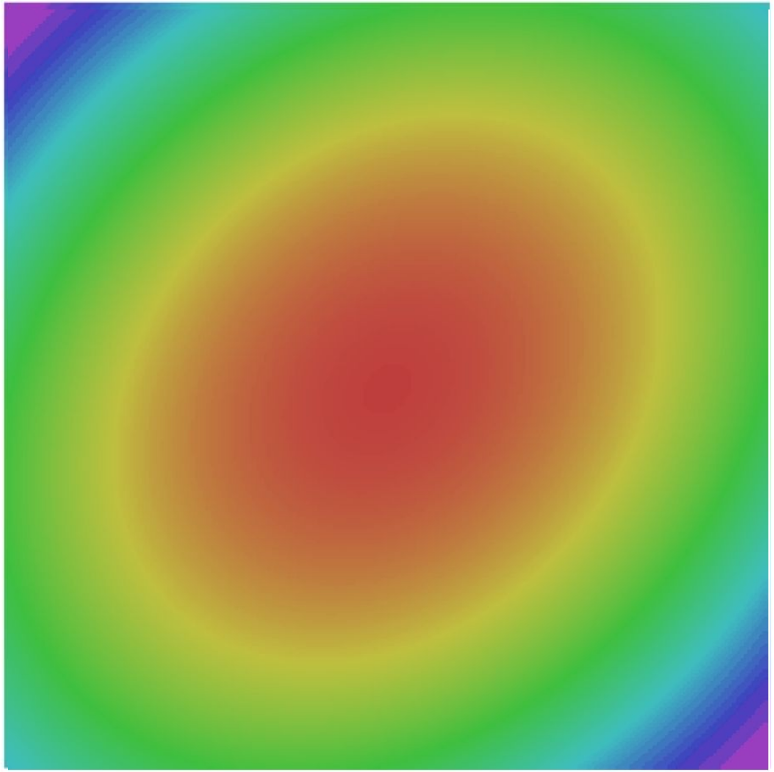
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

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

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



Update Rules

SGD
Momentum
Nesterov Momentum
AdaGrad
RMSProp
Adam

Overview of today's session

Summary of Course Material:

- How we "power" neural networks:
  - Loss function
  - Optimization
- How we build complex network models
  - Nonlinear Activations
  - Convolutional Layers
- How we "rein in" complexity
  - Regularization

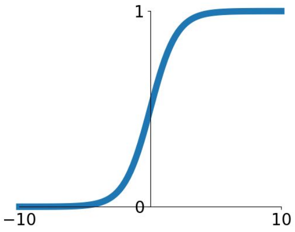
Practice Midterm Problems

Lecture 6:
Training Neural Networks,
Part I

Activation Functions

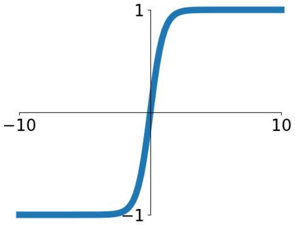
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



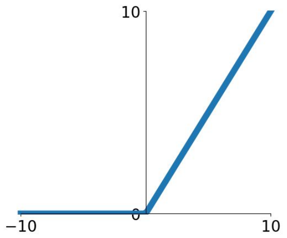
tanh

$\tanh(x)$



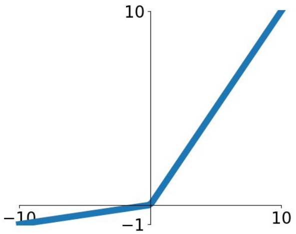
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

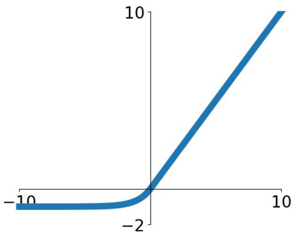


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

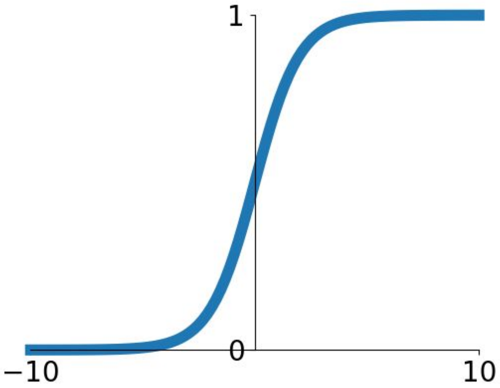
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions

$$\sigma(x) = 1/(1+e^{-x})$$



Sigmoid

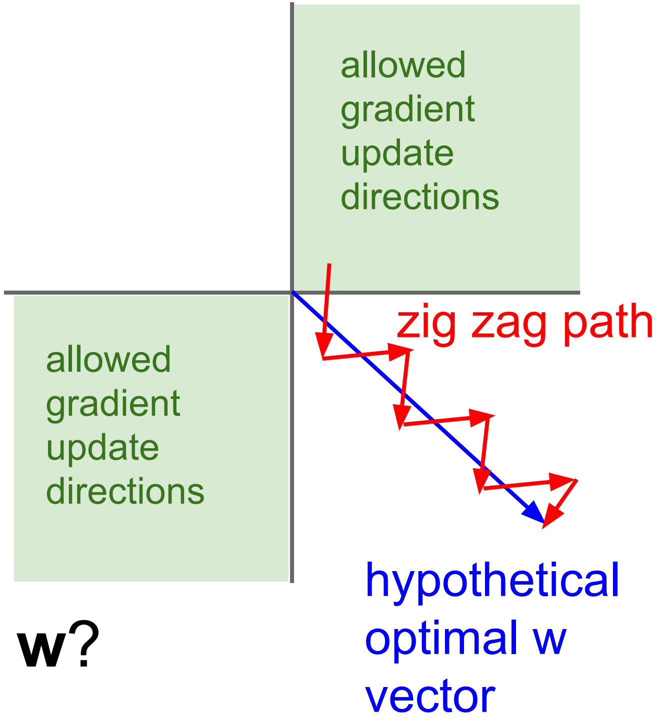
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

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

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

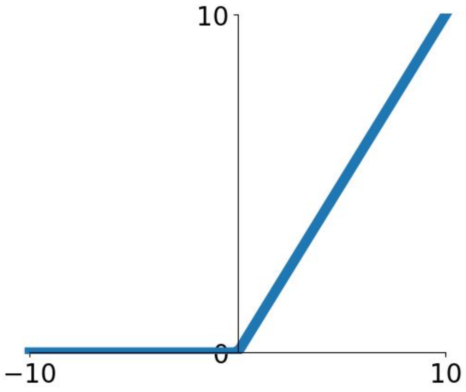


What can we say about the gradients on **w**?

Always all positive or all negative :(

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

Activation Functions



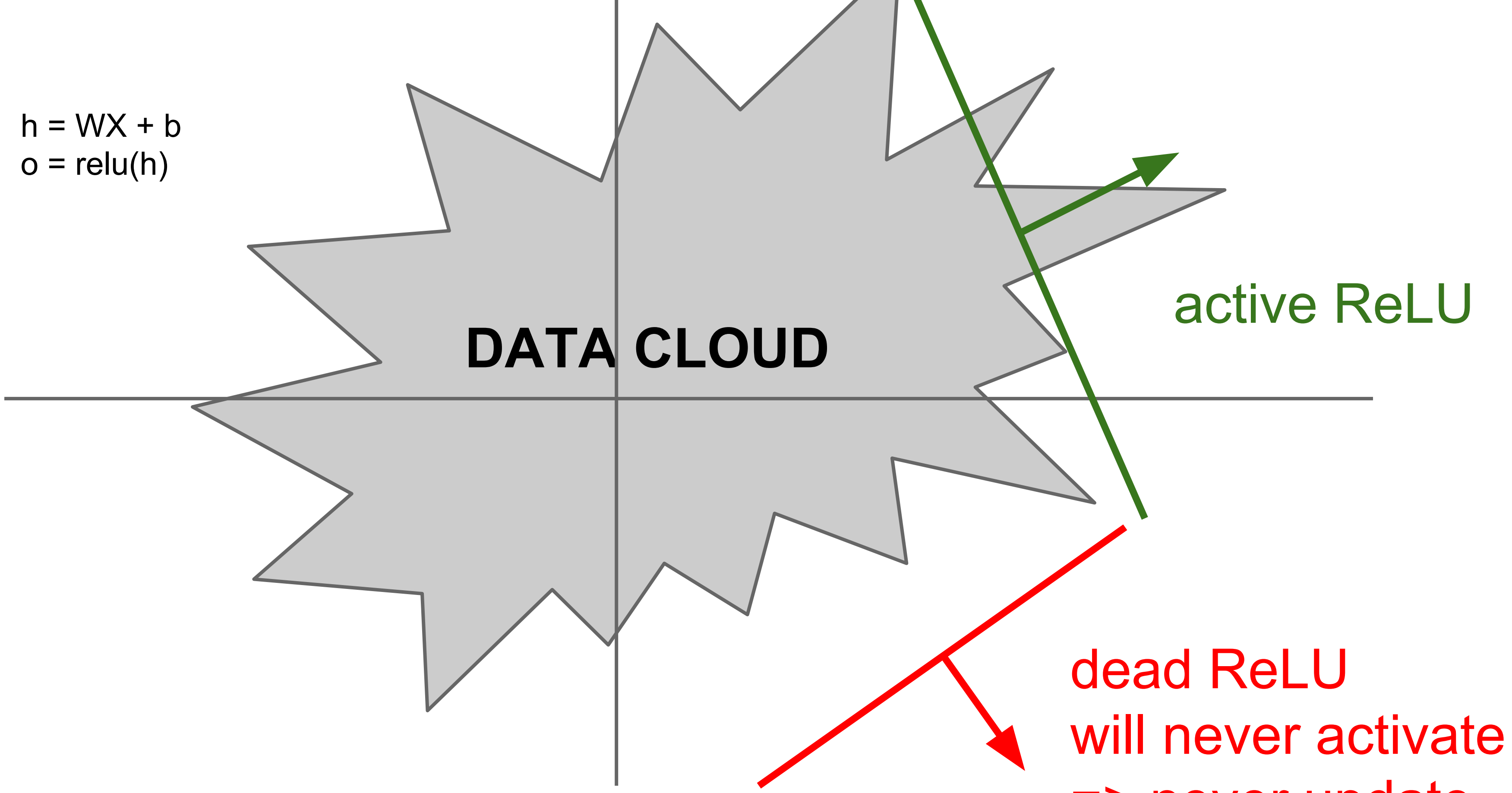
ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

