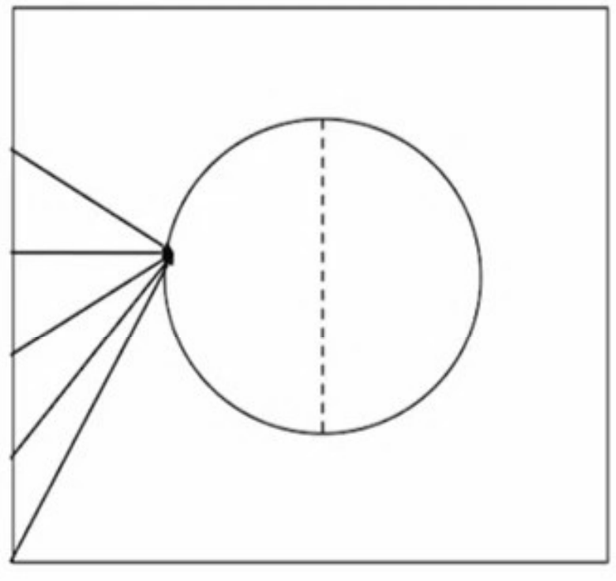
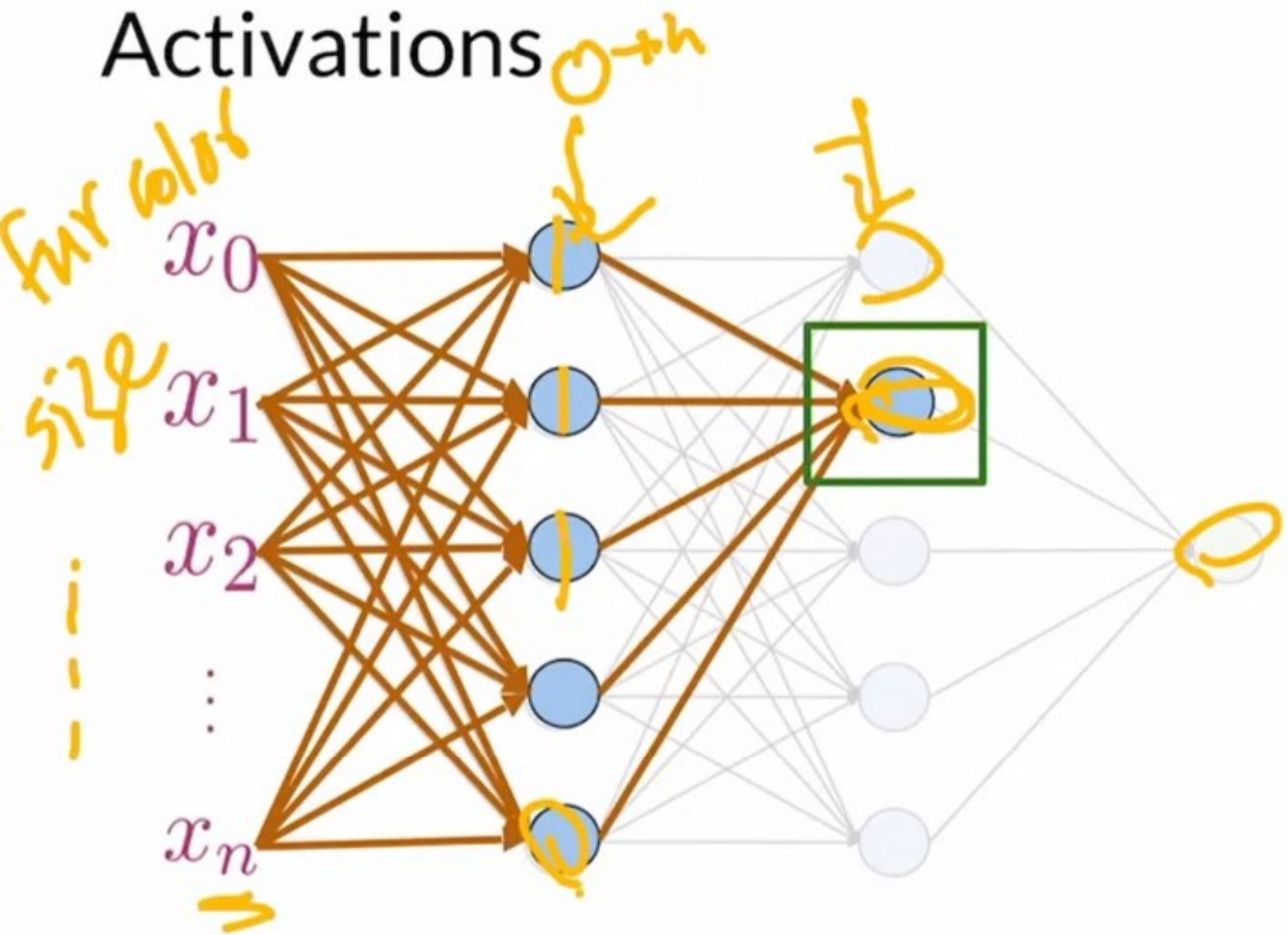

# Activations (Basic Properties)

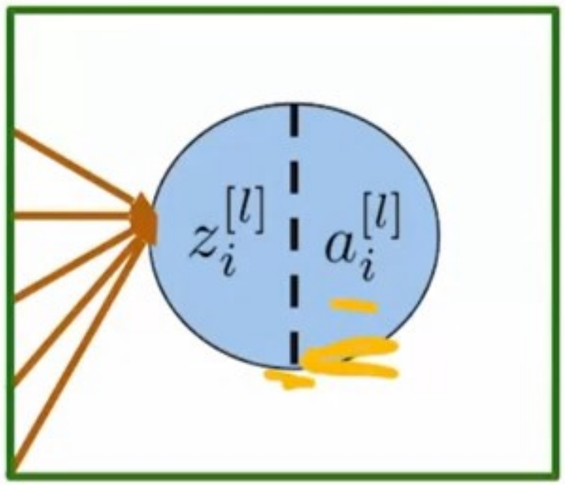
# Outline

- What are activations
- Reasoning behind non-linear differential activations



# Activations




$$z_i^{[l]} = \sum_{i=0}^{n} W_i^{[l]} a_i^{[l-1]}$$

$$a_i^{[l]} = g^{[l]}(z_i^{[l]})$$

Differentiable non-linear function

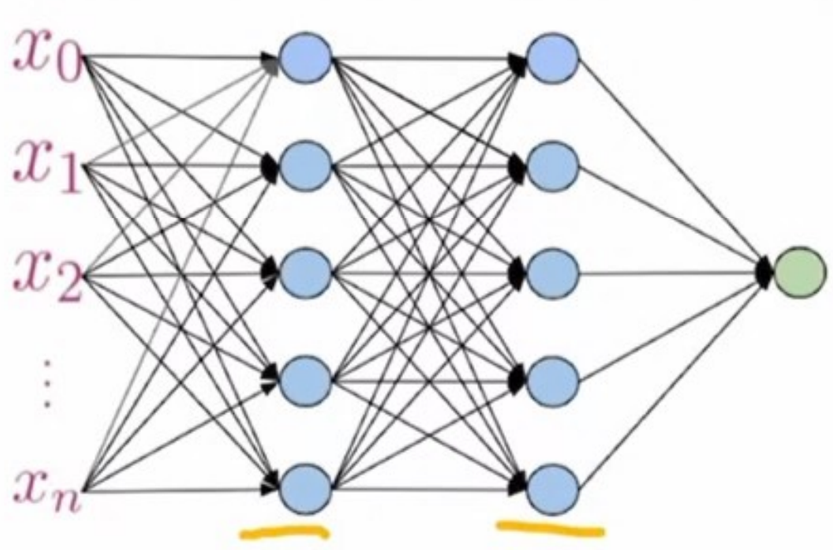
# Activations

$$a_i^{[l]} = g^{[l]}(z_i^{[l]})$$

Differentiable non-linear function

1. Differentiable for backpropagation

2. Non-linear to compute complex features, **if not**:




$\equiv$

$$WX + b$$

Linear regression

# Summary

- Activation functions are non-linear and differentiable
- Differentiable for backpropagation
- Non-linear to approximate complex functions

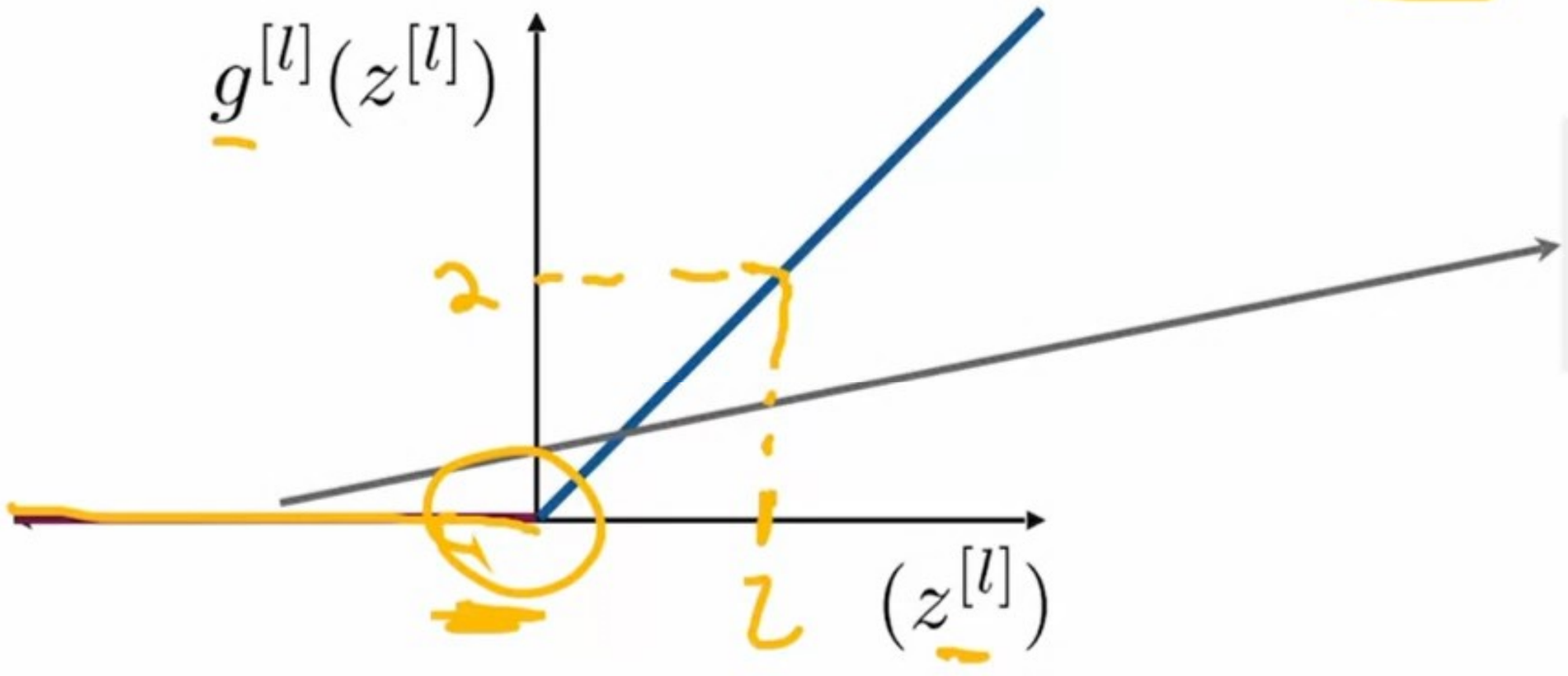

# Common Activation Functions

# Outline

- Common activations and their structure
  - ReLU
  - Leaky ReLU
  - Sigmoid
  - Tanh

