CS3232 Fundamental of Deep learning

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Module 1

Part 5

Types of convolution, Regularization, Data augmentation

Credits: Parts of the material have been borrowed from Dr. Andrew Ng's course material, other sources on the Internet

Types of Convolutions

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Credits: Analyticsvidhya

1. Standard convolution

The basic convolution operation where a filter (kernel) is applied to the input to produce a feature map.

Use Case: General feature extraction from images.

2. Separable convolution

A process in which a single convolution can be divided into two or more convolutions to produce the same output

Spatially Separable Convolution:

Decomposes a filter into two smaller filters applied sequentially, reducing computational complexity.

Depthwise Separable Convolution:

Decomposes convolution into two steps:

depthwise convolution (applying a single filter per input channel) and pointwise convolution (applying a 1x1 filter to combine channels).

Use Case: Efficient feature extraction with fewer parameters, commonly used in lightweight models like MobileNet.

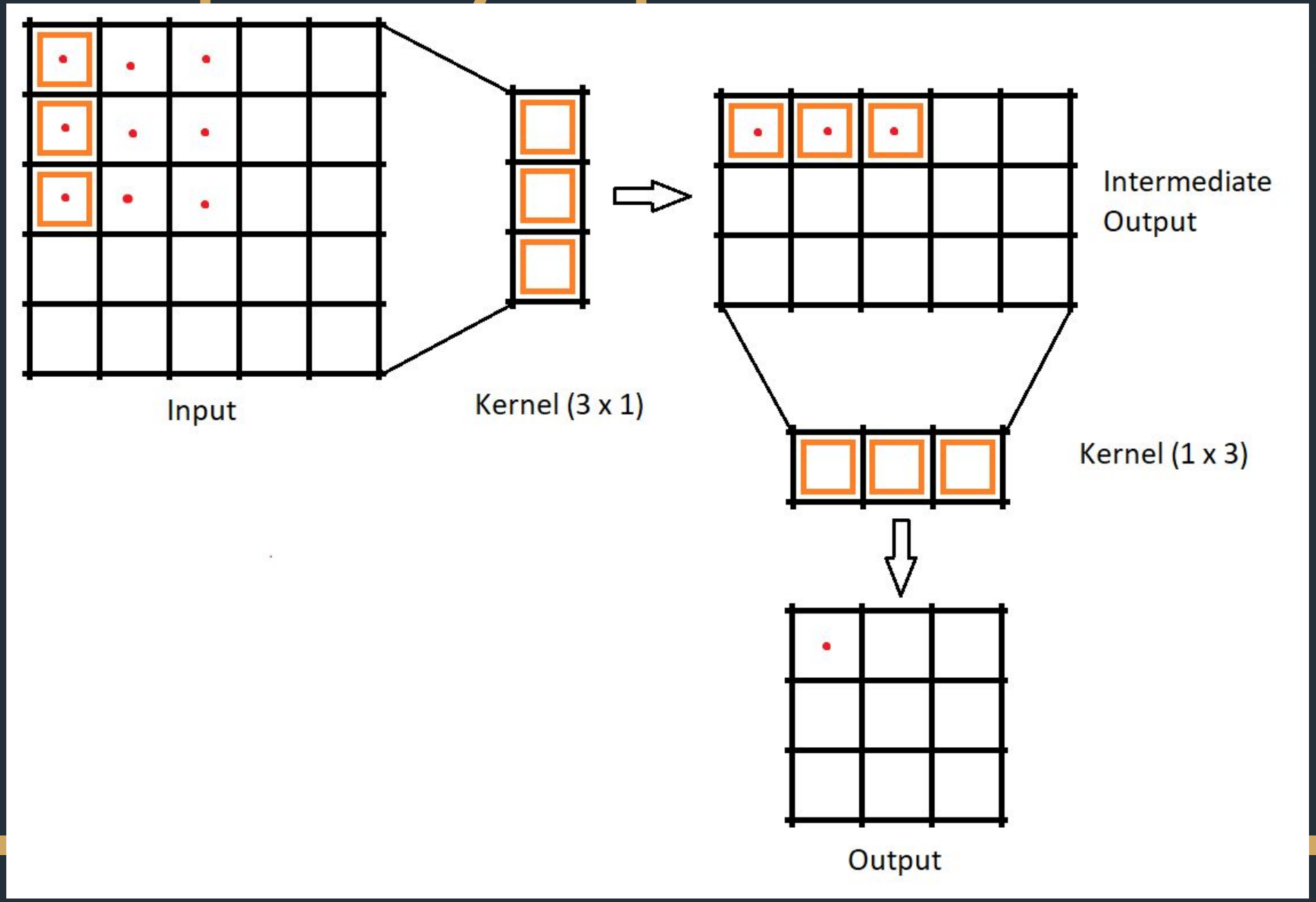
2.1 Spatially separable convolution

- In images, height and width are called spatial axes.
- The kernel that can be separated across spatial axes is called the spatially separable kernel.
- The kernel is broken into two smaller kernels and those kernels are multiplied sequentially with the input image to get the same effect of the full kernel.