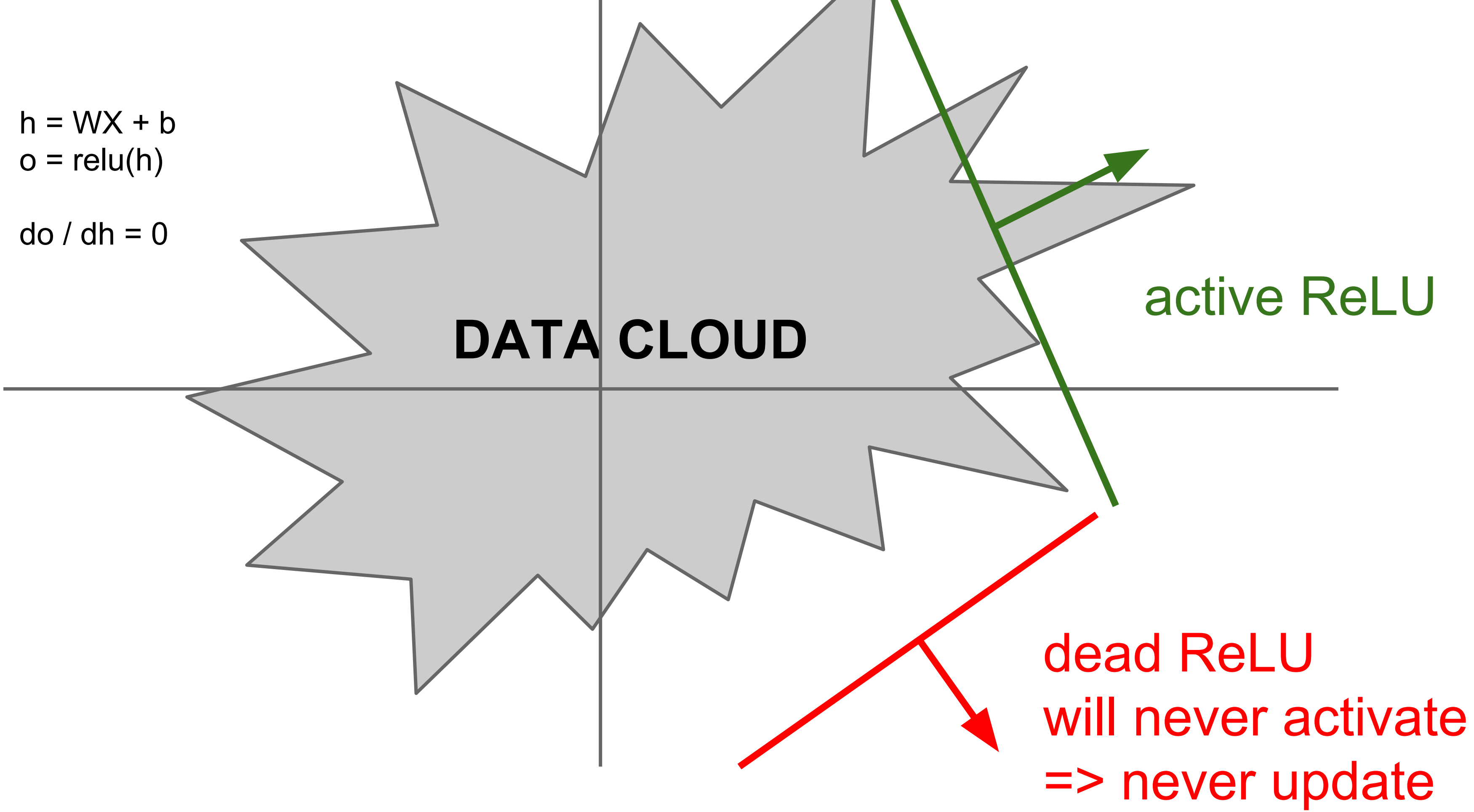
- Not zero-centered output
- An annoyance:

hint: what is the gradient when x < 0?


h = WX + b
o = relu(h)
DATA CLOUD
active ReLU
dead ReLU
will never activate
=> never update

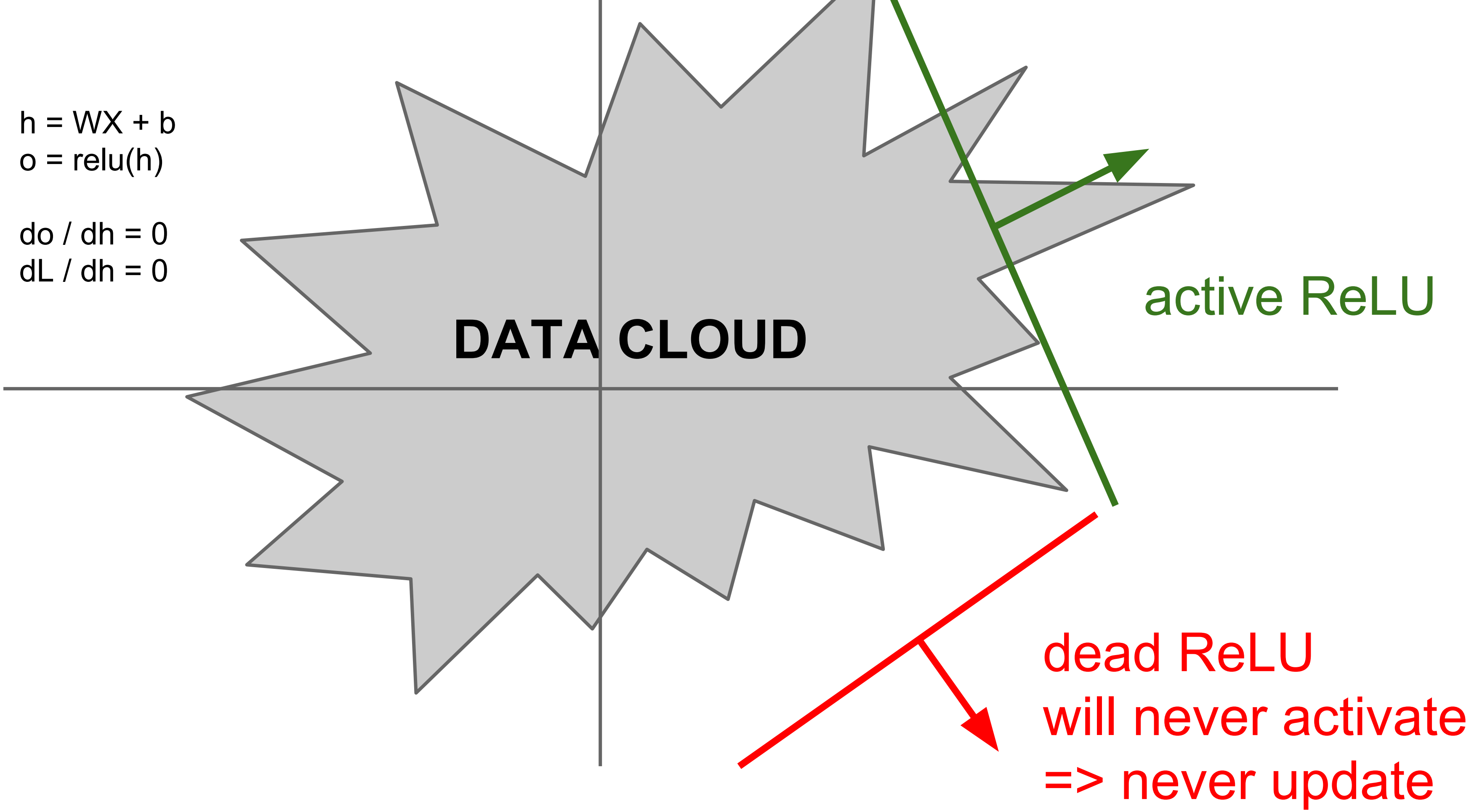
$h = WX + b$
$o = \text{relu}(h)$

$do / dh = 0$



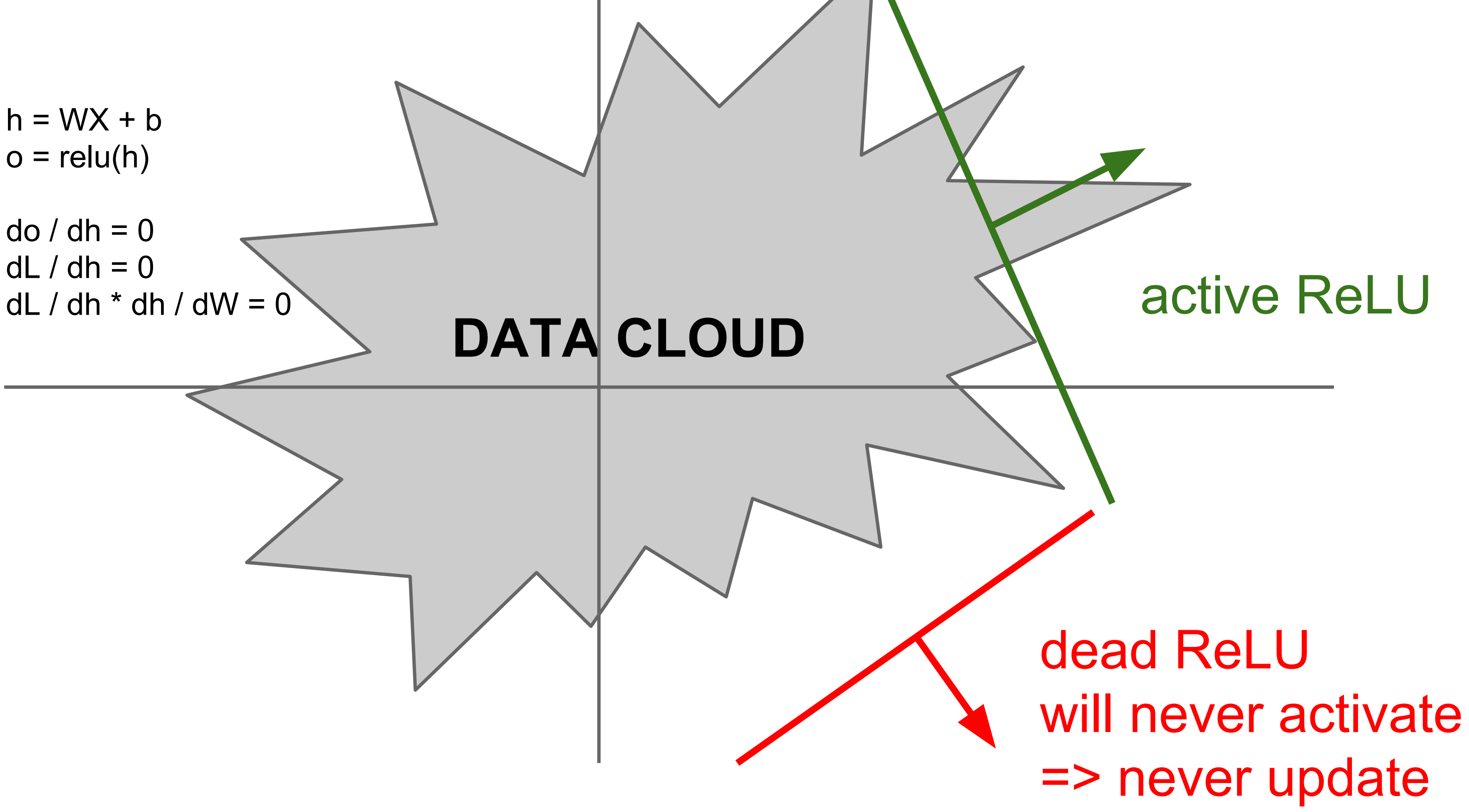
$h = WX + b$
$o = \mathrm{relu}(h)$

$do / dh = 0$
$dL / dh = 0$



h = WX + b
o = relu(h)

do / dh = 0
dL / dh = 0
dL / dh * dh / dW = 0



Vanishing/Exploding Gradient

Vanishing Gradient:

- Gradient becomes too small

Vanishing/Exploding Gradient

Vanishing Gradient:

- Gradient becomes too small
- Some causes:
  - Choice of activation function
  - Multiplying many small numbers together

Vanishing/Exploding Gradient

Vanishing Gradient:

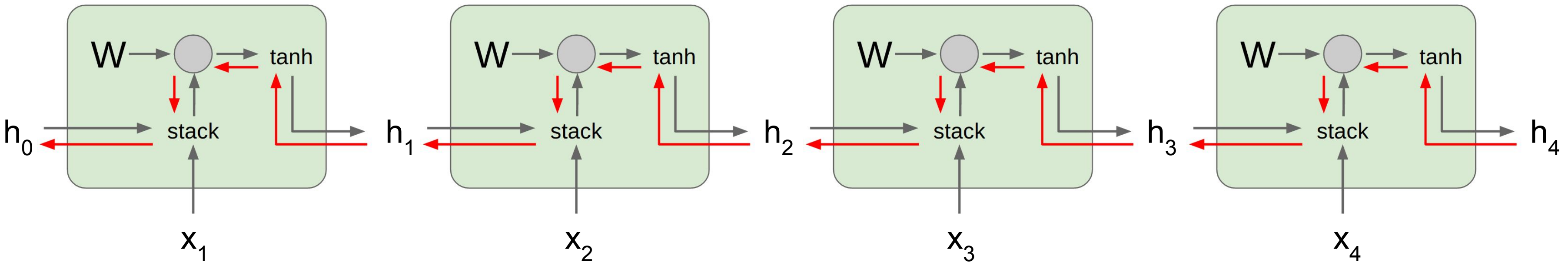
- Gradient becomes too small
- Some causes:
  - Choice of activation function
  - Multiplying many small numbers together

Exploding Gradient:

- Gradient becomes too large

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

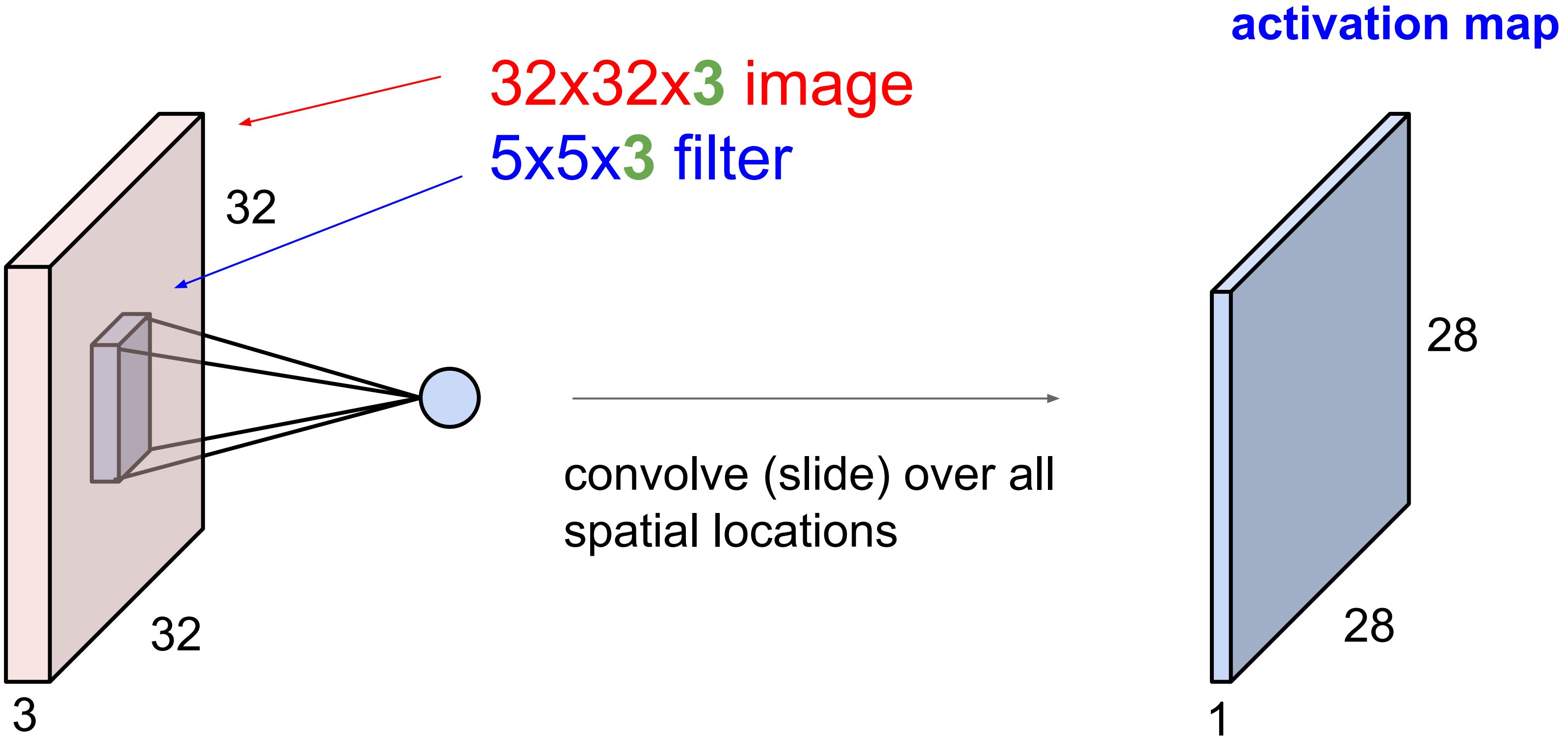
Overview of today’s session

Summary of Course Material:

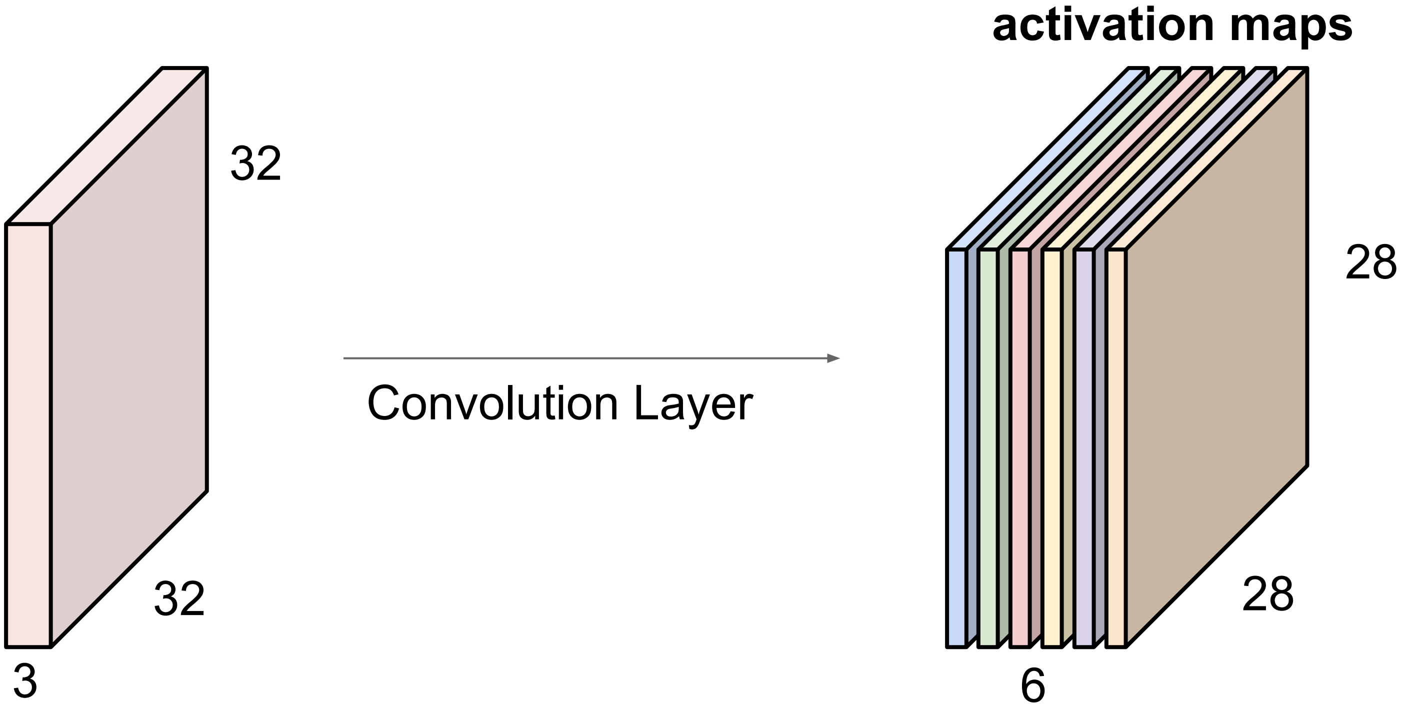
- How we “power” neural networks:
  - Loss function
  - Optimization
- How we build complex network models
  - Nonlinear Activations
  - Convolutional Layers
- How we “rein in” complexity
  - Regularization

Practice Midterm Problems

Convolution Layer



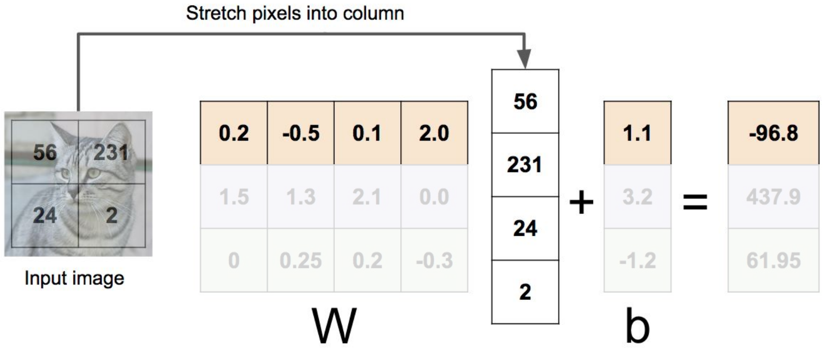
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

Convolution Layer

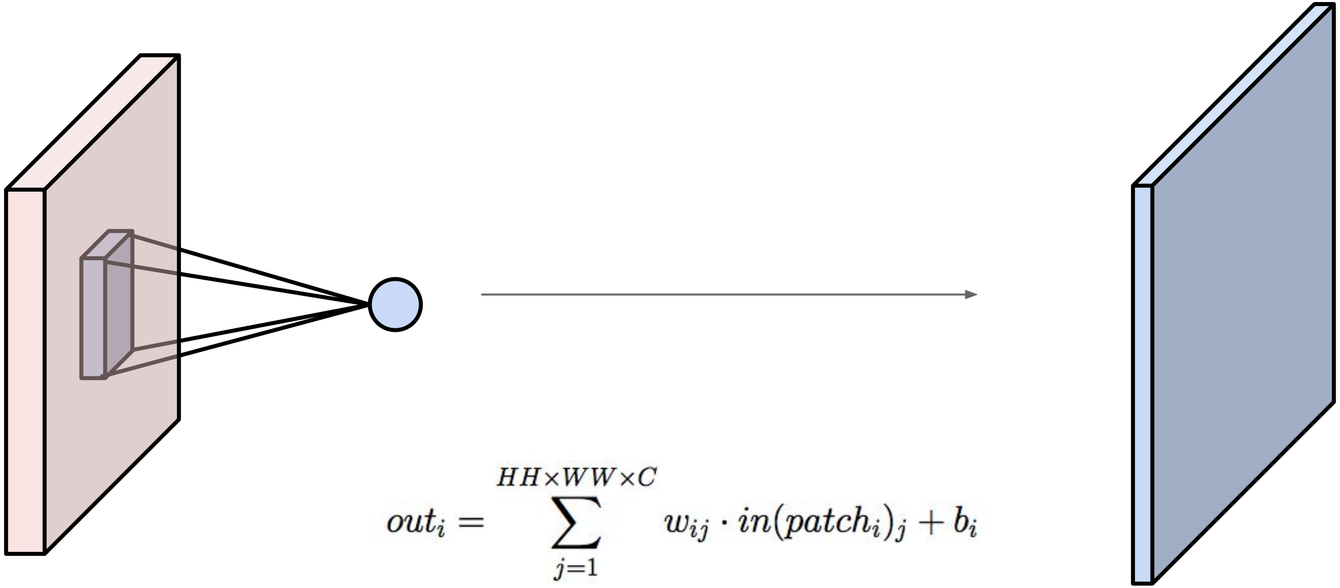
In contrast to fully connected layer,
Each term in output is dependent on spatially local 'subregions' of input



$$out_i = \sum_{j=1}^{H \times W \times C} w_{ij} \cdot in_j + b_i$$

Convolution Layer

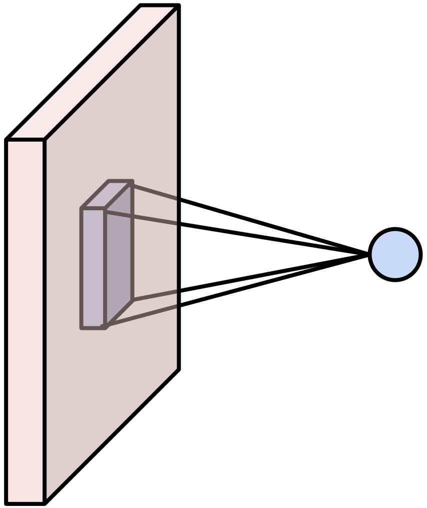
In contrast to fully connected layer,
Each term in output is dependent on spatially local 'subregions' of input



$$out_i = \sum_{j=1}^{HH \times WW \times C} w_{ij} \cdot in(patch_i)_j + b_i$$

Convolution Layer

In contrast to fully connected layer,
Each term in output is dependent on spatially local 'subregions' of input