# Activations: ReLU

ReLU = Rectified Linear Unit

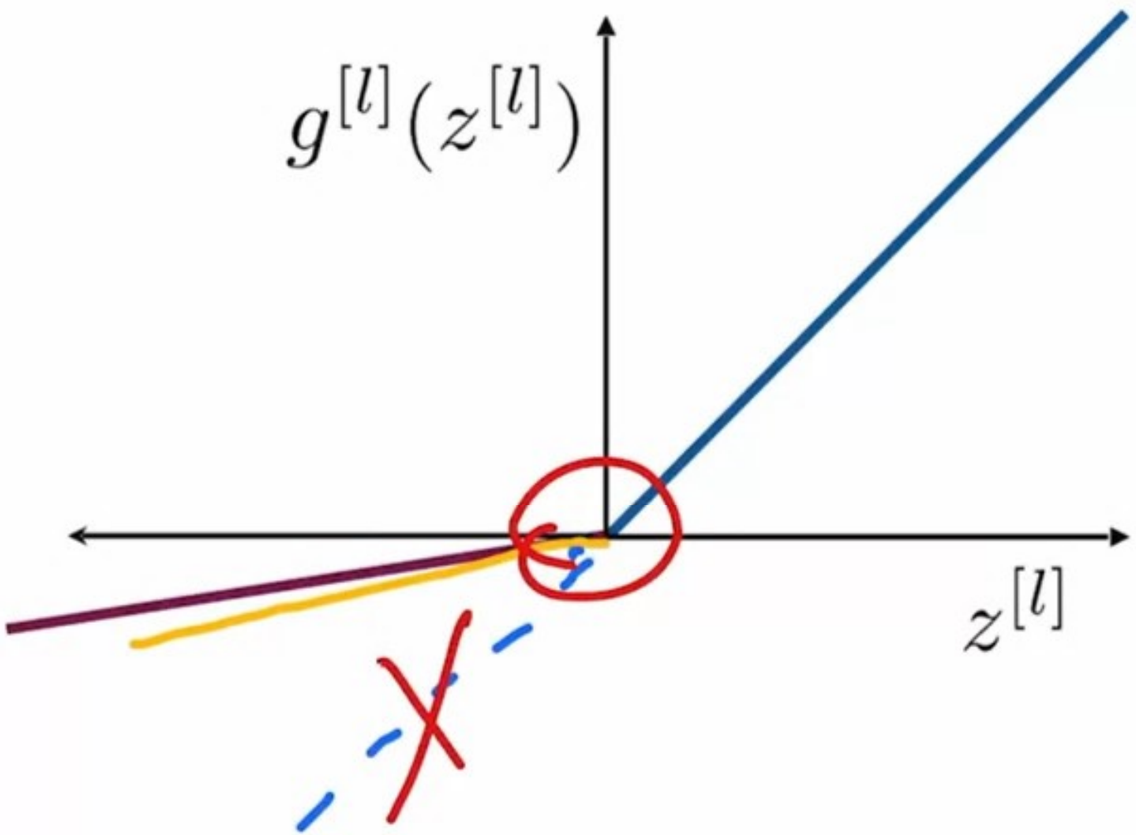



ReLU

$$g^{[l]}(z^{[l]}) = \max(0, z^{[l]})$$

Dying ReLU problem

# Activations: Leaky ReLU



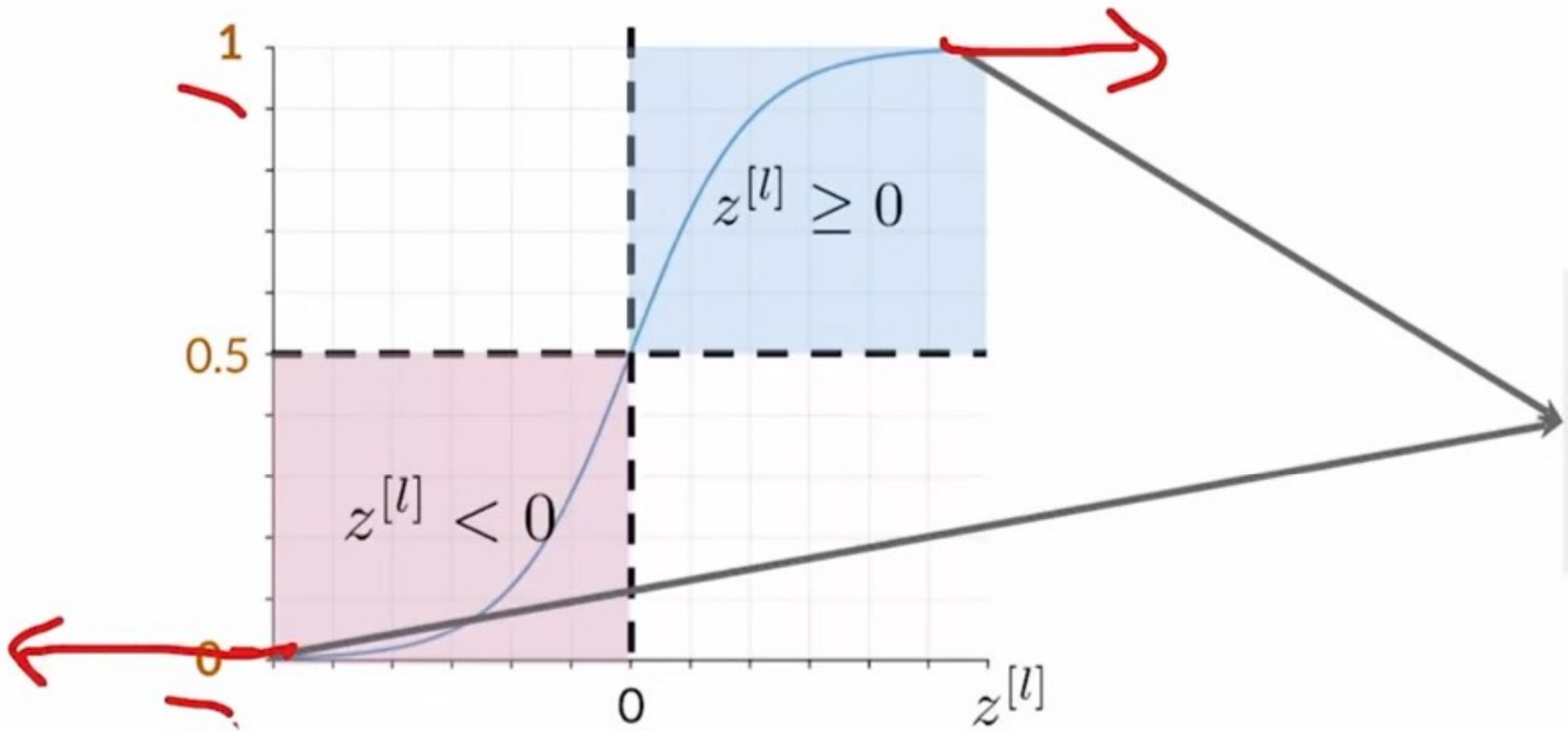

$$g^{[l]}(z^{[l]}) = max(\underline{a}z^{[l]}, \underline{z^{[l]}})$$

0.1

Solves the dying ReLU problem

# Activations: Sigmoid

$$g^{[l]}(z^{[l]}) = \frac{1}{1 + e^{-z^{[l]}}}$$

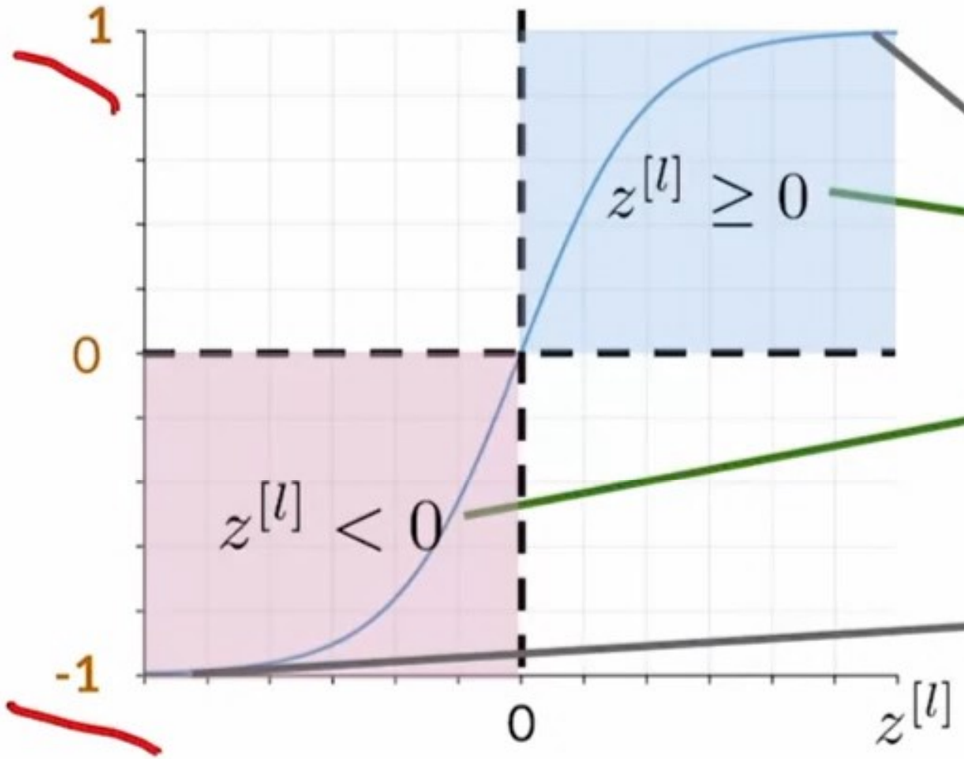



Values between 0 and 1

Vanishing gradient and saturation problems

# Activations: Tanh

$$g^{[l]}(z^{[l]}) = tanh(z^{[l]})$$




Values between -1 and 1

Keeps the sign of the input

Same issues as Sigmoid

# Summary

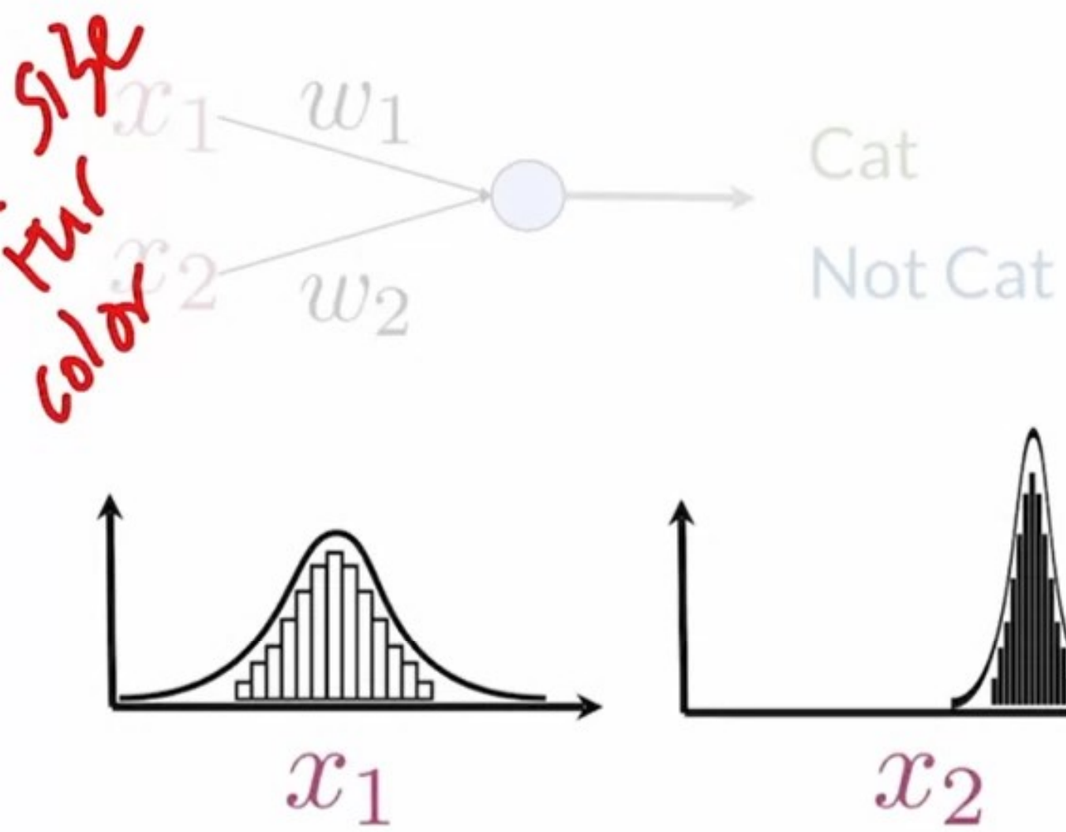
- ReLU activations suffer from dying ReLU
- Leaky ReLU solve the dying ReLu problem
- Sigmoid and Tanh have vanishing gradient and saturation problems


# Batch Normalization (Explained)

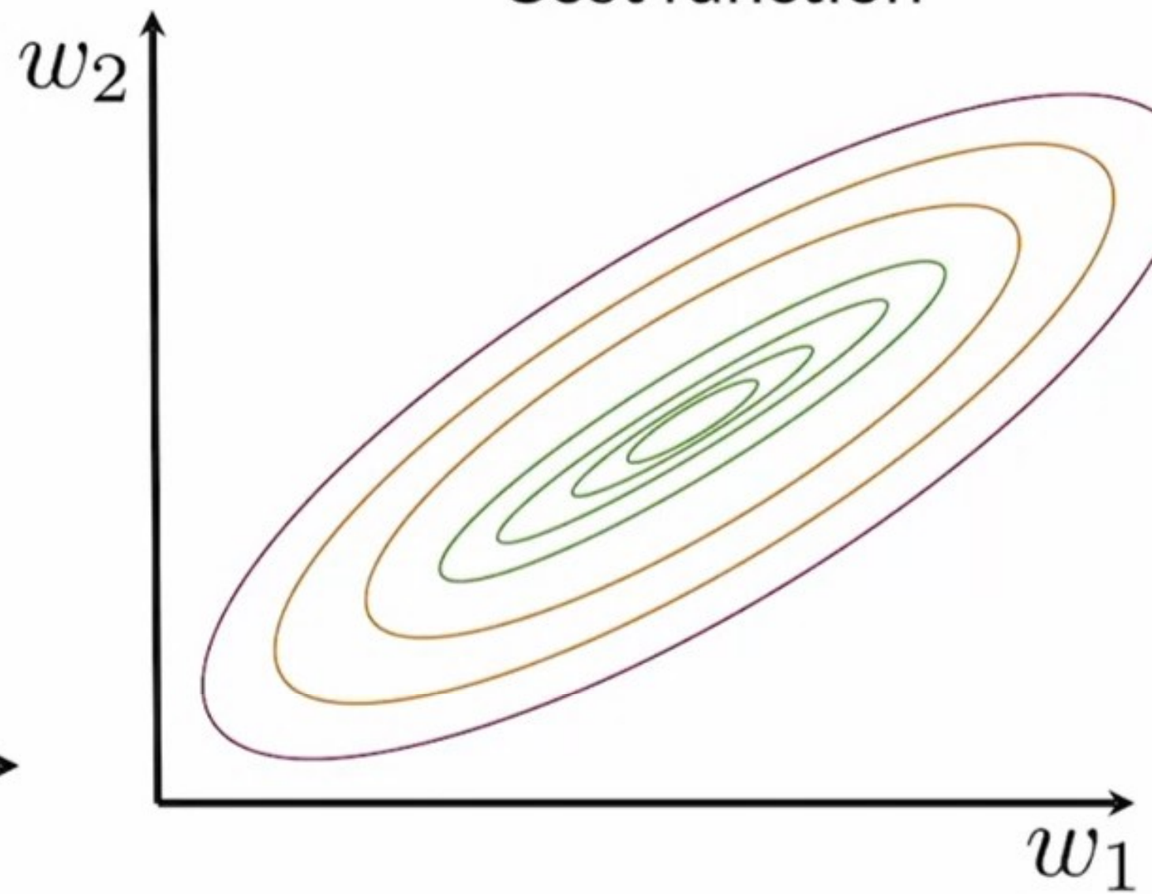
# Outline

- How normalization helps models
- Internal covariate shift
- Batch normalization