  Eg. Prewitt kernel for detecting edges

$$\begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

2.1 Spatially separable convolution



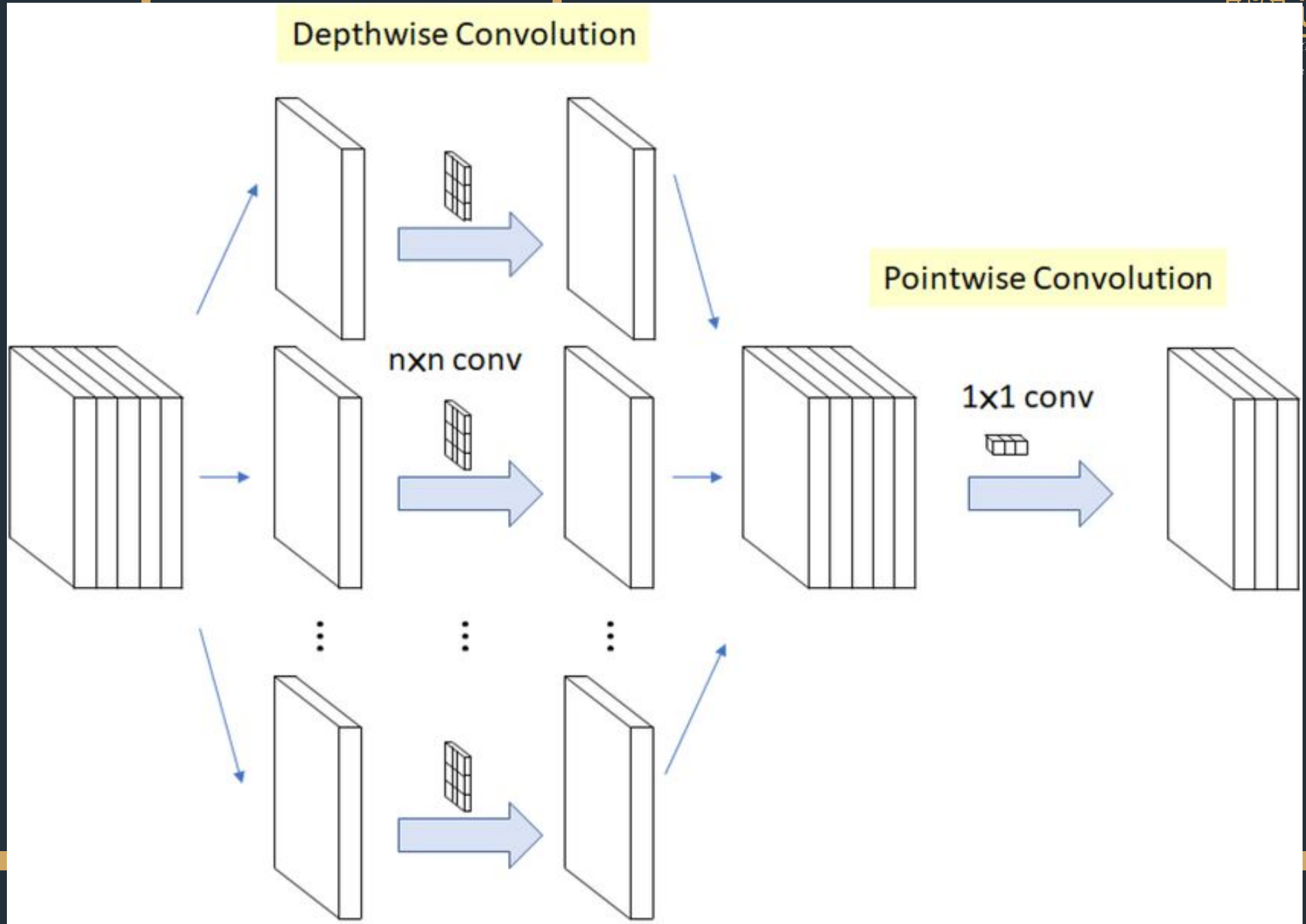
2.1 Spatially separable convolution

- In the conventional convolutions, if the kernel is 3 x 3 then the number of parameters would be ?.
- In spatially separable convolution we divide the kernel into two kernels of shapes 3 x 1 and 1 x 3.
  - number of parameters would be ?.

2.1 Spatially separable convolution

- In the conventional convolutions, if the kernel is 3 x 3 then the number of parameters would be 9.
- In spatially separable convolution we divide the kernel into two kernels of shapes 3 x 1 and 1 x 3.
  - The input is first convolved with 3 x 1 kernel and then with 1 x 3, then the number of parameters would be 3 + 3 = 6. So less matrix multiplication is required.
- Note that not every kernel can be separated.
  - Because of this drawback, this method is used lesser compared to Depthwise separable convolutions.

2.2 Depthwise separable convolutio



2.2 Depthwise separable convolution

- When we call tf.keras.layers.SeparableConv2D, we would be calling a Depthwise separable convolution layer.
- Here, we can use even the kernels that are not spatially separable.
- This is in turn divided into two convolutions namely
  - Depthwise convolution
  - Pointwise convolution

2.2.1 Depthwise convolution

- Assume an image input of shape 7x7x3.
- Make sure that after the depthwise convolution, the intermediate image has the same depth.
  - This is done by convoluting with 3 kernels with shape 3x3x?