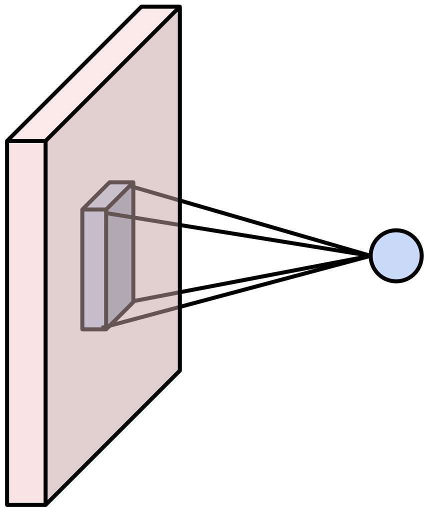
Question: connection between an FC layer and a convolutional layer?

$$out_i = \sum_{j=1}^{H\times W\times C} w_{ij} \cdot in_j + b_i$$

$$out_i = \sum_{j=1}^{HH\times WW\times C} w_{ij} \cdot in(patch_i)_j + b_i$$

Convolution Layer

In contrast to fully connected layer,
Each term in output is dependent on spatially local 'subregions' of input



Question: connection between an FC layer and a convolutional layer?
Answer: FC looks like convolution layer with filter size HxW

$$out_i = \sum_{j=1}^{H \times W \times C} w_{ij} \cdot in_j + b_i$$

$$out_i = \sum_{j=1}^{HH \times WW \times C} w_{ij} \cdot in(patch_i)_j + b_i$$

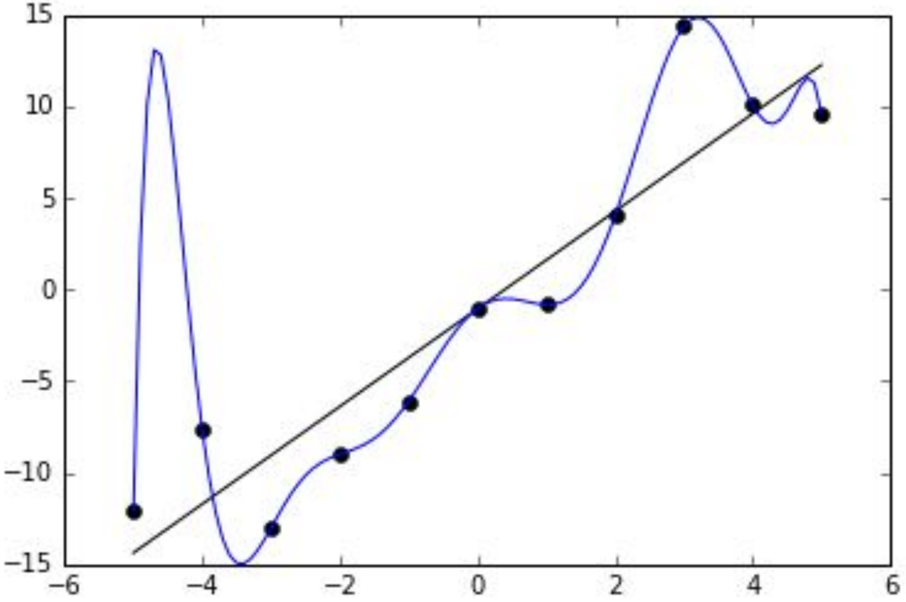
Overview of today's session

Summary of Course Material:

- How we "power" neural networks:
  - Loss function
  - Optimization
- How we build complex network models
  - Nonlinear Activations
  - Convolutional Layers
- How we "rein in" complexity
  - Regularization

Practice Midterm Problems

Drawbacks of increased complexity: Overfitting (Bias vs Variance)



Source: Wikipedia

Combat overfitting

- Increase data quantity/quality
- Impose extra constraints
- Introduce randomness/uncertainty

Combat overfitting

- Increase data quantity/quality
  - Data augmentation
- Impose extra constraints
  - On model parameters: L2 regularization
  - On layer outputs: Batchnorm
- Introduce randomness/uncertainty
  - Dropout
  - Batchnorm
  - Stochastic depth, drop connect

Overview of today's session

Summary of Course Material:

- How we “power” neural networks:
  - Loss function
  - Optimization
- How we build complex network models
  - Nonlinear Activations
  - Convolutional Layers
- How we “rein in” complexity
  - Regularization

Practice Midterm Problems

3.2 Convolutional Architectures

Consider the convolutional network defined by the layers in the left column below. Fill in the size of the activation volumes at each layer, and the number of parameters at each layer. You can write your answer as a multiplication (e.g. 128x128x3).

- CONV5-N denotes a convolutional layer with N filters, each of size 5x5xD, where D is the depth of the activation volume at the previous layer. Padding is 2, and stride is 1.
- POOL2 denotes a 2x2 max-pooling layer with stride 2 (pad 0)
- FC-N denotes a fully-connected layer with N output neurons.

Layer	Activation Volume Dimensions (memory)	Number of parameters
INPUT	32x32x1	0
CONV5-10		
POOL2		

3.2 Convolutional Architectures

Consider the convolutional network defined by the layers in the left column below. Fill in the size of the activation volumes at each layer, and the number of parameters at each layer. You can write your answer as a multiplication (e.g. 128x128x3).

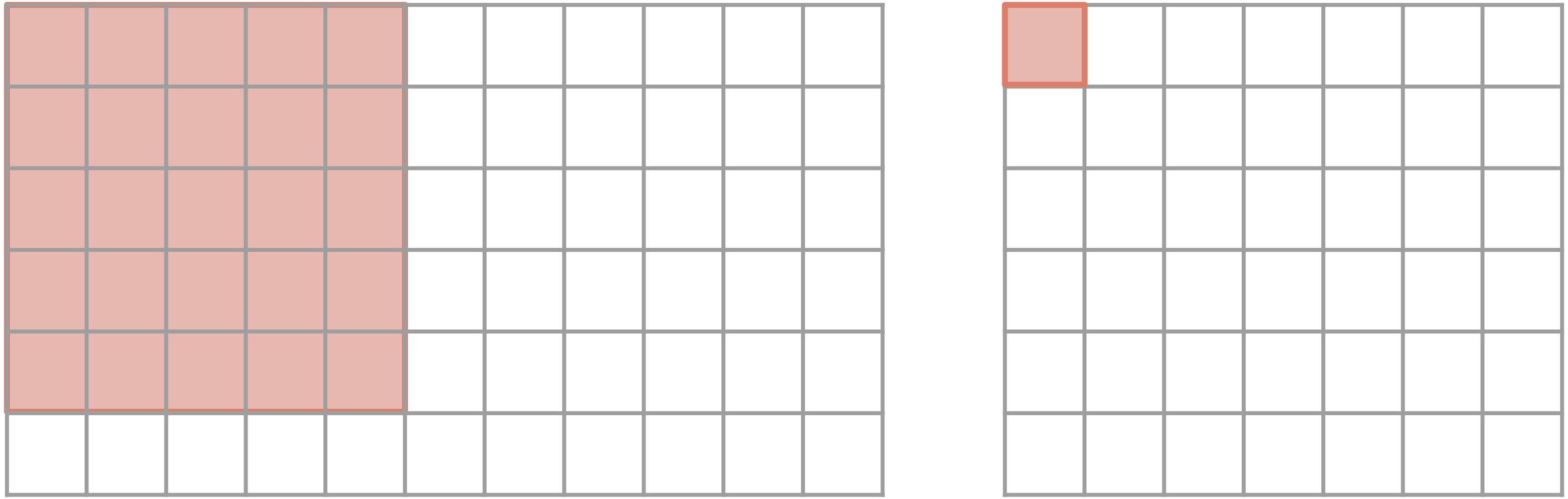
- CONV5-N denotes a convolutional layer with N filters, each of size 5x5xD, where D is the depth of the activation volume at the previous layer. Padding is 2, and stride is 1.
- POOL2 denotes a 2x2 max-pooling layer with stride 2 (pad 0)
- FC-N denotes a fully-connected layer with N output neurons.

Layer	Activation Volume Dimensions (memory)	Number of parameters
INPUT	32x32x1	0
CONV5-10		
POOL2		

$$w_{out} = \frac{w_{in} + w_{pad} - k}{s} + 1$$

Receptive field size

‘Input data seen/received’ in single activation layer ‘pixel’



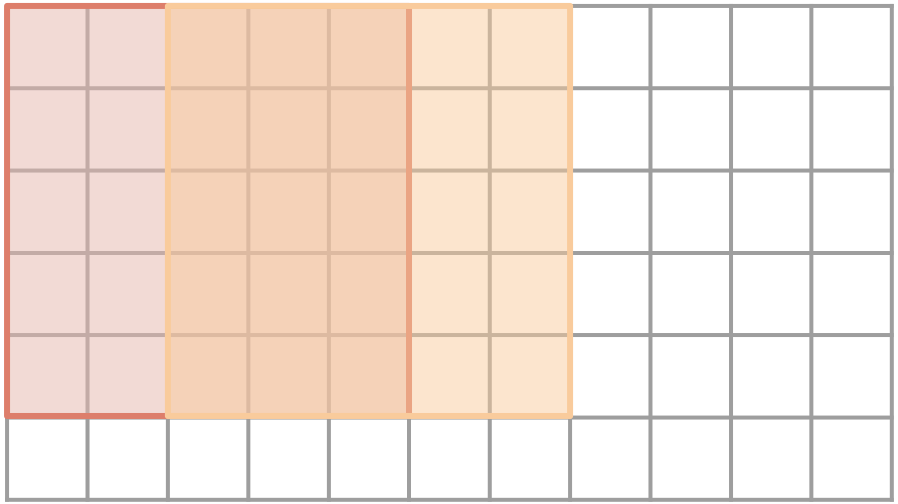
Input

Conv2d

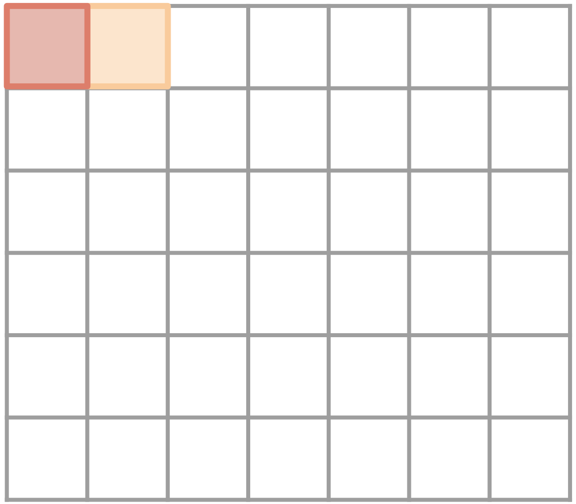
$$k = 5\ ,\ s = 2$$

Receptive field size

'Input data seen/received' in single activation layer 'pixel'