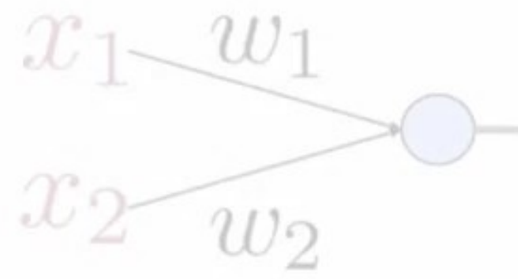
# Different Distributions




# Cost function



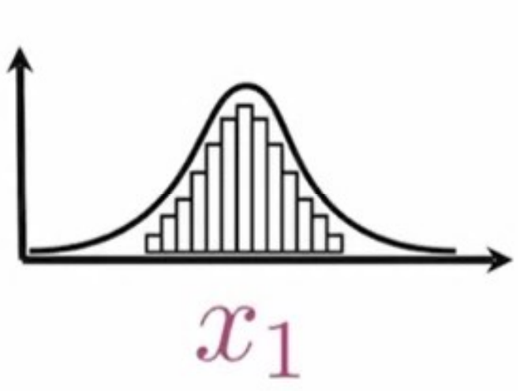
# Covariate Shift

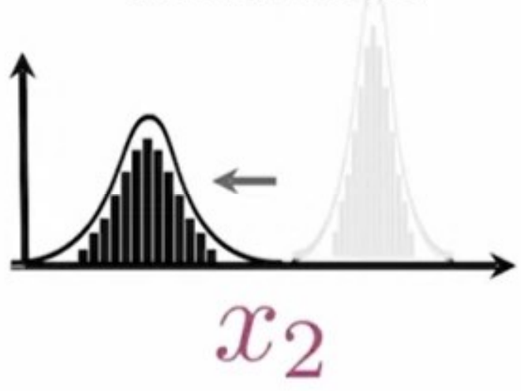



Cat

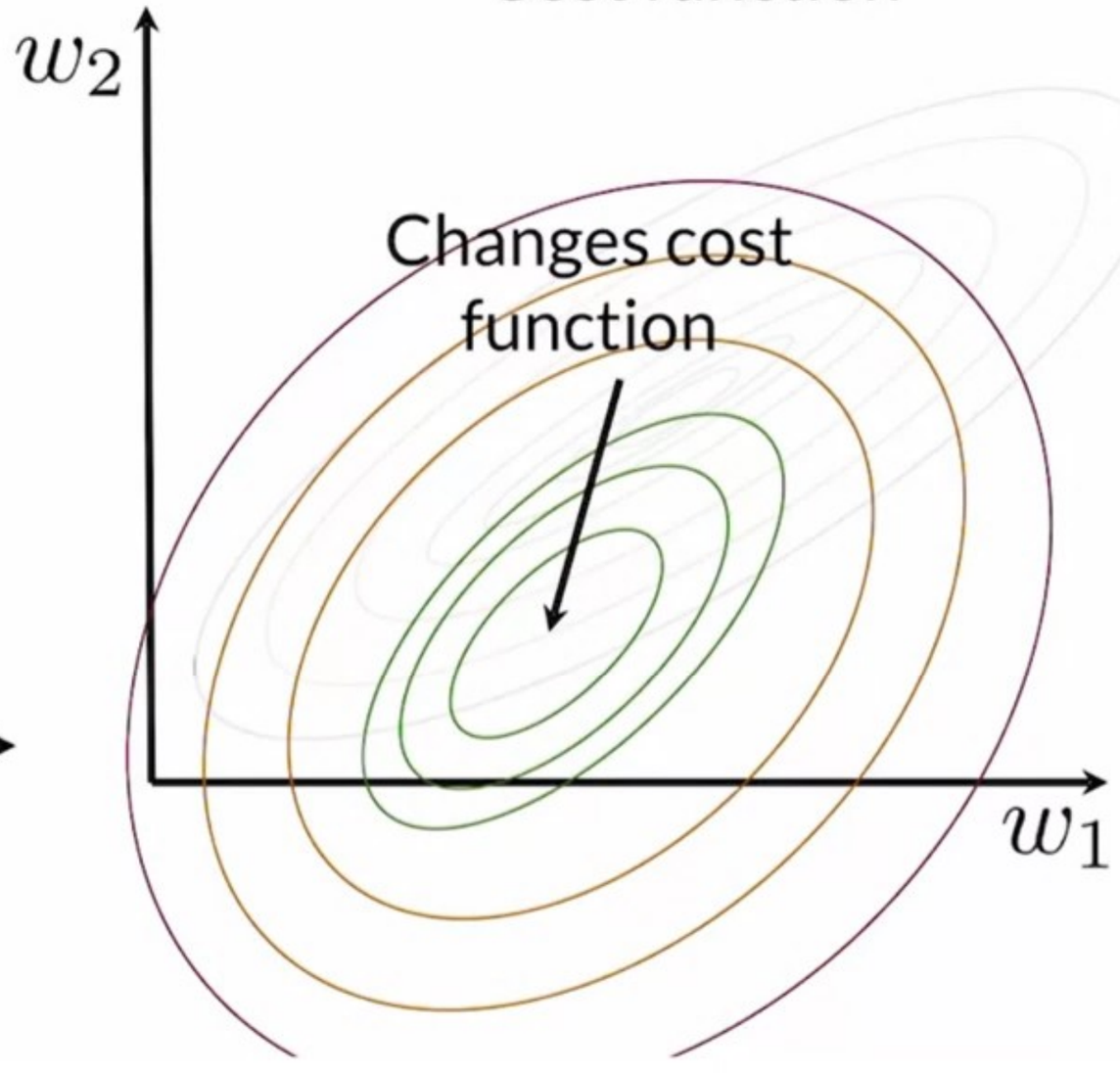
Not Cat

Change in data distribution

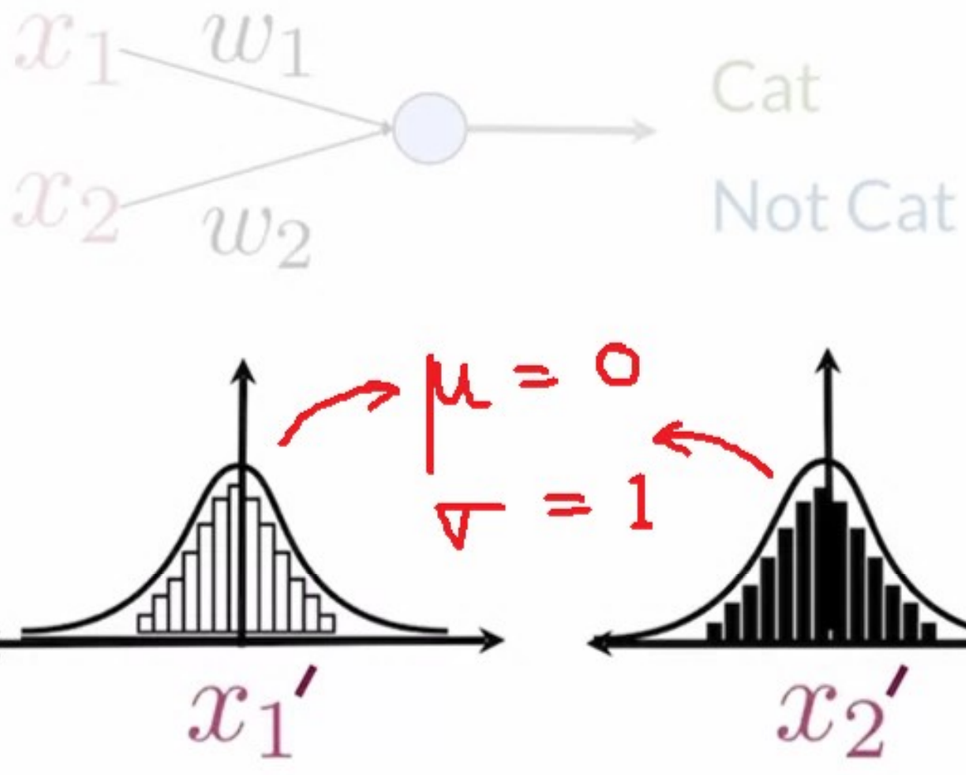


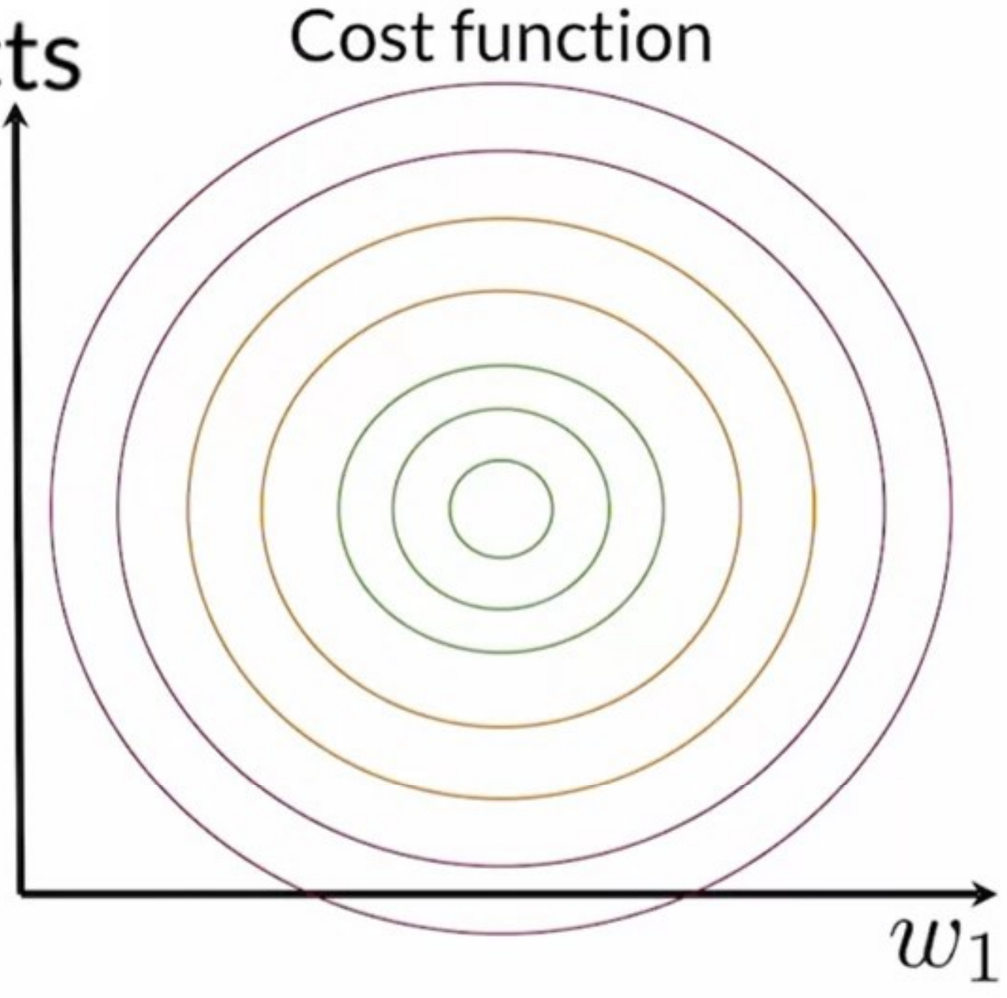




Cost function

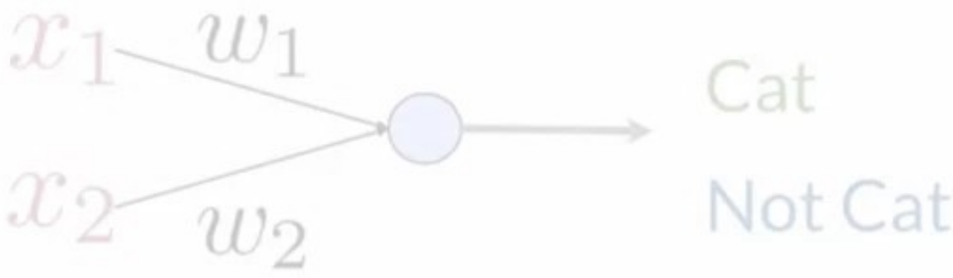


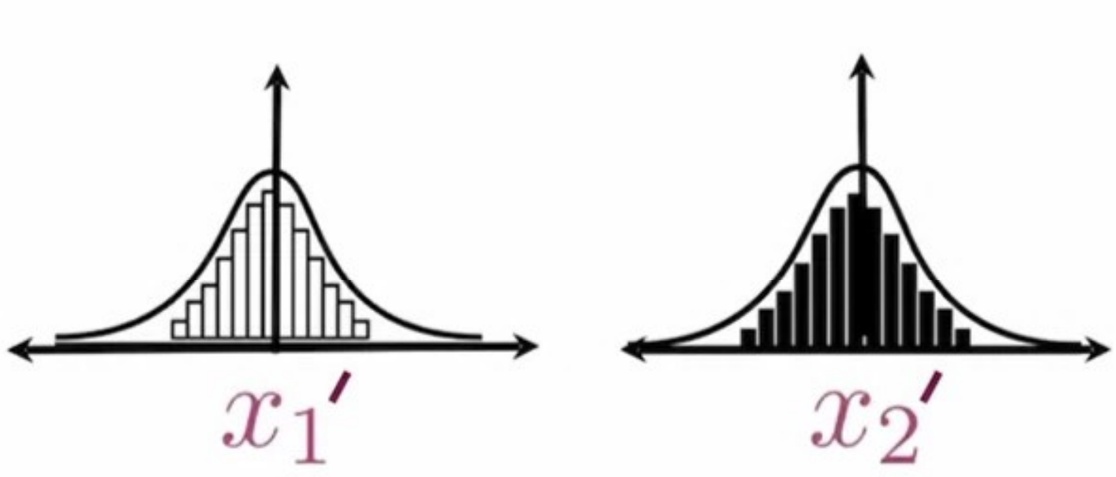
# Normalization and Its Effects



# Normalization and Its Effects



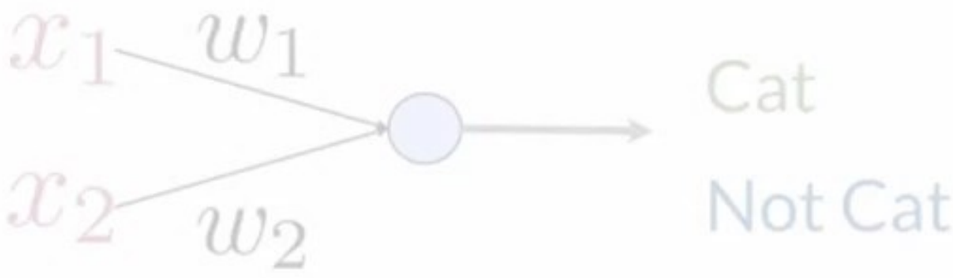


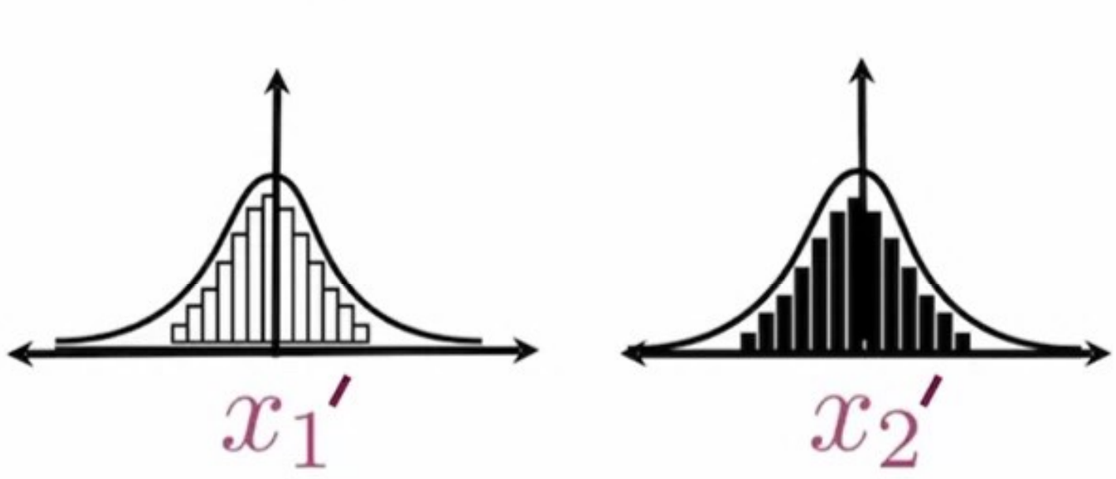