  - Each kernel iterates on only one channel of the image, producing an intermediate output of shape ? which are stacked together to create an output of shape ?

2.2.1 Depthwise convolution

- Assume an image input of shape 7x7x3.
- Make sure that after the depthwise convolution, the intermediate image has the same depth.
  - This is done by convoluting with 3 kernels with shape 3x3x1.
  - Each kernel iterates on only one channel of the image, producing an intermediate output of shape 5x5x1 which are stacked together to create an output of shape 5x5x3.

2.2.2 Pointwise convolution

- Increase the depth of the output
  - by convoluting with a kernel with shape 1 x 1 x depth
    - Known as *pointwise convolution*.
  - Assume the depth is 32.
    - After pointwise convolution, the output would have the shape 5x5x32. This is equivalent to convolution with 32 filters of shape 5×5.
- In this example, the regular convolution would have to do 32 3x3x3 kernels that move 5×5 times that is 21,600 multiplications
- But pointwise convolution does 3x3x3x5x5 + 32x1x1x3x5x5 = 3075 multiplication.
  - Eliminates a large number of multiplications

2. Separable convolution – drawbacks

Though useful, separable convolution is not used very often because

- Not every kernel can be divided into two kernels
- As the number of parameters in separated convolutions is lesser, we would lose some accuracy.
- Models with Separable convolutions tend to underfit.
- When the model itself is very small, reducing the parameters may lead to complete failure of the model.