Input

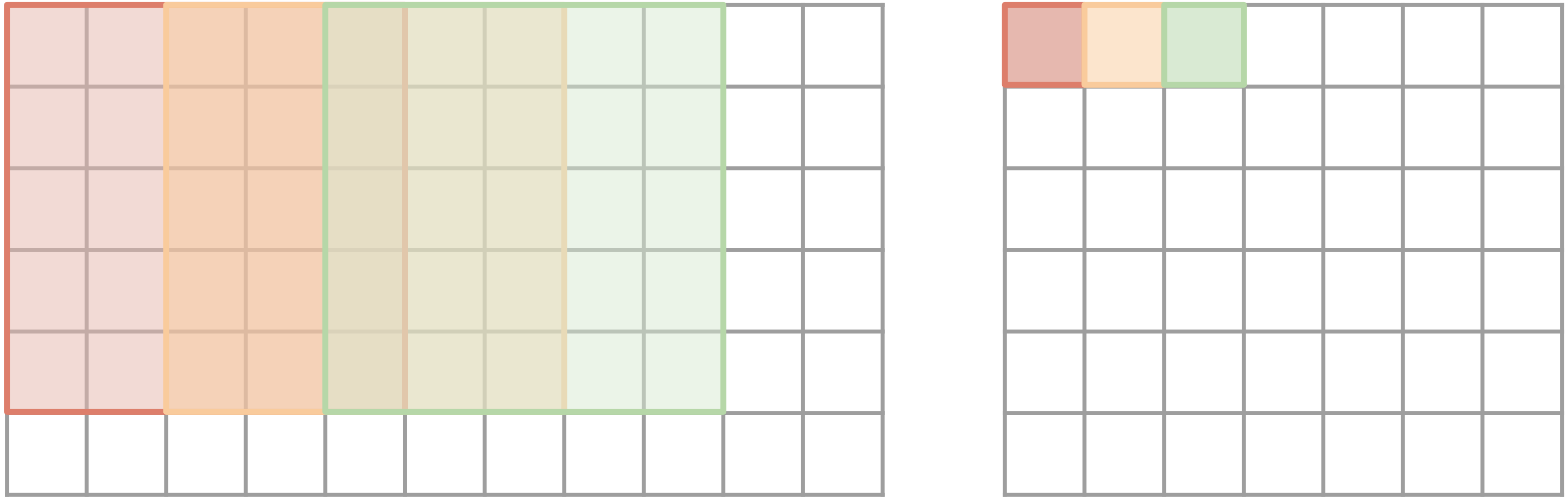


Conv2d

$$k = 5\ ,\ s = 2$$

Receptive field size

‘Input data seen/received’ in single output layer ‘pixel’

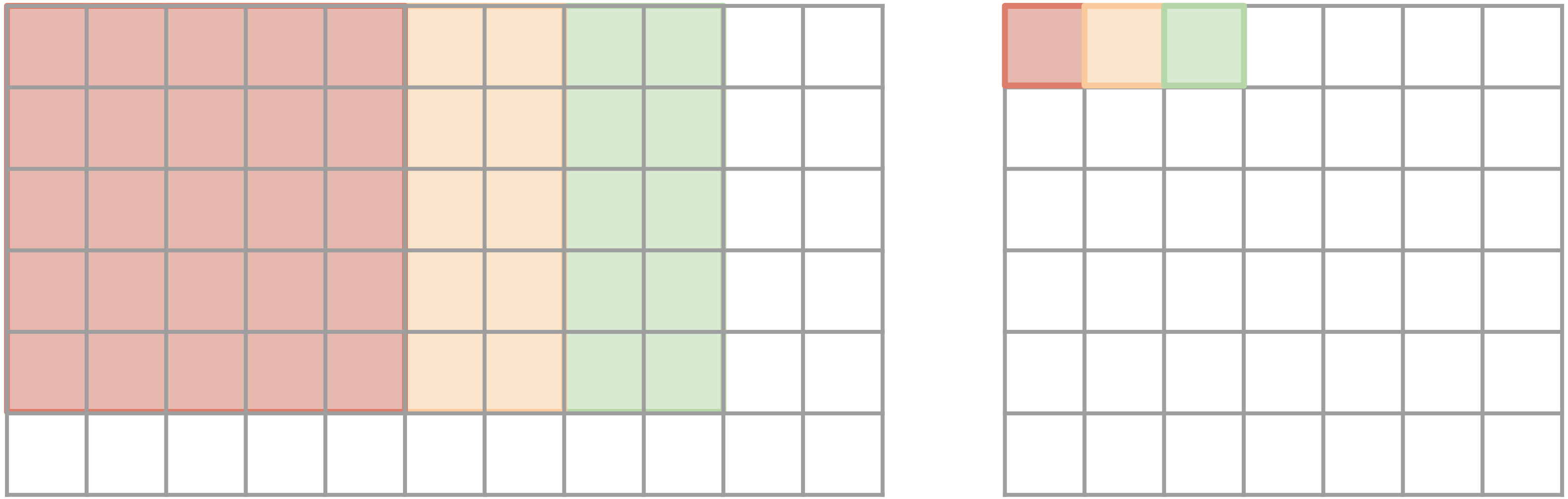


Input

Conv2d

$$k = 5\ ,\ s = 2$$

$$n = 5 + 2 \times (3 - 1)$$

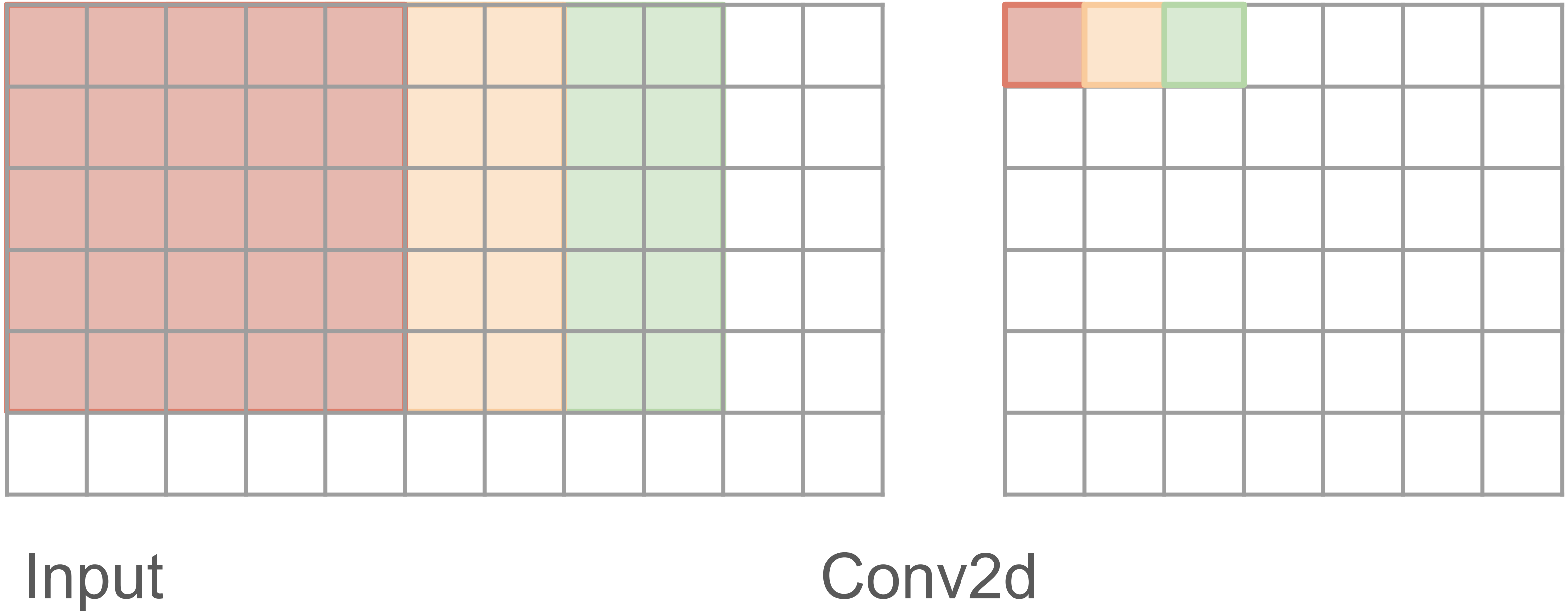


Input

Conv2d

$$k = 5 \ , \ s = 2$$

$$n = 5 + 2 \times (3 - 1) \Rightarrow n = k + s(m - 1)$$



$$k = 5 \ , \ s = 2$$

Summary

Given kernel width **k** and stride **s**,
For **m** adjacent pixels in the activation output,
Cumulative receptive field **n** *with respect to layer input* is

$$n = k + s(m - 1)$$

Summary

Given kernel width **k** and stride **s**,
For **m** adjacent pixels in the activation output,
Cumulative receptive field **n** *with respect to layer input* is

$$n = k + s(m - 1)$$

Note: Generally when we refer to 'receptive field',
we mean with respect to **input data/layer 0/original image**,
not with respect to **direct input to the layer**

Summary

Given kernel width **k** and stride **s**,
For **m** adjacent pixels in the activation output,
Cumulative receptive field **n** *with respect to layer input* is

$$n = k + s(m - 1)$$

(Need to compute recursively!)

Going back to activation dimensions...

For kernel width $\mathbf{k}$ and stride $\mathbf{s}$,
Input width $\mathbf{w}_{in}$ and total padding $\mathbf{w}_{pad}$,
Output width $\mathbf{w}_{out}$ is

Going back to activation dimensions...

For kernel width $\mathbf{k}$ and stride $\mathbf{s}$,
Input width $\mathbf{w}_{in}$ and total padding $\mathbf{w}_{pad}$,
Output width $\mathbf{w}_{out}$ is

Cumulative receptive field of layer output = layer input

Going back to activation dimensions...