**Around** mean at 0
and std. at 1

**Training data** uses batch stats

**Test data** uses training stats

# Normalization and Its Effects



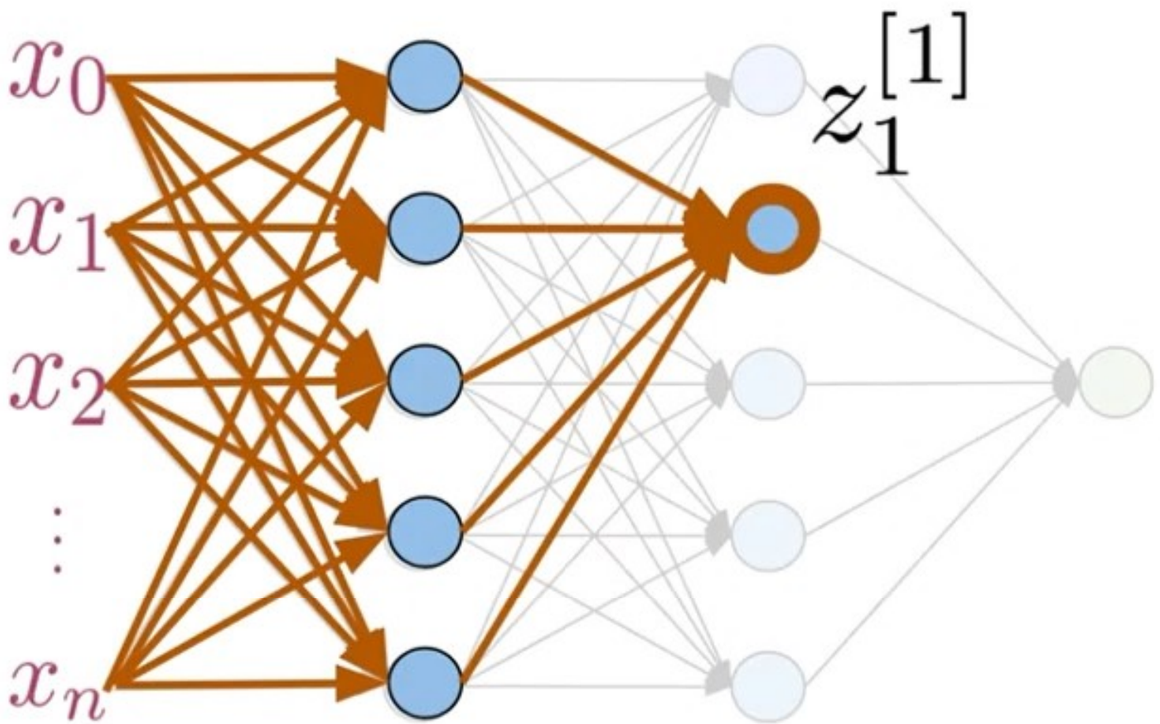


**Around** mean at 0
and std. at 1

↓

Reduction of
covariate shift

# Internal Covariate Shift

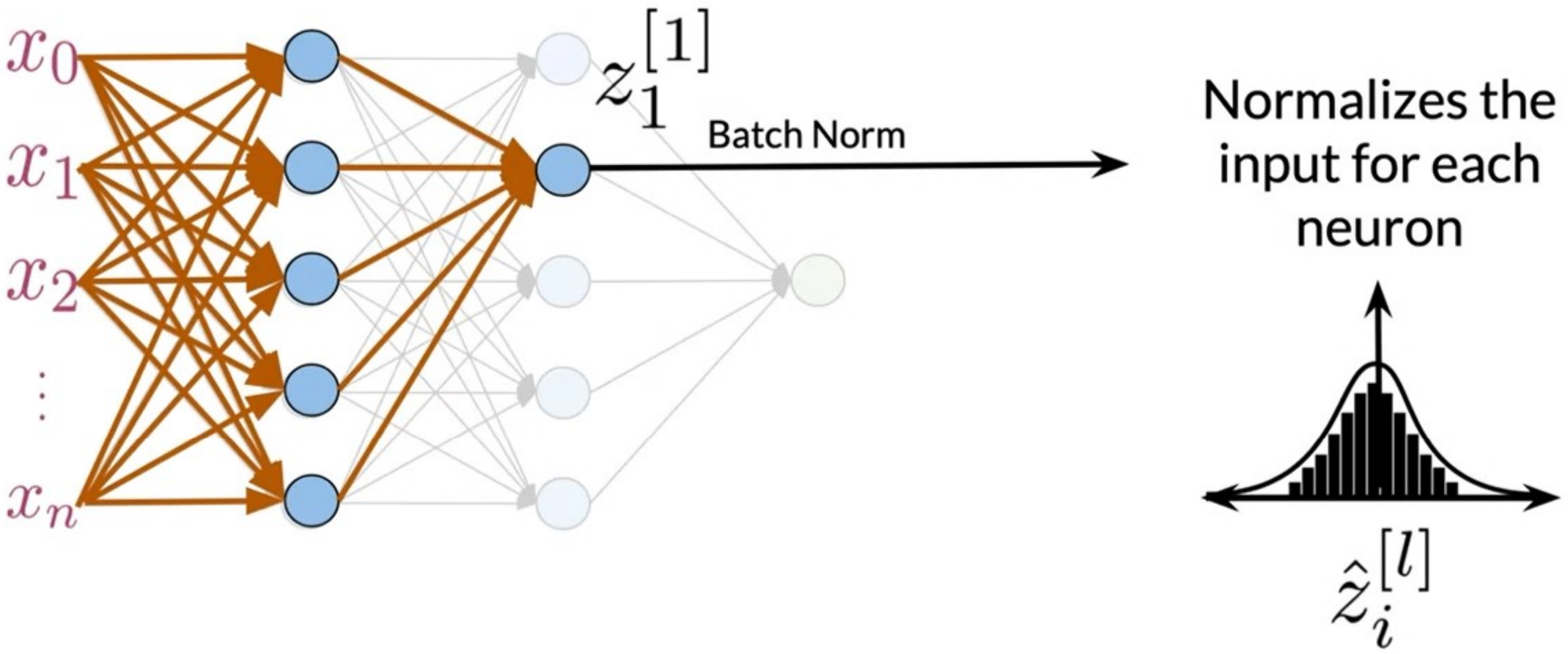



Changes in weights

↓

Changes in activation distribution

# Batch Normalization



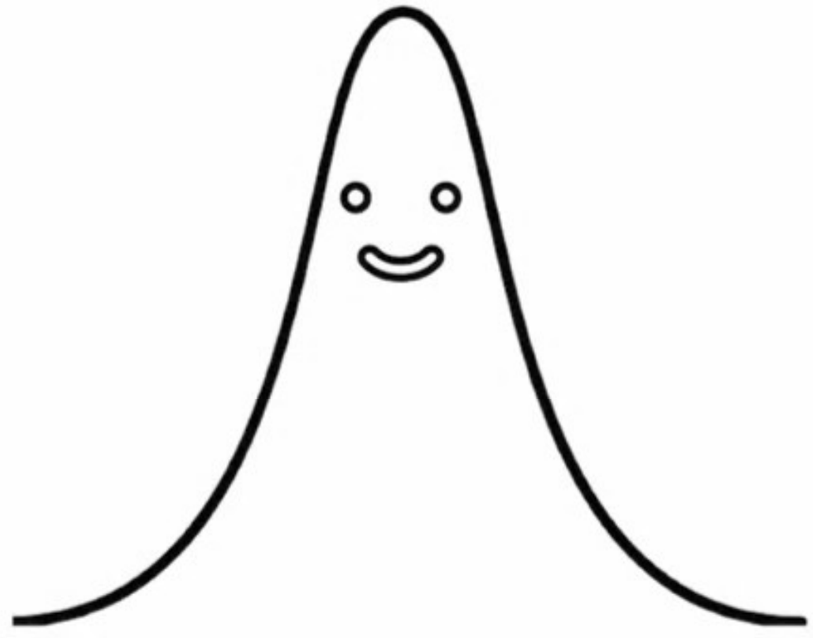
# Summary

- Batch normalization smooths the cost function
- Batch normalization reduces the internal covariate shift
- Batch normalization speeds up learning!