3. Transposed convolution (deconvolution)

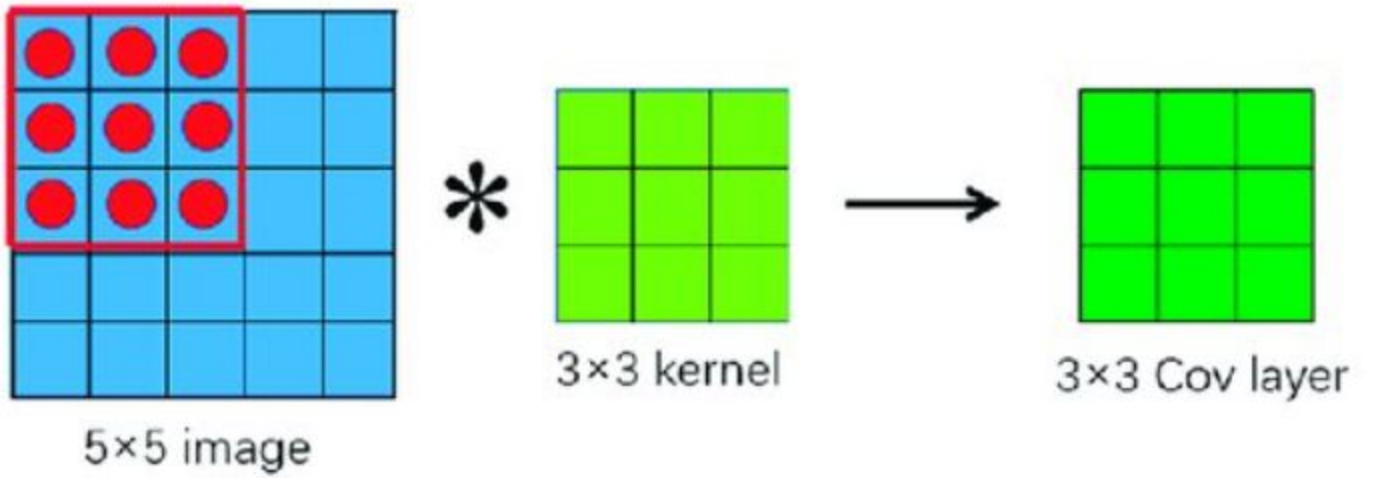
Transposed Convolution (Deconvolution): Reverses the convolution operation, used to increase the spatial dimensions of the input.

Use Case: Up-sampling feature maps, commonly used in generative models and image segmentation.