For kernel width $\mathbf{k}$ and stride $\mathbf{s}$,
Input width $\mathbf{w}_{in}$ and total padding $\mathbf{w}_{pad}$,
Output width $\mathbf{w}_{out}$ is

Cumulative receptive field of layer output = layer input

$$n = k + s(w_{out} - 1)$$

$$n = w_{in} + w_{pad}$$

Activation dimensions

$$n = k + s(w_{out} - 1)$$

$$k + s(w_{out} - 1) = w_{in} + w_{pad}$$

$$n = w_{in} + w_{pad}$$

Activation dimensions

$$n = k + s(w_{out} - 1)$$

$$n = w_{in} + w_{pad}$$

$$\begin{aligned}
k + s(w_{out} - 1) &= w_{in} + w_{pad} \\
s(w_{out} - 1) &= w_{in} + w_{pad} - k \\
w_{out} - 1 &= \frac{1}{s}(w_{in} + w_{pad} - k) \\
w_{out} &= \frac{1}{s}(w_{in} + w_{pad} - k) + 1
\end{aligned}$$

Activation dimensions

$$n = k + s(w_{out} - 1)$$

$$n = w_{in} + w_{pad}$$

$$k + s(w_{out} - 1) = w_{in} + w_{pad}$$

$$s(w_{out} - 1) = w_{in} + w_{pad} - k$$

$$w_{out} - 1 = \frac{1}{s}(w_{in} + w_{pad} - k)$$

$$w_{out} = \frac{1}{s}(w_{in} + w_{pad} - k) + 1$$

$$w_{out} = \frac{w_{in} + w_{pad} - k}{s} + 1$$

Summary

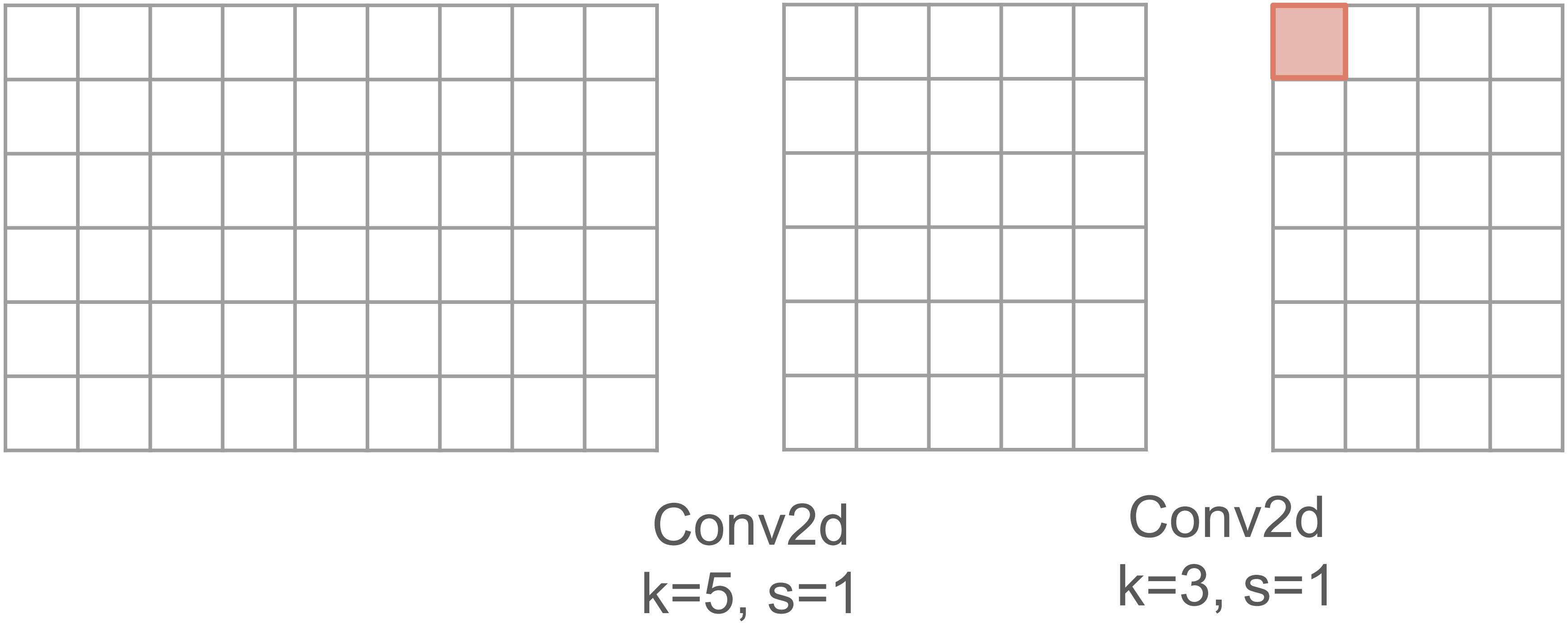
For kernel width $\mathbf{k}$ and stride $\mathbf{s}$,
Input width $\mathbf{w}_{in}$ and total padding $\mathbf{w}_{pad}$,
Output width $\mathbf{w}_{out}$ is