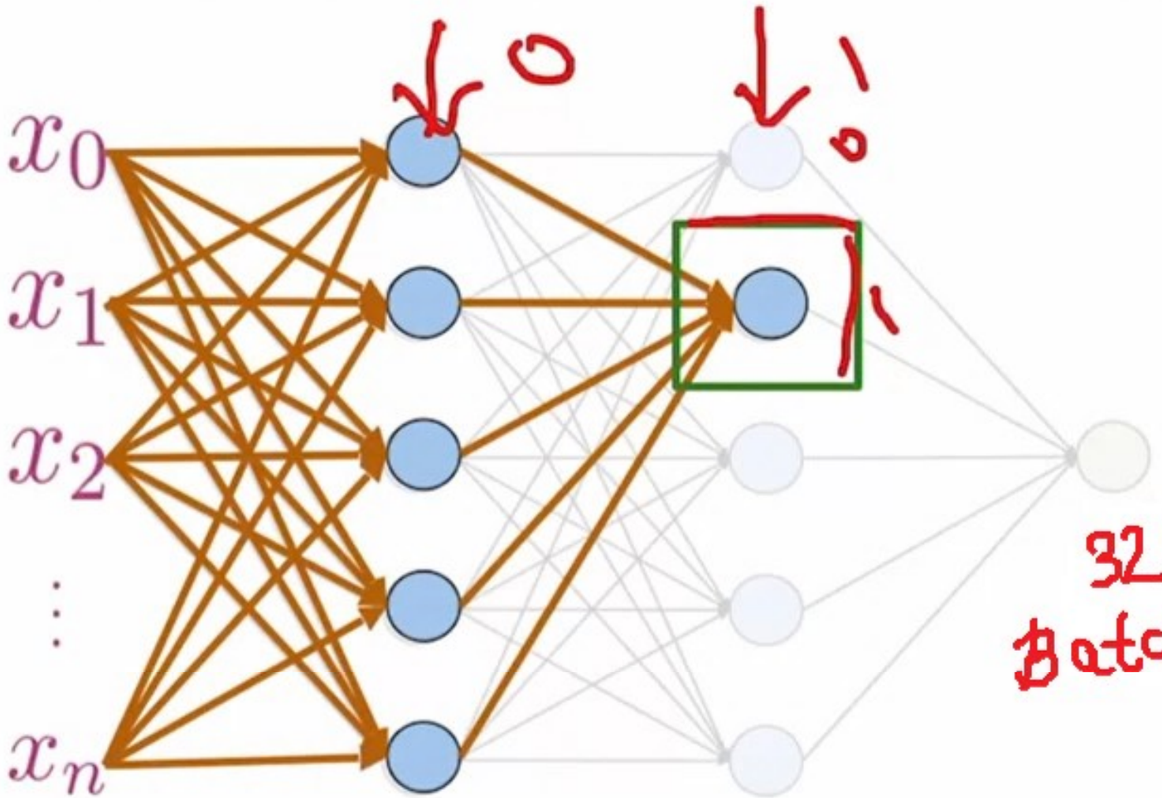

# Batch Normalization (Procedure)

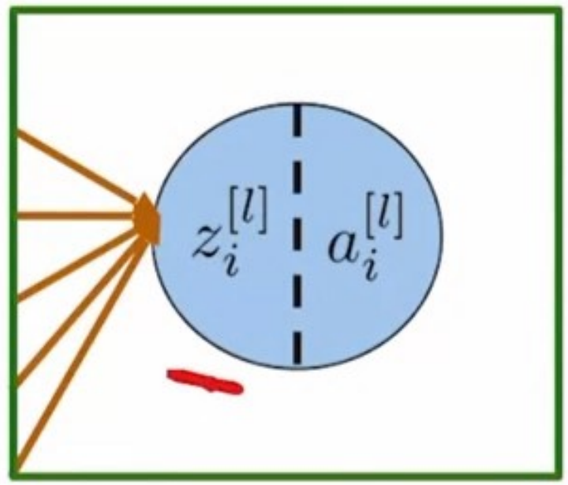
# Outline

- Batch norm for training
- Batch norm for testing



# Batch Normalization: Training



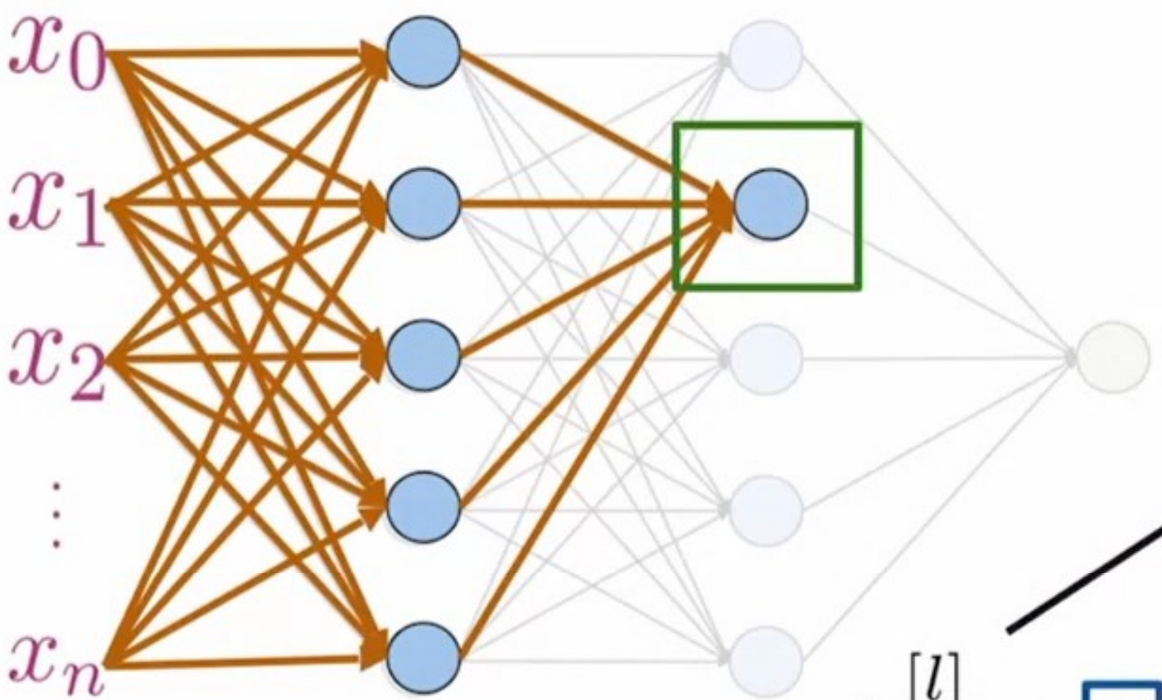

$$z_i^{[l]} = \sum_{i=0} W_i^{[l]} a_i^{[l-1]}$$ → For every example in the batch

$$\hat{z}_i^{[l]} = \frac{z_i^{[l]} - \mu_{z_i^{[l]}}}{\sqrt{\sigma^2_{z_i^{[l]}} + \epsilon}}$$

Batch mean of $z_i^{[l]}$

Batch std of $z_i^{[l]}$

# Batch Normalization: Training







$$\hat{z}_i^{[l]} = \frac{z_i^{[l]} - \mu_{z_i^{[l]}}}{\sqrt{\sigma^2_{z_i^{[l]}} + \epsilon}}$$

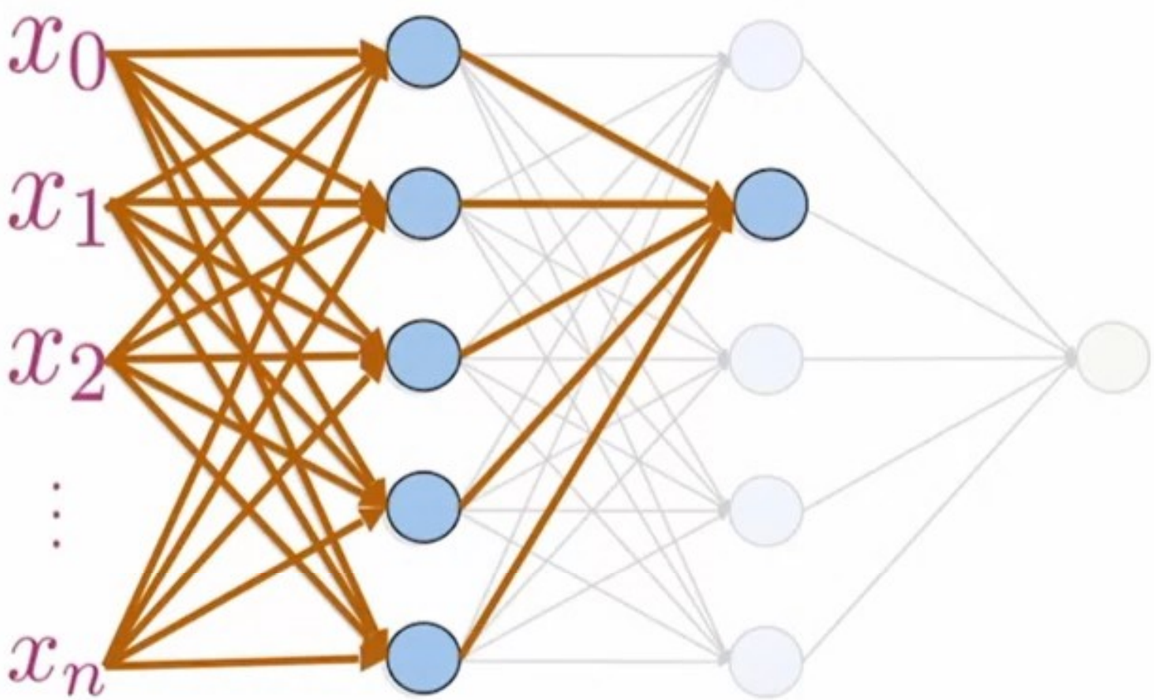
$$y_i^{[l]} = \gamma \hat{z}_i^{[l]} + \beta$$

Shift factor

Scale Factor

Learnable parameters to get the optimal dist.

# Batch Normalization: Test




During testing, mean and variance of training sets are used i.e. they are fixed for each test batches unlike training batches.

$$\hat{z}_i^{[l]} = \frac{z_i^{[l]} - \mathbf{E}(z_i^{[l]})}{\sqrt{\mathrm{Var}(z_i^{[l]}) + \epsilon}}$$

Running mean and running std from training

Frameworks like Tensorflow and Pytorch keep track of them

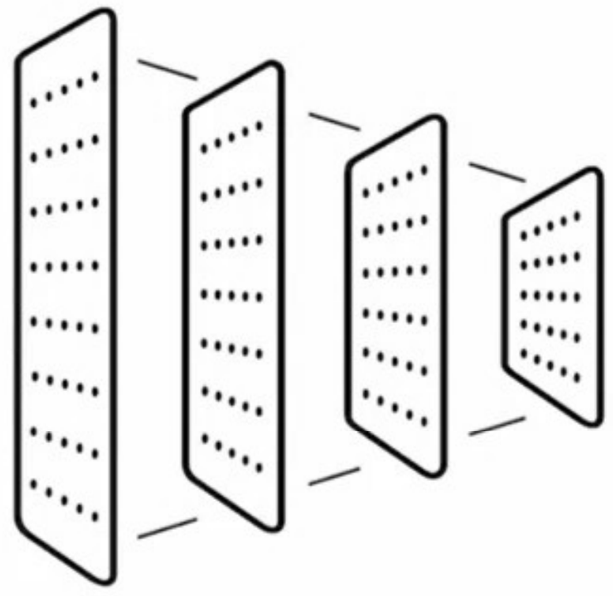
# Summary

- Batch norm introduces learnable shift and scale factors
- During test, the running statistics from training are used
- Frameworks take care of the whole process

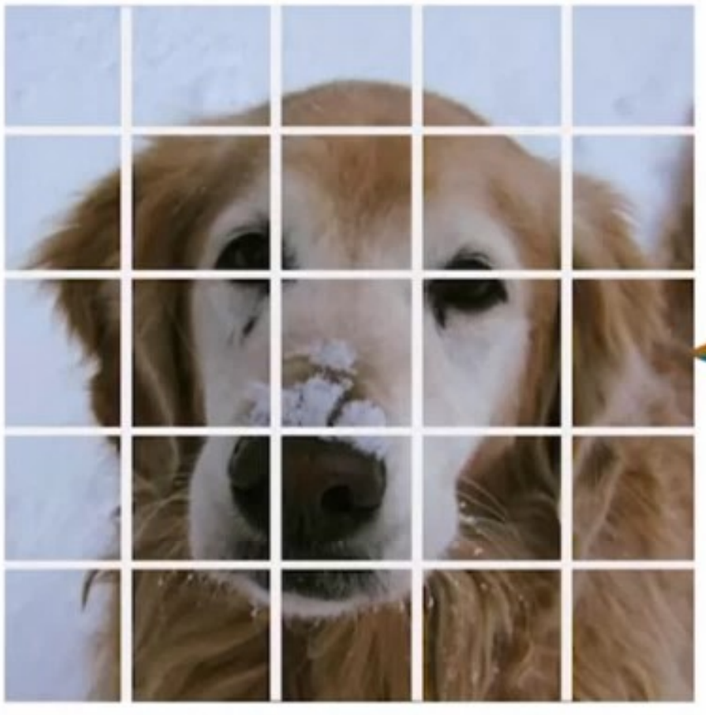

# Review of Convolutions

# Outline

- What convolutions are
- How they work



# What is a convolution?



Eye Filter

| | | |
|---|---|---|
| 5 | 3 | 1 |
| 10 | 23 | 1 |
| 1 | 42 | 4 |

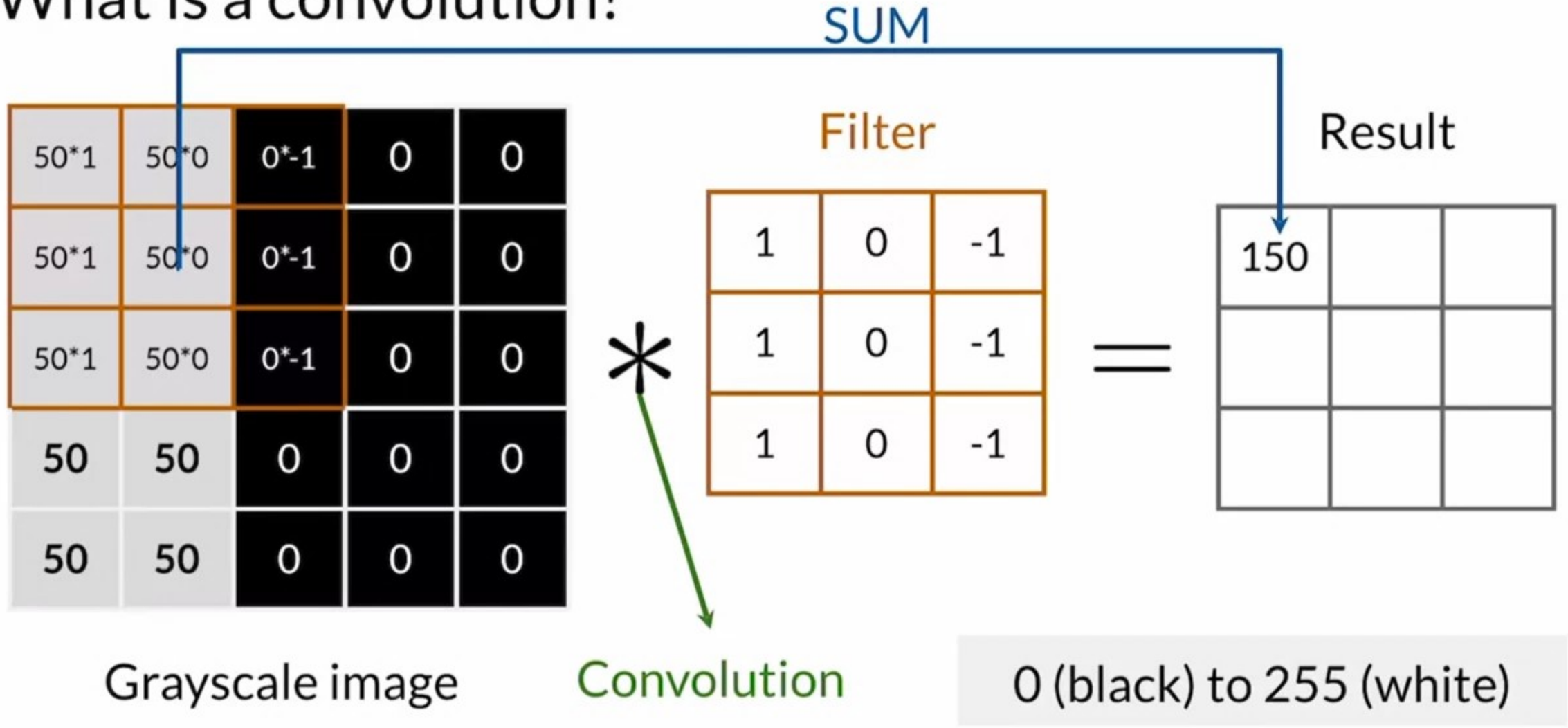
Nose filter

| | | |
|---|---|---|
| 2 | 21 | 5 |
| 23 | 45 | 12 |
| 3 | 32 | 4 |

Ear filter

| | | |
|---|---|---|
| 32 | 2 | 24 |
| 25 | 12 | 66 |
| 3 | 45 | 2 |

# What is a convolution?




Grayscale image

Convolution

0 (black) to 255 (white)