$$w_{out} = \frac{1}{s}(w_{in} + w_{pad} - k) + 1$$

Receptive field size

k=3, s=1, m=1

$$n = k + s(m - 1)$$

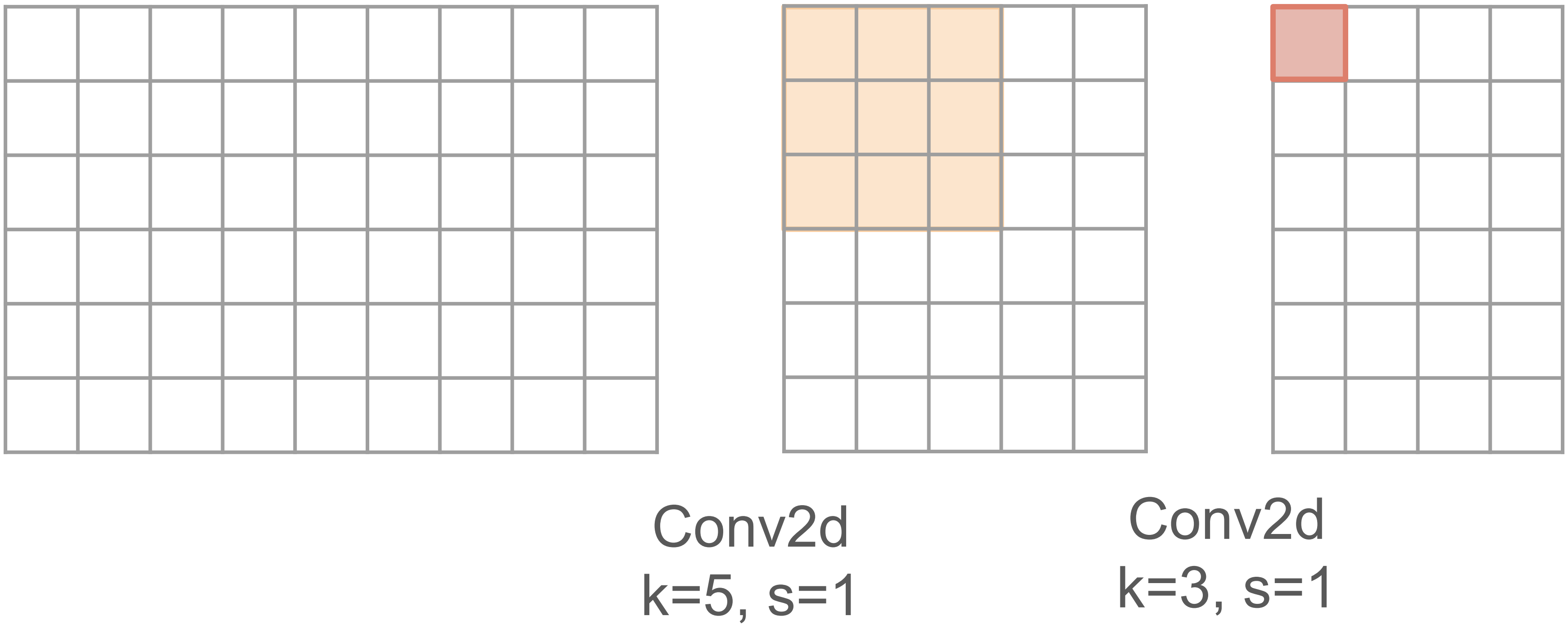


Receptive field size

k=3, s=1, m=1

$$n = k + s(m-1)$$

n=3

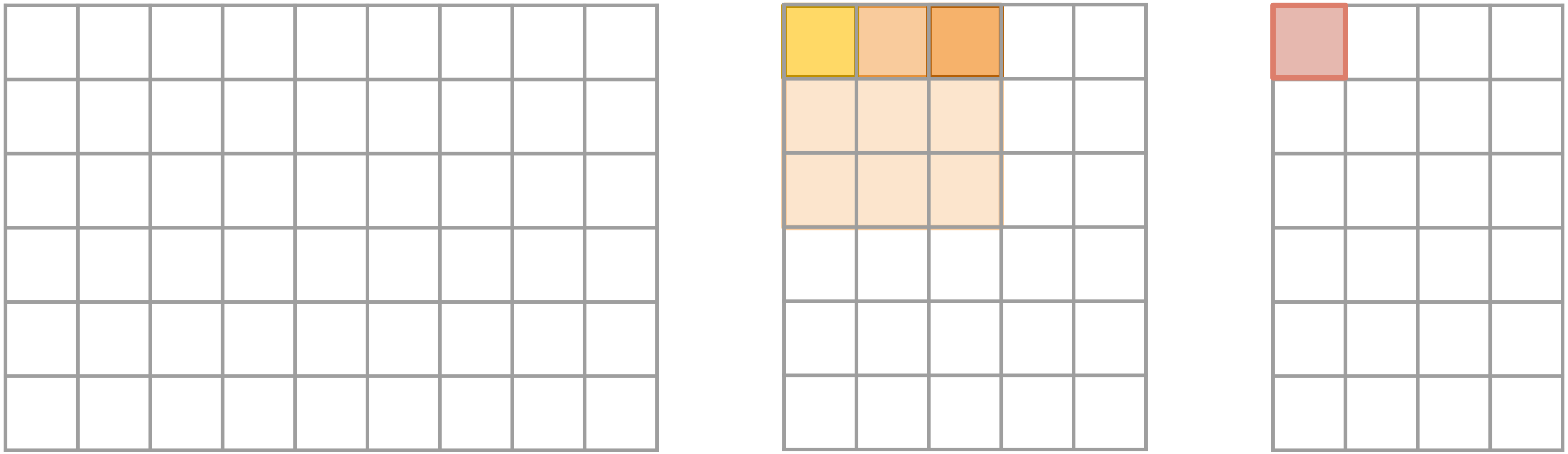


Receptive field size

k=3, s=1, m=1

$$n = k + s(m-1)$$

n=3



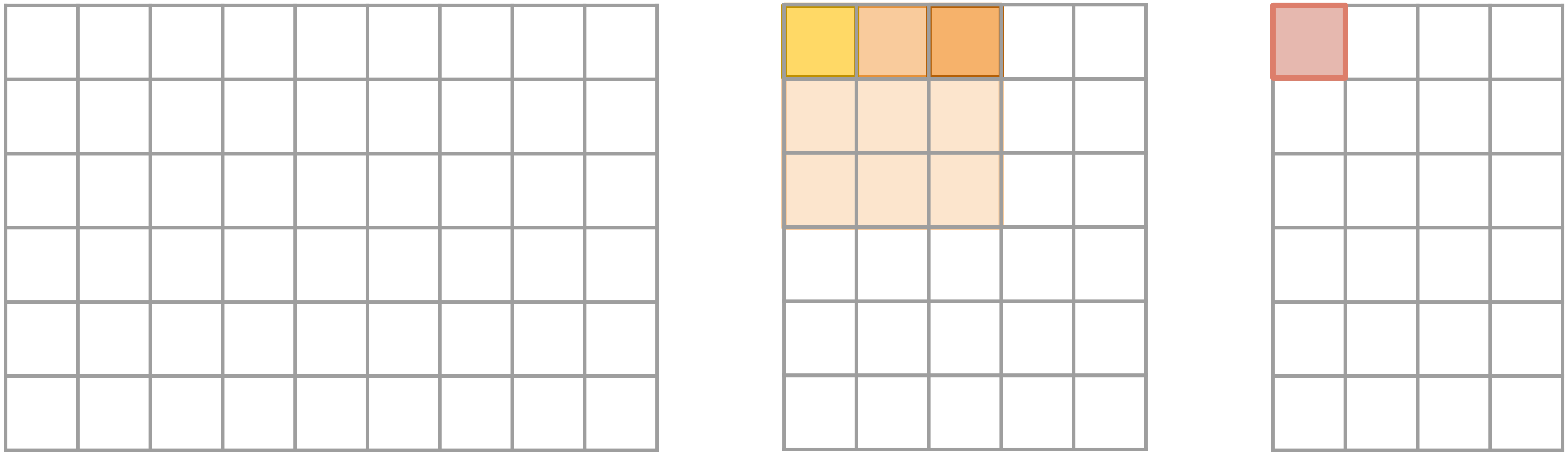
Conv2d
k=5, s=1

Conv2d
k=3, s=1

Receptive field size

k=5, s=1, m=3

$$n = k + s(m-1)$$



Conv2d
k=5, s=1

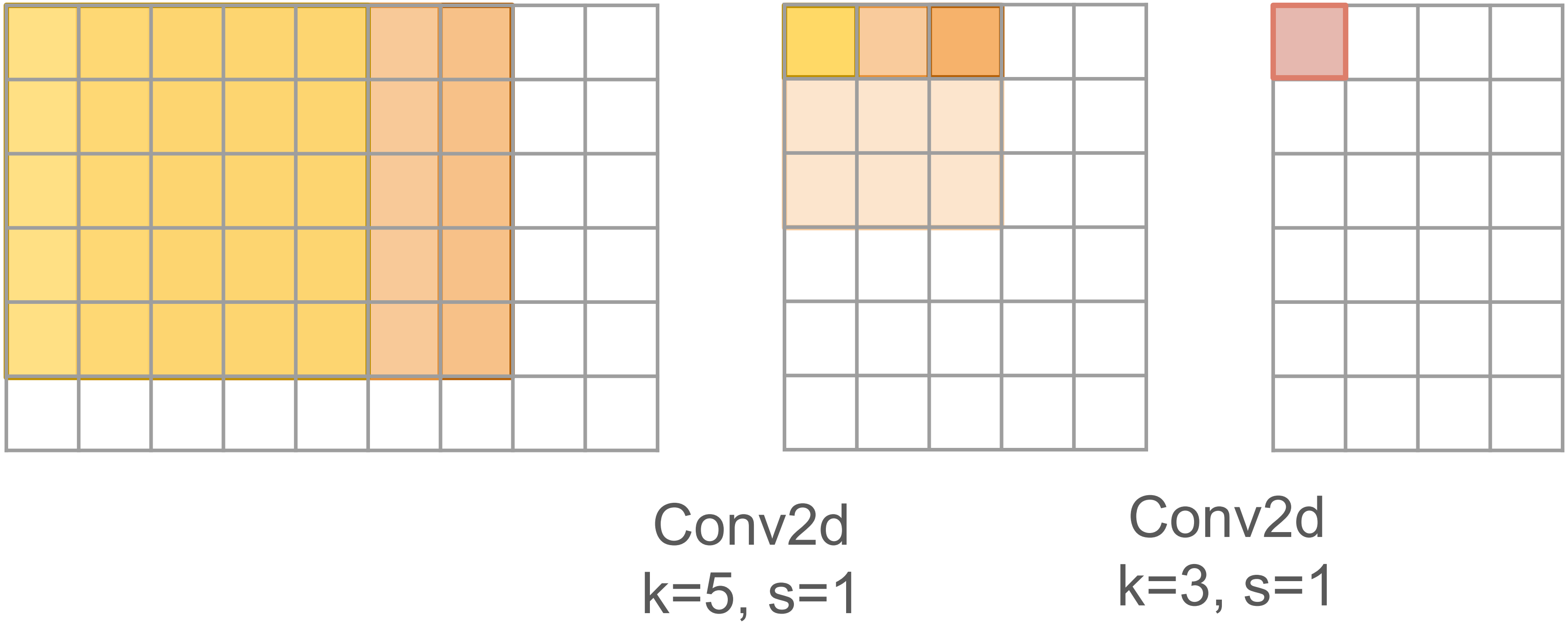
Conv2d
k=3, s=1

Receptive field size

k=5, s=1, m=3

$$n = k + s(m-1)$$

n=7



Case Study: VGGNet

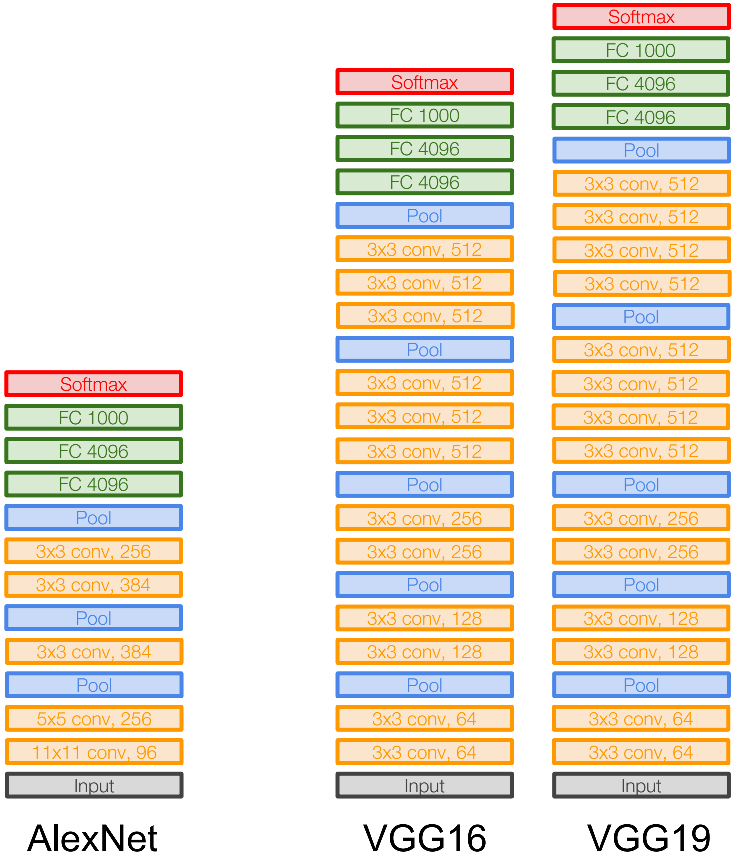
[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer



3.2 Convolutional Architectures

Consider the convolutional network defined by the layers in the left column below. Fill in the size of the activation volumes at each layer, and the number of parameters at each layer. You can write your answer as a multiplication (e.g. 128x128x3).

- CONV5-N denotes a convolutional layer with N filters, each of size 5x5xD, where D is the depth of the activation volume at the previous layer. Padding is 2, and stride is 1.
- POOL2 denotes a 2x2 max-pooling layer with stride 2 (pad 0)
- FC-N denotes a fully-connected layer with N output neurons.

Layer	Activation Volume Dimensions (memory)	Number of parameters
INPUT	32x32x1	0
CONV5-10	32x32x10	10x(25+1)
POOL2	16x16x10	0

5x5x1

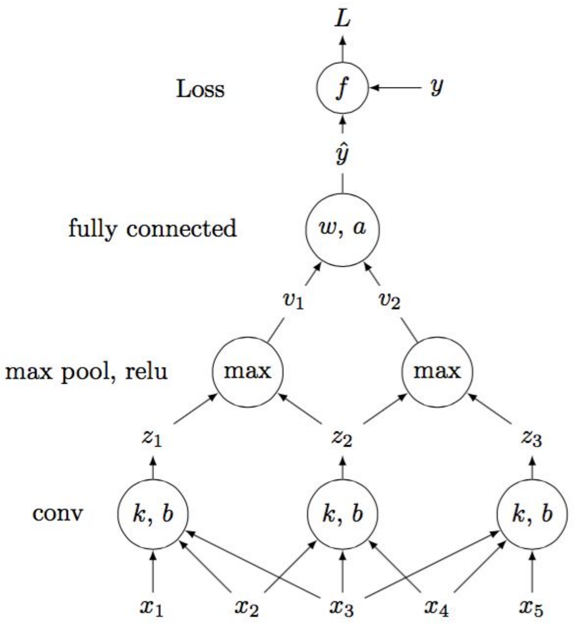
$\frac{1}{1}(32+4-5)+1=32$

For kernel width **k** and stride **s**,
Input width $\mathbf{w}_{in}$ and total padding $\mathbf{w}_{pad}$,
Output width $\mathbf{w}_{out}$ is

$$w_{out} = \frac{1}{s}(w_{in} + w_{pad} - k) + 1$$

3.3 Simple ConvNet (12 points)

Consider the following 1-dimensional ConvNet, where all variables are scalars:



$$L = \frac{1}{2}(y - \hat{y})^2$$

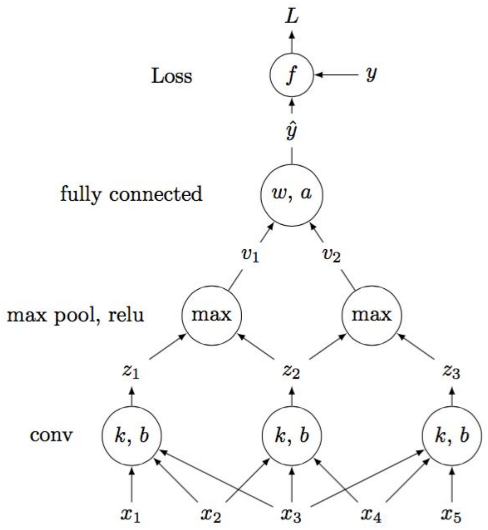
$$\hat{y} = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + a$$

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \max\{z_1, z_2, 0\} \\ \max\{z_2, z_3, 0\} \end{bmatrix}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix}$$

3.3 Simple ConvNet (12 points)

Consider the following 1-dimensional ConvNet, where all variables are scalars:



$$L = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y} = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + a$$

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \max\{z_1, z_2, 0\} \\ \max\{z_2, z_3, 0\} \end{bmatrix}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix}$$

(c) **(3 points)** Given the gradients of the loss L with respect to the second layer activations v, derive the gradient of the loss with respect to the first layer activations z. More precisely, given

$$\frac{\partial L}{\partial v_1} = \delta_1 \qquad \frac{\partial L}{\partial v_2} = \delta_2$$

Determine the following

$$\frac{\partial L}{\partial z_1} =$$

Chain Rule

$$x \;,\; y \;,\; z \text{ with}$$

$$y = f(x)$$
$$z = g(y)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Chain Rule?

$$x \ , y_1, y_2, \ldots y_n \ , z \text{ with}$$

$$y_i = f_i(x)$$
$$z = g(y_1, \ldots y_n)$$

Chain Rule!