# What is a convolution?

Vertical edge detector

| | | | | |
|---|---|---|---|---|
| 50 | 50 | 0 | 0 | 0 |
| 50 | 50 | 0 | 0 | 0 |
| 50 | 50 | 0 | 0 | 0 |
| 50 | 50 | 0 | 0 | 0 |
| 50 | 50 | 0 | 0 | 0 |

Grayscale image

*

Filter

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

Convolution

=

Result

| | | |
|---|---|---|
| 150 | 150 | 0 |
| 150 | 150 | 0 |
| 150 | 150 | 0 |

0 (black) to 255 (white)

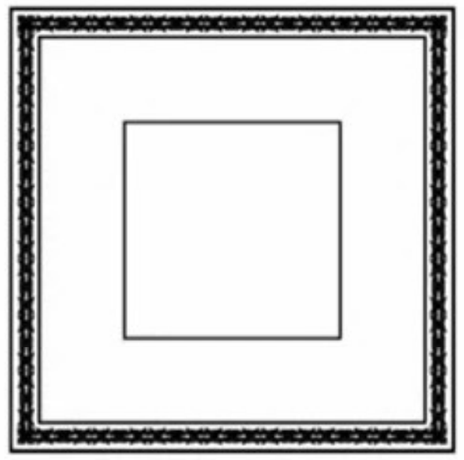
# Summary

- Convolutions are useful layers for processing images
- They scan the image to detect useful features
- Just element-wise products and sums!

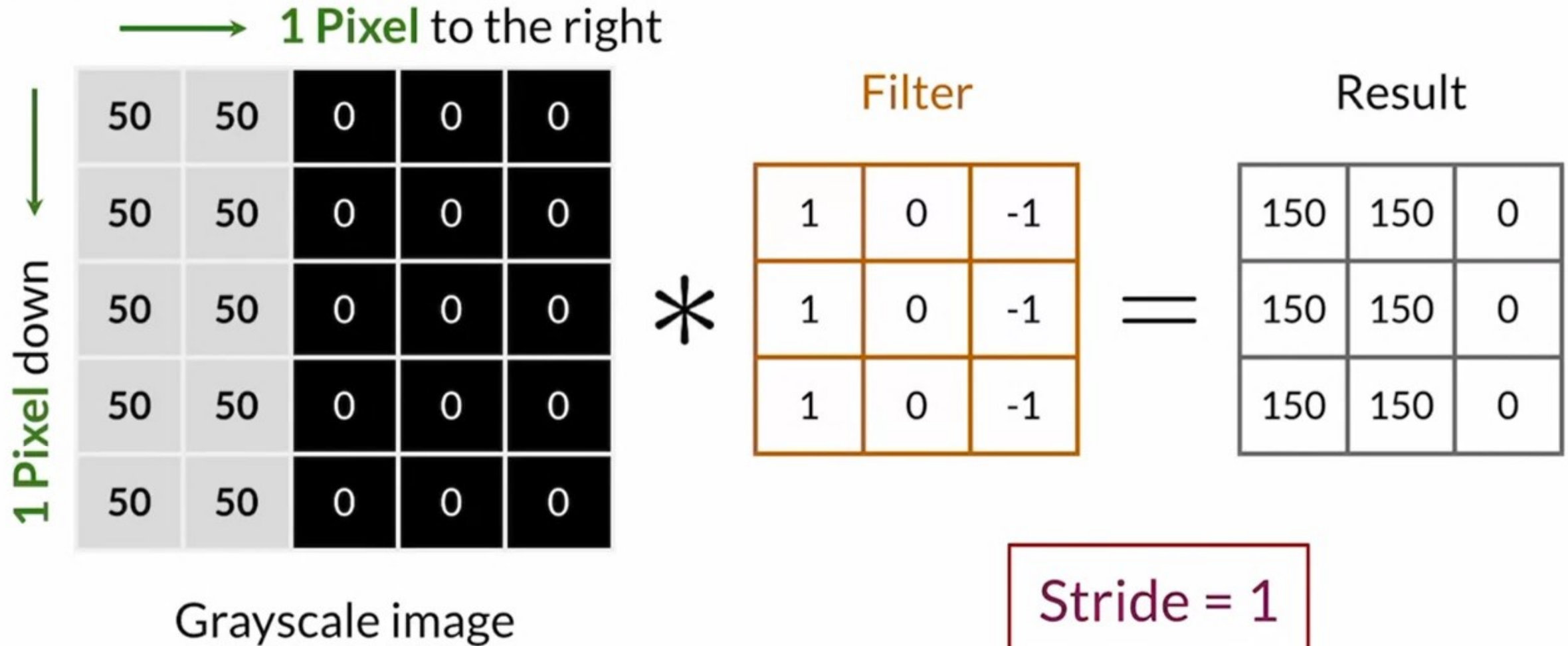

# Padding and Stride

# Outline

- Padding and stride
- The intuition behind padding



# Stride


1 Pixel to the right
1 Pixel down
50 50 0 0 0
50 50 0 0 0
50 50 0 0 0
50 50 0 0 0
50 50 0 0 0
Filter
1 0 -1
1 0 -1
1 0 -1
Result
150 150 0
150 150 0
Grayscale image
Stride = 1

# Stride

→ **2 Pixels** to the right

↓ **2 Pixels** down

| | | | | |
|---|---|---|---|---|
| 50 | 50 | 0 | 0 | 0 |
| 50 | 50 | 0 | 0 | 0 |
| 50 | 50 | 0 | 0 | 0 |
| 50 | 50 | 0 | 0 | 0 |
| 50 | 50 | 0 | 0 | 0 |

Grayscale image

*

Filter

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

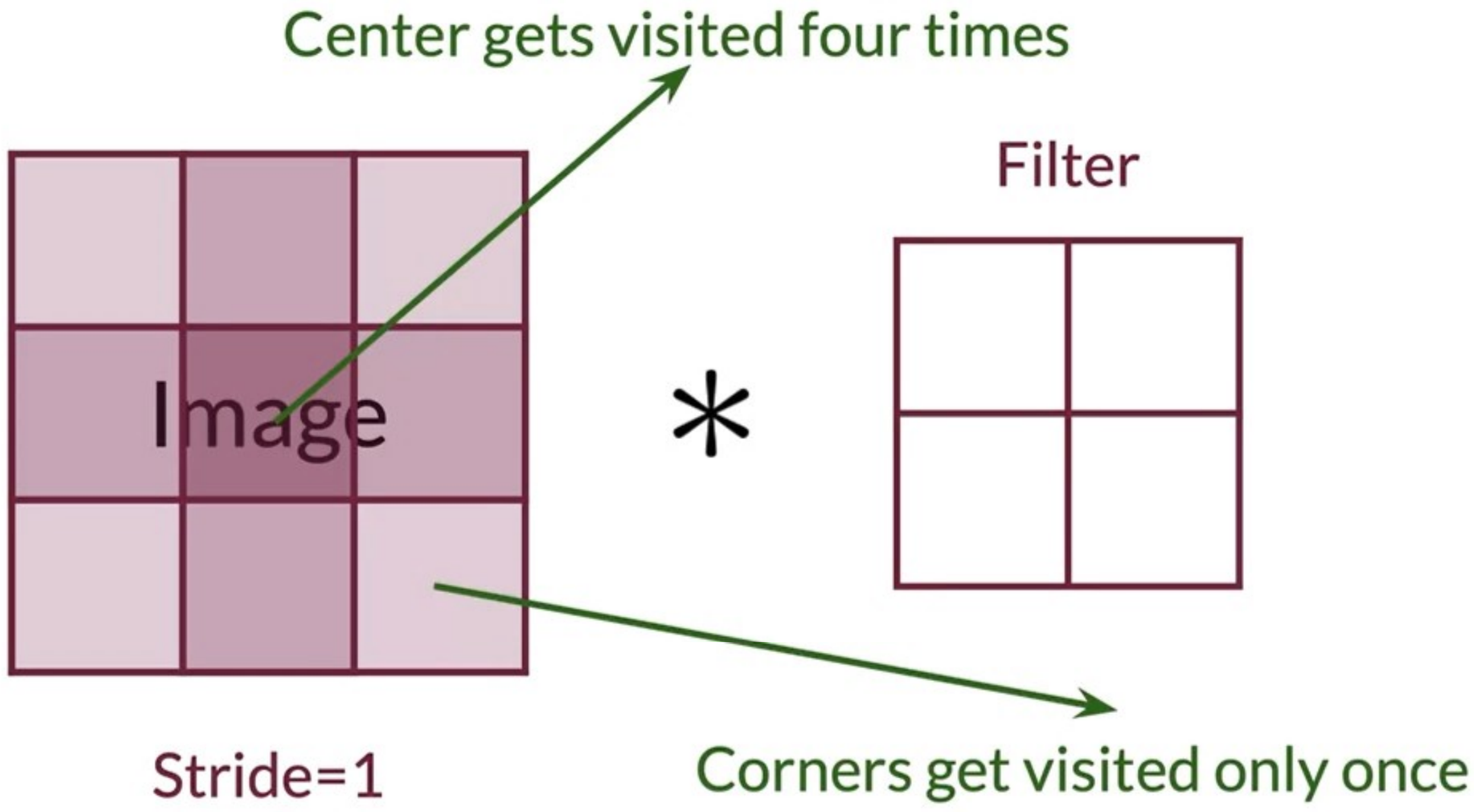
=

Result

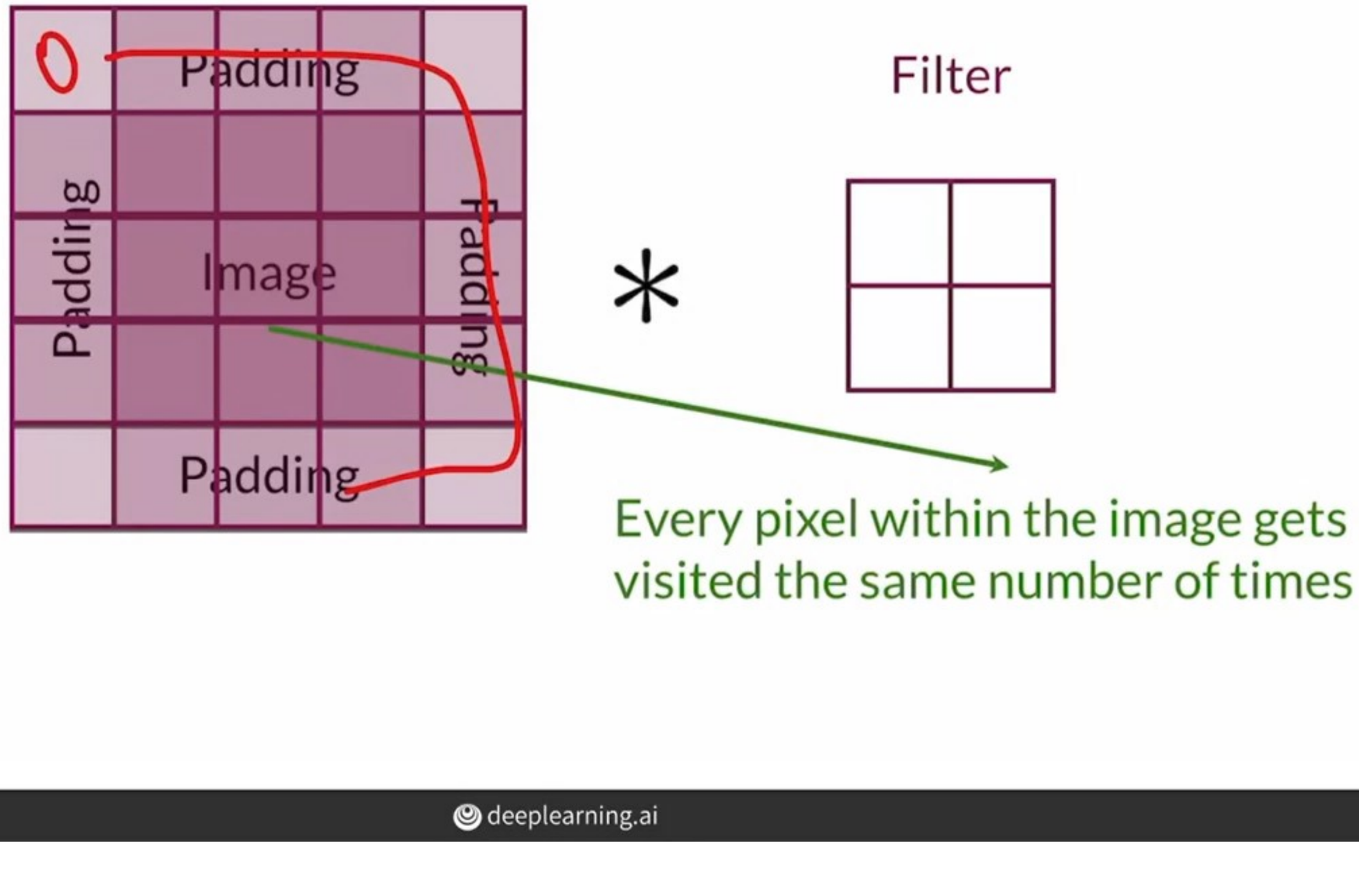
| | |
|---|---|
| 150 | 0 |
| 150 | 0 |

Stride = 2

# Padding


Center gets visited four times
Filter
Image
*
Stride=1
Corners get visited only once

# Padding


0
Padding
Padding
Image
Padding
Padding
Filter
Every pixel within the image gets visited the same number of times

# Summary

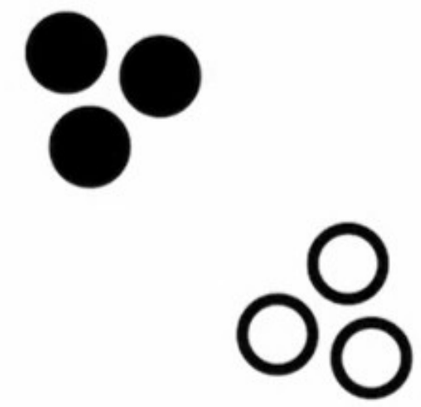
- Stride determines how the filter scans the image
- Padding is like a frame on the image
- Padding gives similar importance to the edges and the center

deeplearning.ai

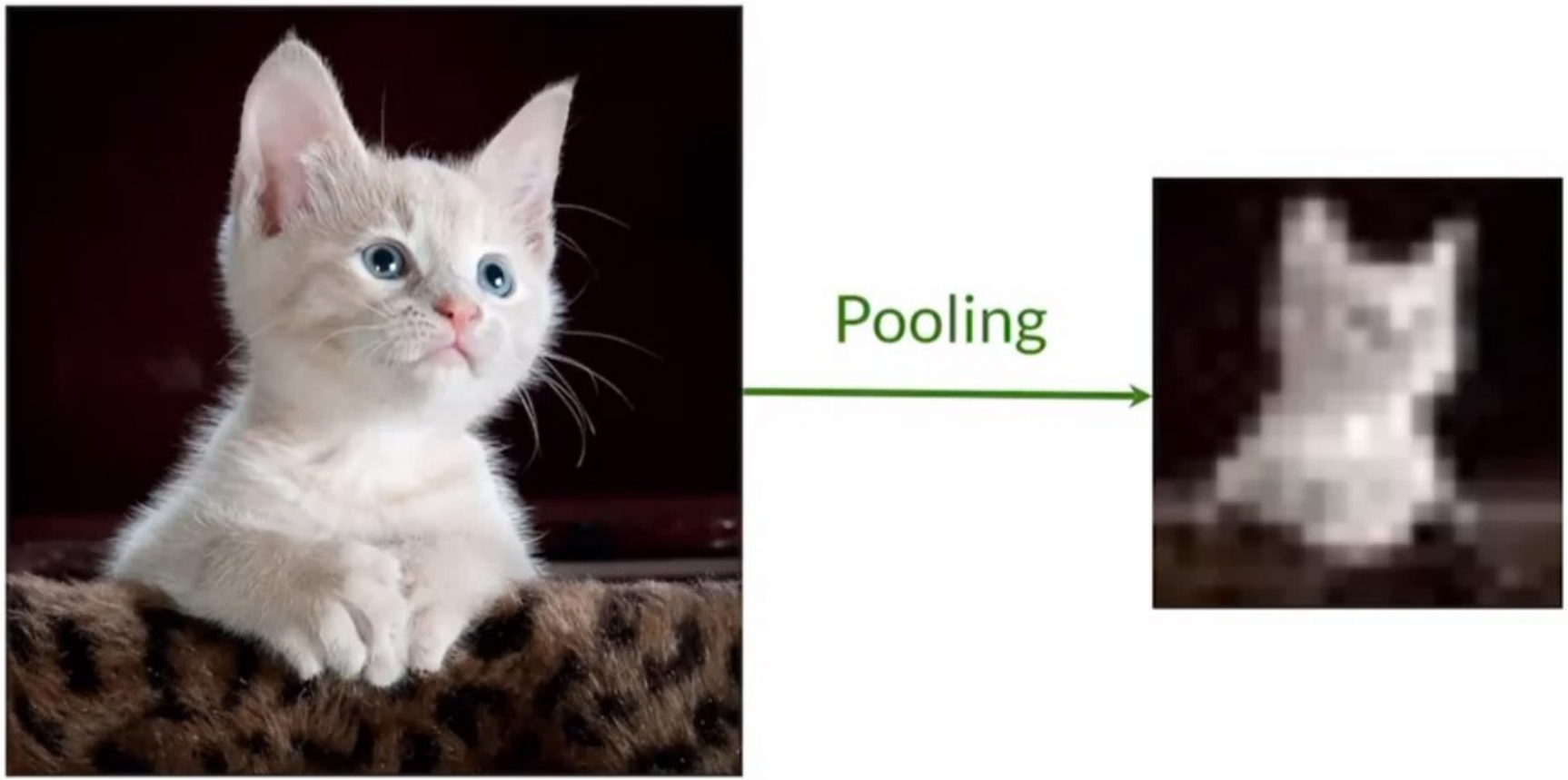
# Pooling and Upsampling

# Outline

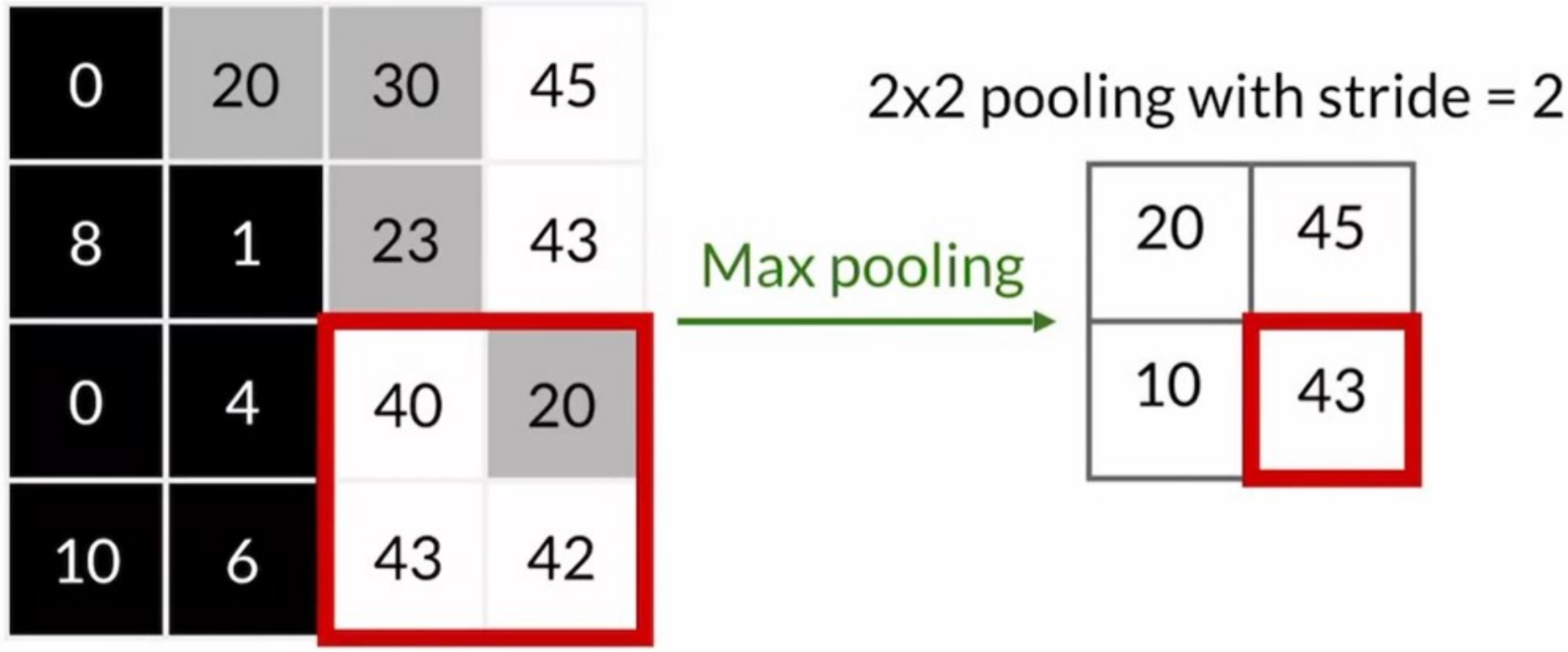
- Pooling
- Upsampling and its relation to pooling



# Pooling



# Max Pooling




4x4 input to 2x2 output

# Max Pooling

NOTE: Pooling have no weights

| | | | |
|---|---|---|---|
| 0 | 20 | 30 | 45 |
| 8 | 1 | 23 | 43 |
| 0 | 4 | 40 | 20 |
| 10 | 6 | 43 | 42 |