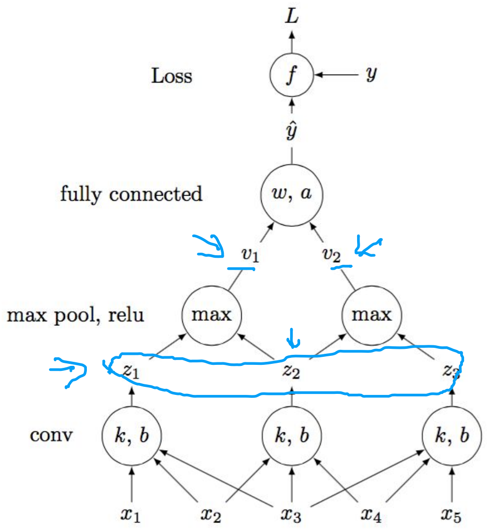
$$x \ , y_1, y_2, \dots y_n \ , z \text{ with}$$

$$\begin{aligned} y_i &= f_i(x) \\ z &= g(y_1, \dots y_n) \end{aligned}$$

$$\frac{\partial z}{\partial x} = \sum \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

3.3 Simple ConvNet (12 points)

Consider the following 1-dimensional ConvNet, where all variables are scalars:



$$L = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y} = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + a$$

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \max\{z_1, z_2, 0\} \\ \max\{z_2, z_3, 0\} \end{bmatrix}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix}$$

$x\ ,\ y_1, y_2, \ldots y_n\ ,\ z$ with

$$y_i = f_i(x)$$
$$z = g(y_1, \ldots y_n)$$

$$\frac{\partial z}{\partial x} = \sum \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$

$L = g(v_1, v_2)$

$v_1 = f_1(z_1, z_2)$

$v_2 = f_2(z_2, z_3)$

$$\frac{\partial L}{\partial z_2} = \delta_1 \frac{\partial v_1}{\partial z_2} + \delta_2 \frac{\partial v_2}{z_2}$$

$$v_1 = \begin{cases} z_1 & z_1 > z_2 \quad z_1 > 0 \\ z_2 & \boxed{z_2 > z_1 \quad z_2 > 0} \\ 0 & \end{cases}$$

$$\frac{\partial}{z_2} \rightarrow \begin{cases} 0 \\ 1 \\ 0 \end{cases} \qquad \mathbb{1}(z_2 > z_1 \,\&\, z_2 > 0)$$

(c) **(3 points)** Given the gradients of the loss L with respect to the second layer activations v, derive the gradient of the loss with respect to the first layer activations z. More precisely, given

$$\boxed{\frac{\partial L}{\partial v_1} = \delta_1 \qquad \frac{\partial L}{\partial v_2} = \delta_2}$$

Determine the following

$$\frac{\partial L}{\partial z_1} =$$

$$\frac{\partial L}{\partial z_2}$$

$\mathbb{1}(x > y)$

$\delta_1 \mathbb{1}$

1. You start training your Neural Network but the total loss (cross entropy loss + regularization loss) is almost completely flat from the start. What could be the cause?

   (a) The learning rate could be too low

   (b) The regularization strength could be too high

   (c) The class distribution could be very uneven in the dataset

   (d) The weight initialization scale could be incorrectly set

1. You start training your Neural Network but the total loss (cross entropy loss + regularization loss) is almost completely flat from the start. What could be the cause?

    (a) The learning rate could be too low

    (b) The regularization strength could be too high

    (c) The class distribution could be very uneven in the dataset

    (d) The weight initialization scale could be incorrectly set

1. You start training your Neural Network but the total loss (cross entropy loss + regularization loss) is almost completely flat from the start. What could be the cause?

   (a) The learning rate could be too low

   (b) ~~The regularization strength could be too high~~

   (c) The class distribution could be very uneven in the dataset

   (d) The weight initialization scale could be incorrectly set

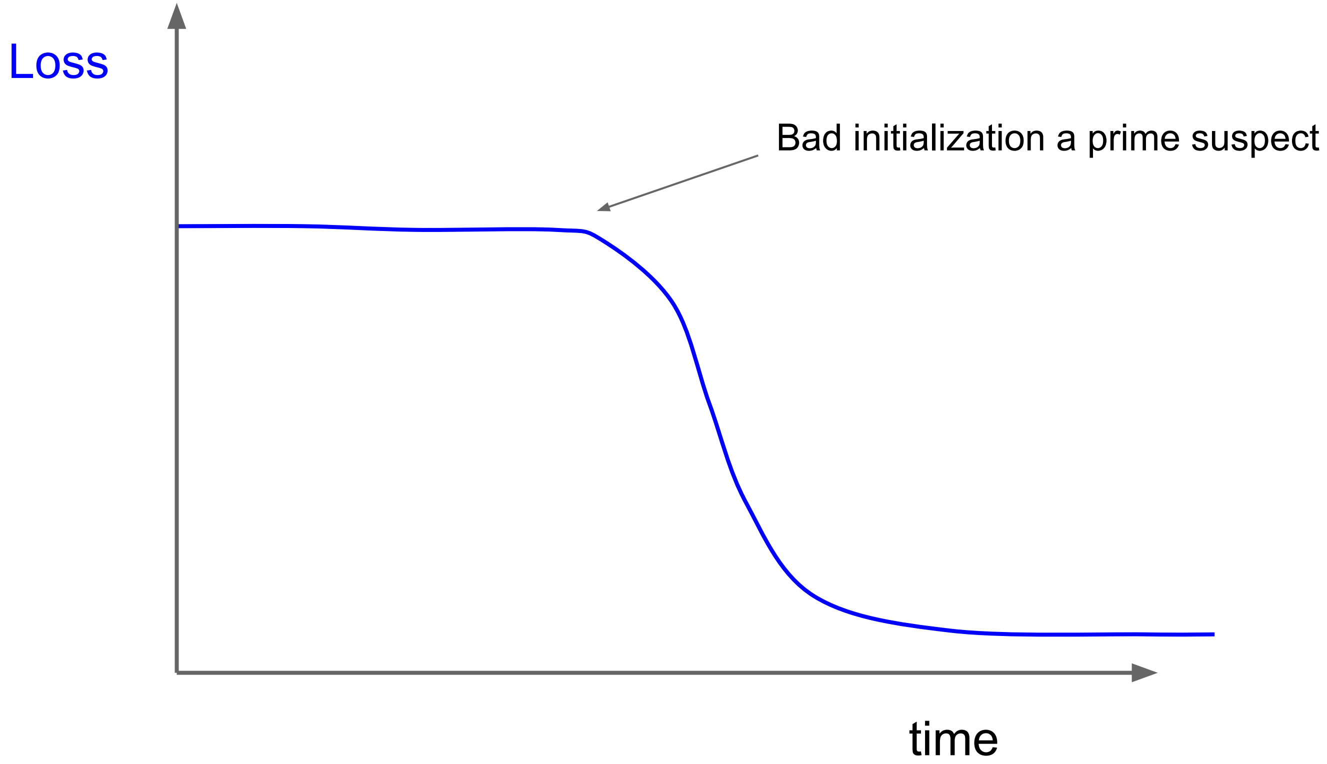
1. You start training your Neural Network but the total loss (cross entropy loss + regularization loss) is almost completely flat from the start. What could be the cause?

   (a) The learning rate could be too low

   (b) ~~The regularization strength could be too high~~

   (c) ~~The class distribution could be very uneven in the dataset~~

   (d) The weight initialization scale could be incorrectly set


Loss
Bad initialization a prime suspect
time

1. You start training your Neural Network but the total loss (cross entropy loss + regularization loss) is almost completely flat from the start. What could be the cause?

   (a) The learning rate could be too low

   (b) ~~The regularization strength could be too high~~

   (c) ~~The class distribution could be very uneven in the dataset~~

   (d) The weight initialization scale could be incorrectly set

3. A max pooling layer in a ConvNet:

   (a) Is approximately as fast to compute in both forward and backward pass as a CONV layer (with the same filter size and strides).

   (b) Is similar to batch normalization in that it will keep all of your neuron activities in a similar range.

   (c) Could contribute to difficulties during gradient checking (higher error than usual, as in the SVM).

   (d) Could contribute to the vanishing gradient problem (recall: this is a problem where by the end of a backward pass the gradients are very small)

3. A max pooling layer in a ConvNet:

   (a) ~~Is approximately as fast to compute in both forward and backward pass as a CONV layer (with~~ the same filter size and strides).

   (b) Is similar to batch normalization in that it will keep all of your neuron activities in a similar range.

   (c) Could contribute to difficulties during gradient checking (higher error than usual, as in the SVM).

   (d) Could contribute to the vanishing gradient problem (recall: this is a problem where by the end of a backward pass the gradients are very small)

3. A max pooling layer in a ConvNet:

   (a) ~~Is approximately as fast to compute in both forward and backward pass as a CONV layer (with~~ the same filter size and strides).

   (b) ~~Is similar to batch normalization in that it will keep all of your neuron activities in a similar~~ range.

   (c) Could contribute to difficulties during gradient checking (higher error than usual, as in the SVM).

   (d) Could contribute to the vanishing gradient problem (recall: this is a problem where by the end of a backward pass the gradients are very small)

3. A max pooling layer in a ConvNet:

    (a) ~~Is approximately as fast to compute in both forward and backward pass as a CONV layer (with~~ the same filter size and strides).

    (b) ~~Is similar to batch normalization in that it will keep all of your neuron activities in a similar~~ range.

    (c) Could contribute to difficulties during gradient checking (higher error than usual, as in the SVM).

    (d) Could contribute to the vanishing gradient problem (recall: this is a problem where by the end of a backward pass the gradients are very small)

3. A max pooling layer in a ConvNet:

(a) ~~Is approximately as fast to compute in both forward and backward pass as a CONV layer (with~~ the same filter size and strides).

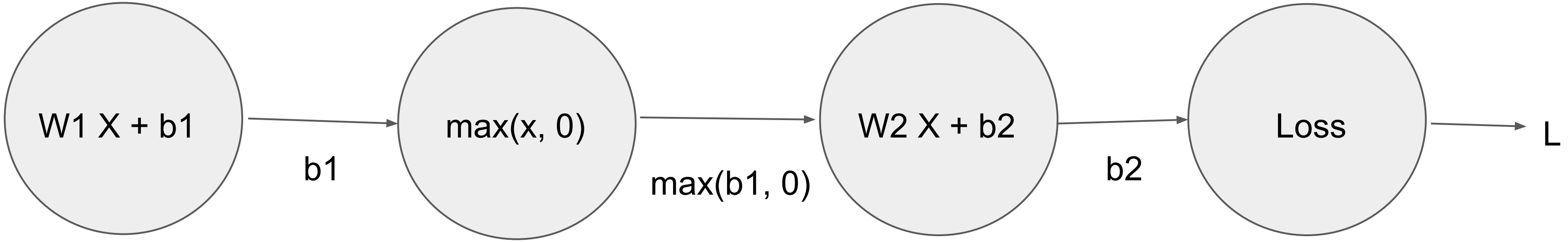
(b) ~~Is similar to batch normalization in that it will keep all of your neuron activities in a similar~~ range.

(c) Could contribute to difficulties during gradient checking (higher error than usual, as in the SVM).

(d) Could contribute to the vanishing gradient problem (recall: this is a problem where by the end of a backward pass the gradients are very small)

4. **True or False:** It's sufficient for symmetry breaking (i.e. there will be no symmetry in the weights after some updates) in a Neural Network to initialize all weights to 0, provided that the biases are random

Symmetry Breaking



9. **True or False:** If a neuron with the ReLU activation function ($y = relu(Wx + b)$) receives input x that is all zero, then the final (not local!) gradient on its weights and biases will also be zero (i.e. none of its parameters will update at all).

11. **True or False:** One reason that the centered difference formula for the finite difference approximation of the gradient is preferable to the uncentered alternative is because it is better at avoiding kinks (non differentiabilities) in the objective. Recall: the centered formula is $f'(x) = (f(x+h) - f(x-h))/2h$ instead of $f'(x) = (f(x+h) - f(x))/h$.