4x4 input to 2x2 output

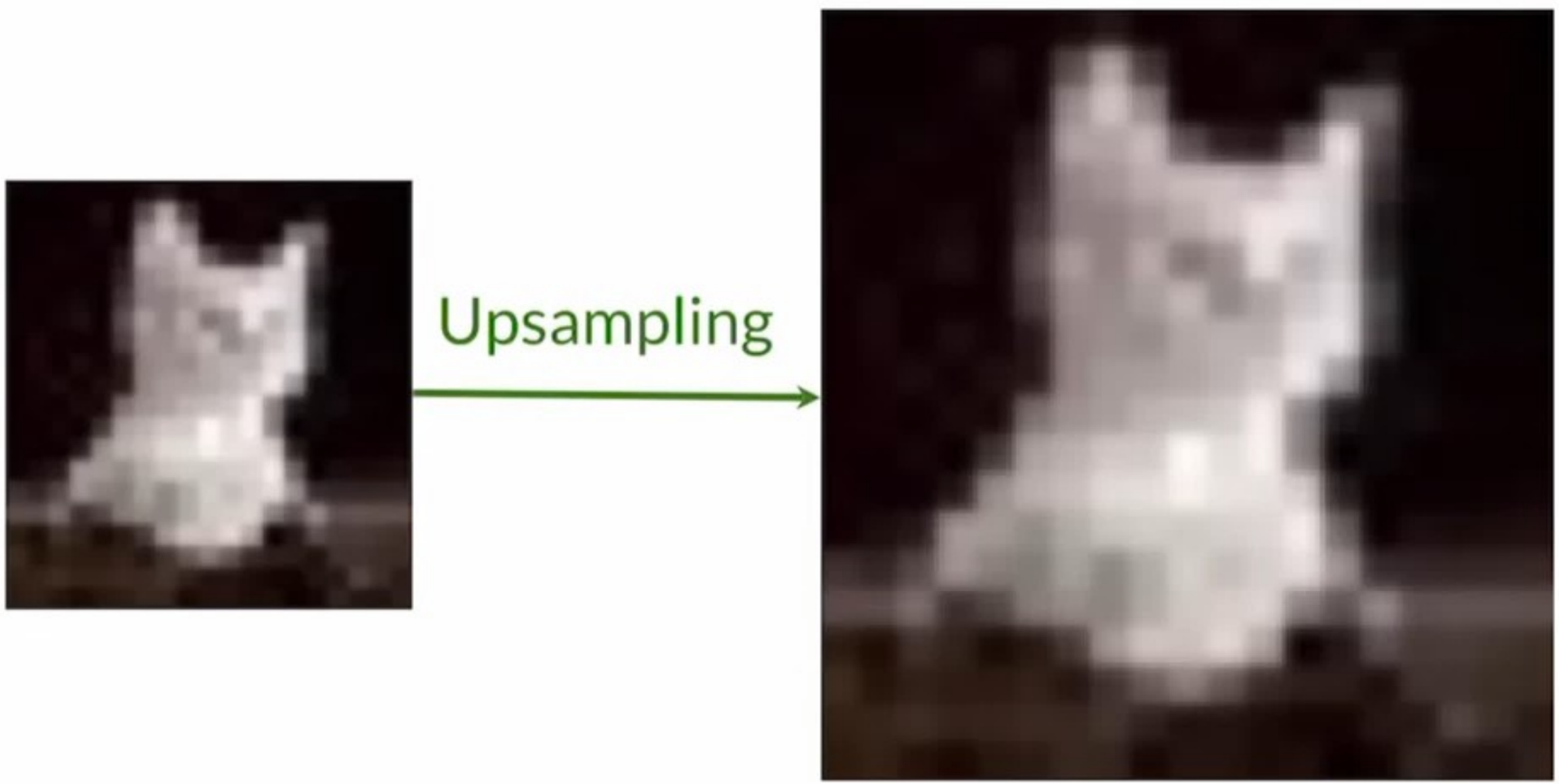
Max pooling

2x2 pooling with stride = 2

| | |
|---|---|
| 20 | 45 |
| 10 | 43 |

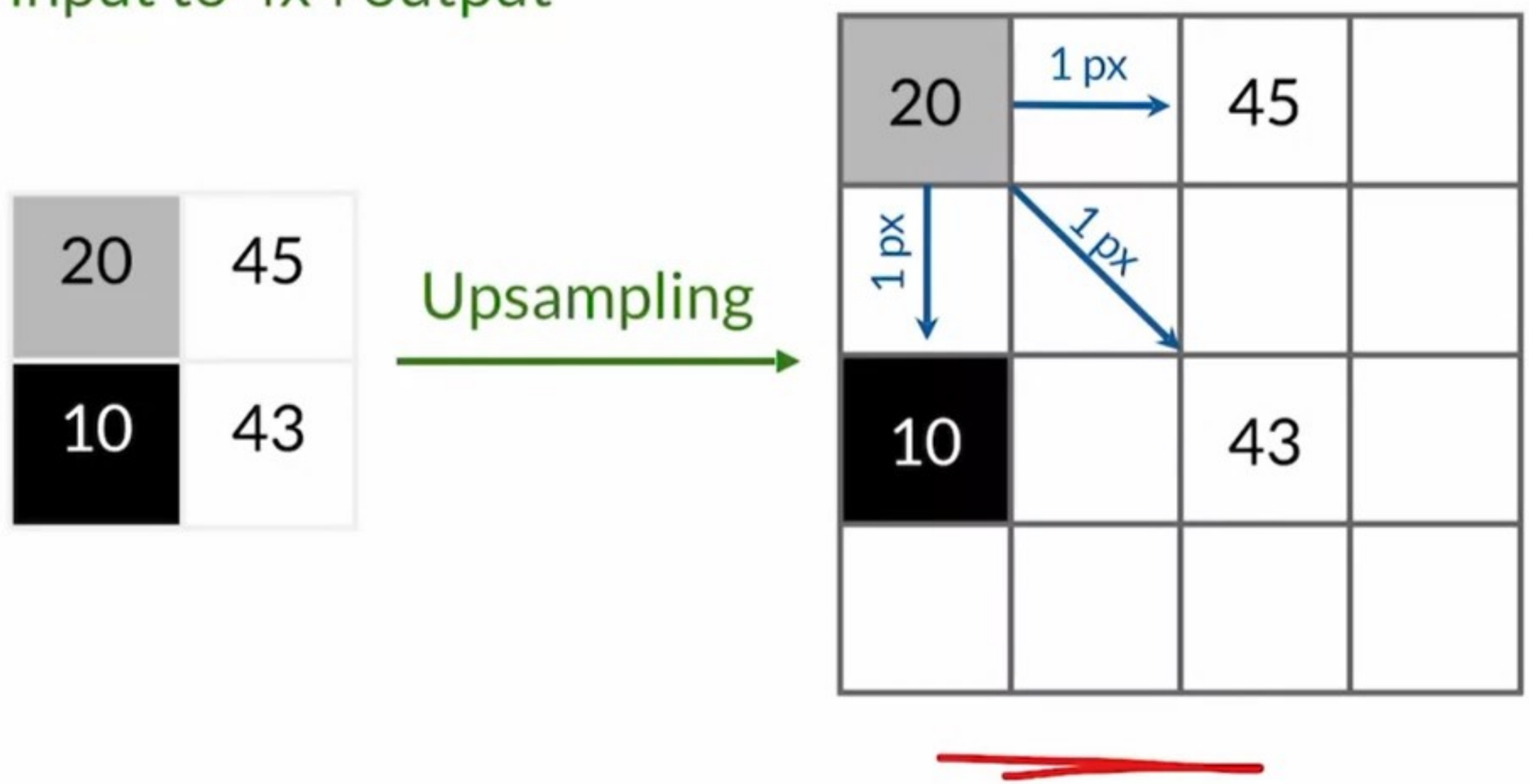
Other types include:
1. Average pooling
2. Min pooling

# Upsampling



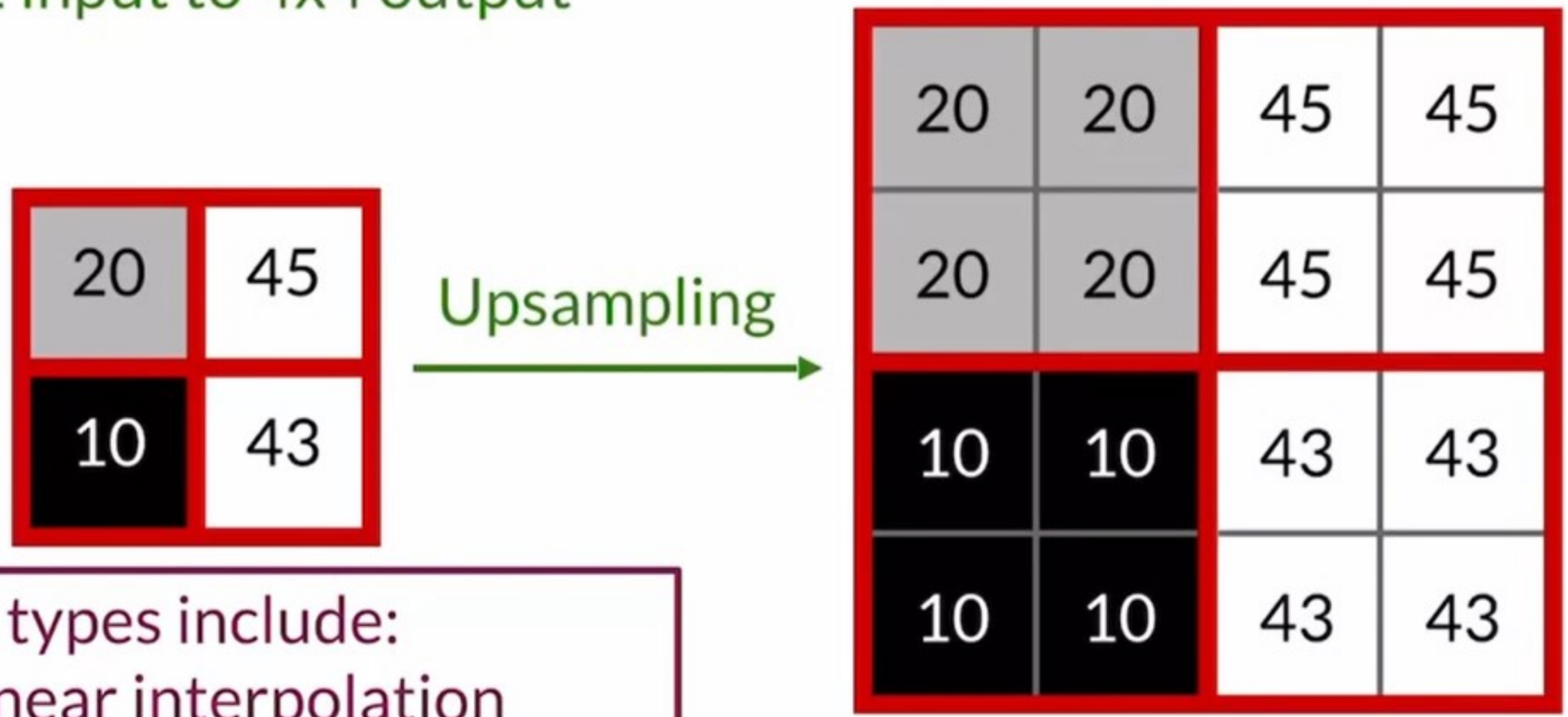
# Upsampling: Nearest Neighbors

2x2 input to 4x4 output



# Upsampling: Nearest Neighbors

2x2 input to 4x4 output

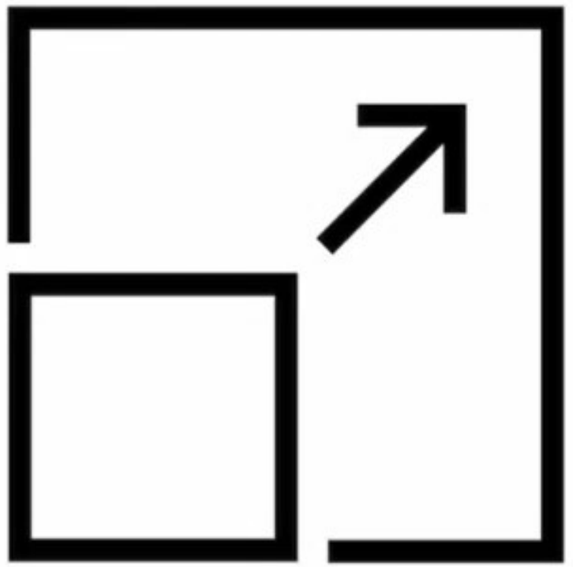



Other types include:

1. Linear interpolation
2. Bi-linear interpolation

# Summary

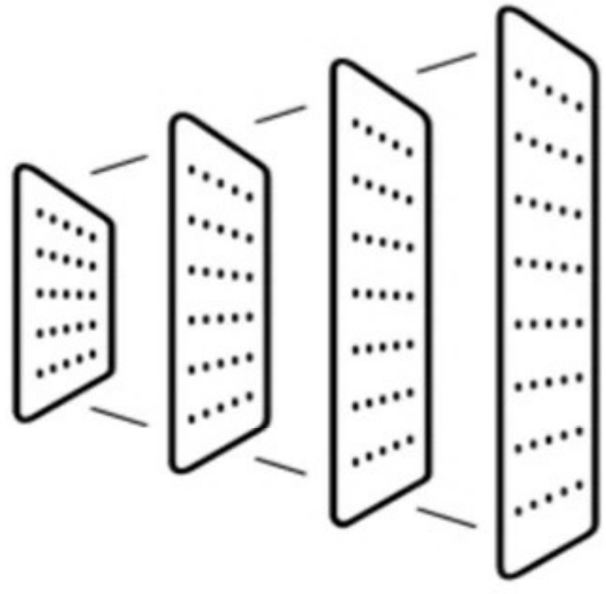
- Pooling reduces the size of the input
- Upsampling increases the size of the input
- No learnable parameters!




# Transposed Convolutions

# Outline

- Transposed convolutions as an upsampling technique
- Issues with transposed convolutions



# Transposed Convolution

Influence from every value in the input

| | |
|---|---|
| 1 | 4 |
| 0 | 2 |

Input

*

| | |
|---|---|
| 2 | 2 |
| 1 | 1 |

Filter

=

| | | |
|---|---|---|
| 1*2 | 1*2+ 4*2 | 4*2 |
| 1*1+ 0*2 | 1*1 +4*1 + 0*2+ 2*2 | 4*1+ 2*2 |
| 0*1 | 0*1+ 2*1 | 2*1 |

Stride = 1

# Transposed Convolution

Influenced by just one value

| | |
|---|---|
| 1 | 4 |
| 0 | 2 |

Input

*

| | |
|---|---|
| 2 | 2 |
| 1 | 1 |

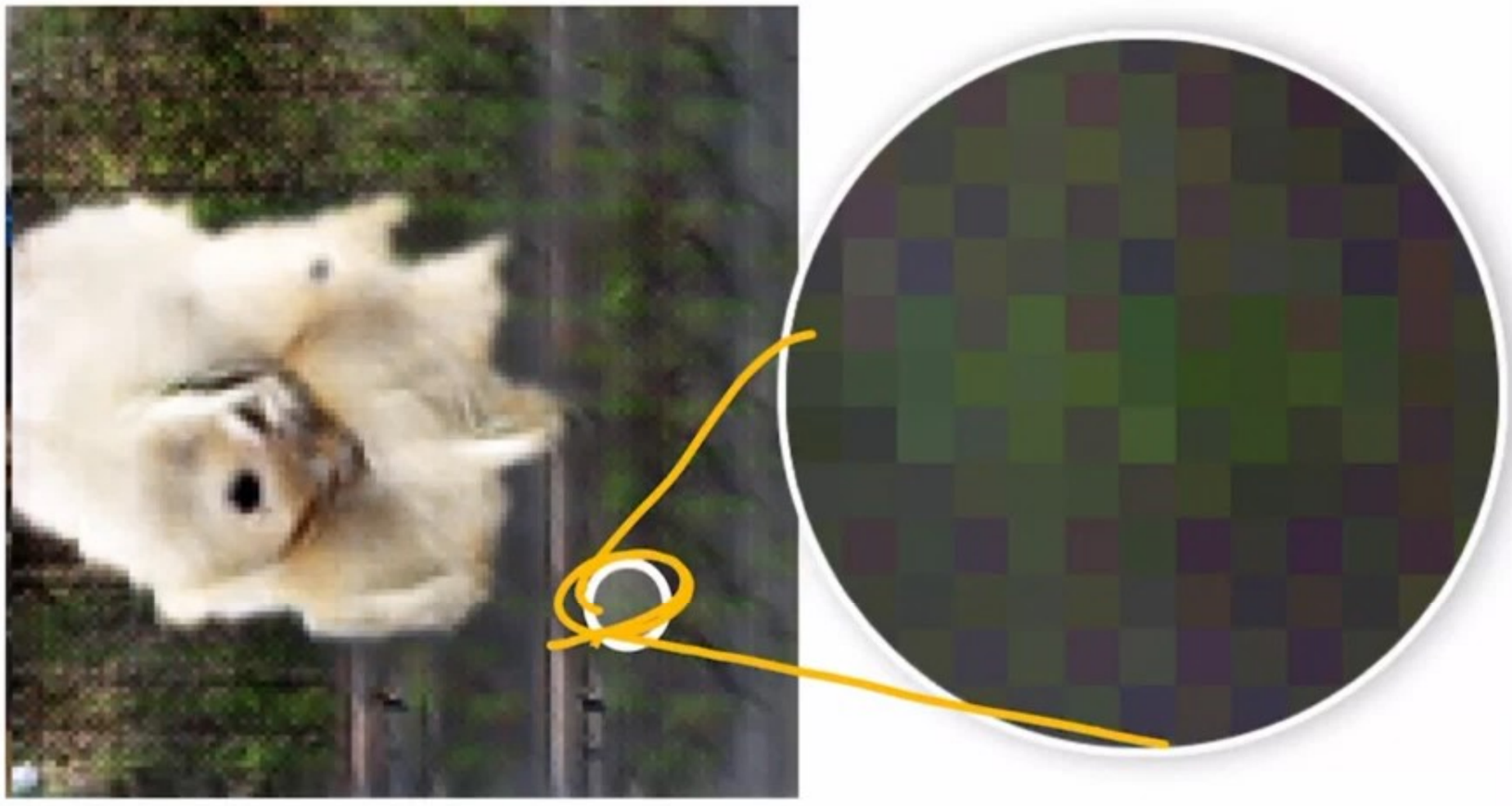
Filter

=

| | | |
|---|---|---|
| 1*2 | 1*2+ 4*2 | 4*2 |
| 1*1+ 0*2 | 1*1 +4*1 + 0*2+ 2*2 | 4*1+ 2*2 |
| 0*1 | 0*1+ 2*1 | 2*1 |

Stride = 1

# The Problems with Transposed Convolution



Checkerboard Pattern

upsampling + convolution

Available from: http://doi.org/10.23915/distill.00003

# Summary

- Transposed convolutions upsample
- They have learnable parameters
- Problem: results have a checkerboard pattern