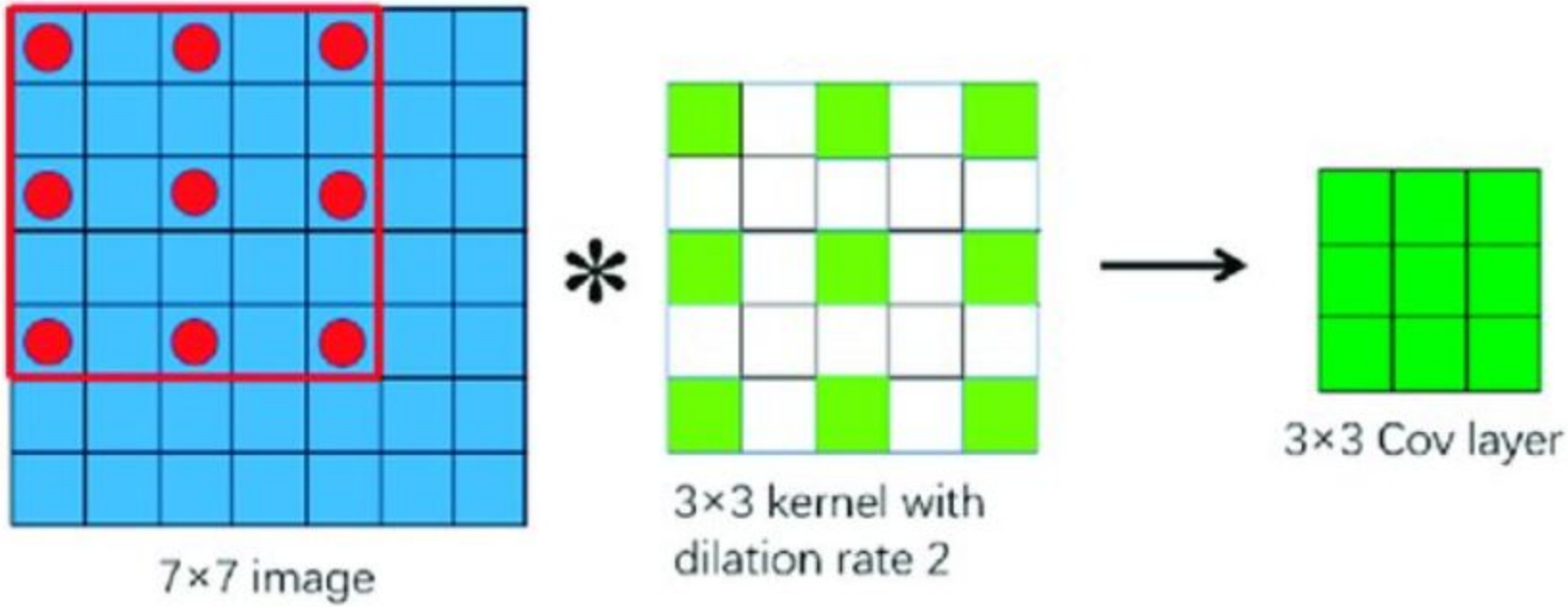
4. Dilated (Atrous) convolution

Enables networks to capture broader context, without significantly increasing computational cost or parameters (by introducing zeros in the kernel).

https://www.exxactcorp.com/blog/Deep-Learning/atrous-convolutions-u-net-architectures-for-deep-learning-a-brief-history


5×5 image
3×3 kernel
3×3 Cov layer

(a)


7×7 image
3×3 kernel with
dilation rate 2
3×3 Cov layer

4. Dilated (Atrous) convolution

- Unlike the standard convolution, which applies filters to adjacent pixels, atrous convolution spaces out the filter parameters by introducing gaps between them, controlled by the dilation rate.
  - A dilation rate of 1 corresponds to standard convolution, 2 doubles the spacing between elements, 3 triples the spacing...
- Increasing the dilation rate effectively increases the field of view of the filter without increasing its size.
  - This allows the network to capture information from a larger region of the input.
- By using multiple layers with different dilation rates, the network can simultaneously extract features at different scales.
  - This is particularly useful for tasks like semantic segmentation where objects can vary in size

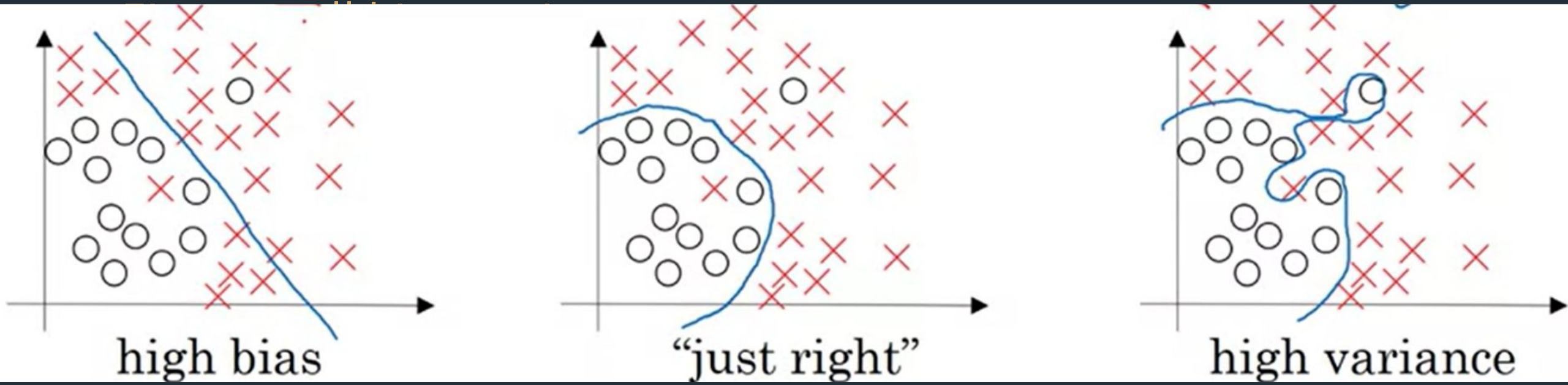
5. Grouped convolution

- Divides the input channels into groups and performs convolution separately on each group
  - effectively reducing the number of parameters and computations.
- Use Case: Reducing computational cost and memory usage, used in architectures like ResNeXt.

https://animatedai.github.io/

Regularization

First, recall bias, variance



Bias and Variance

Cat classification

Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias low variance

Human: ≈0%

Optimal (Bayes) error: ≈ 0%

When high bias or variance

1. Address high bias, if there (training data problems)
   a. Try bigger network
   b. Train for longer
   c. Try different NN architectures (helps sometimes)
2. Address high variance, if there

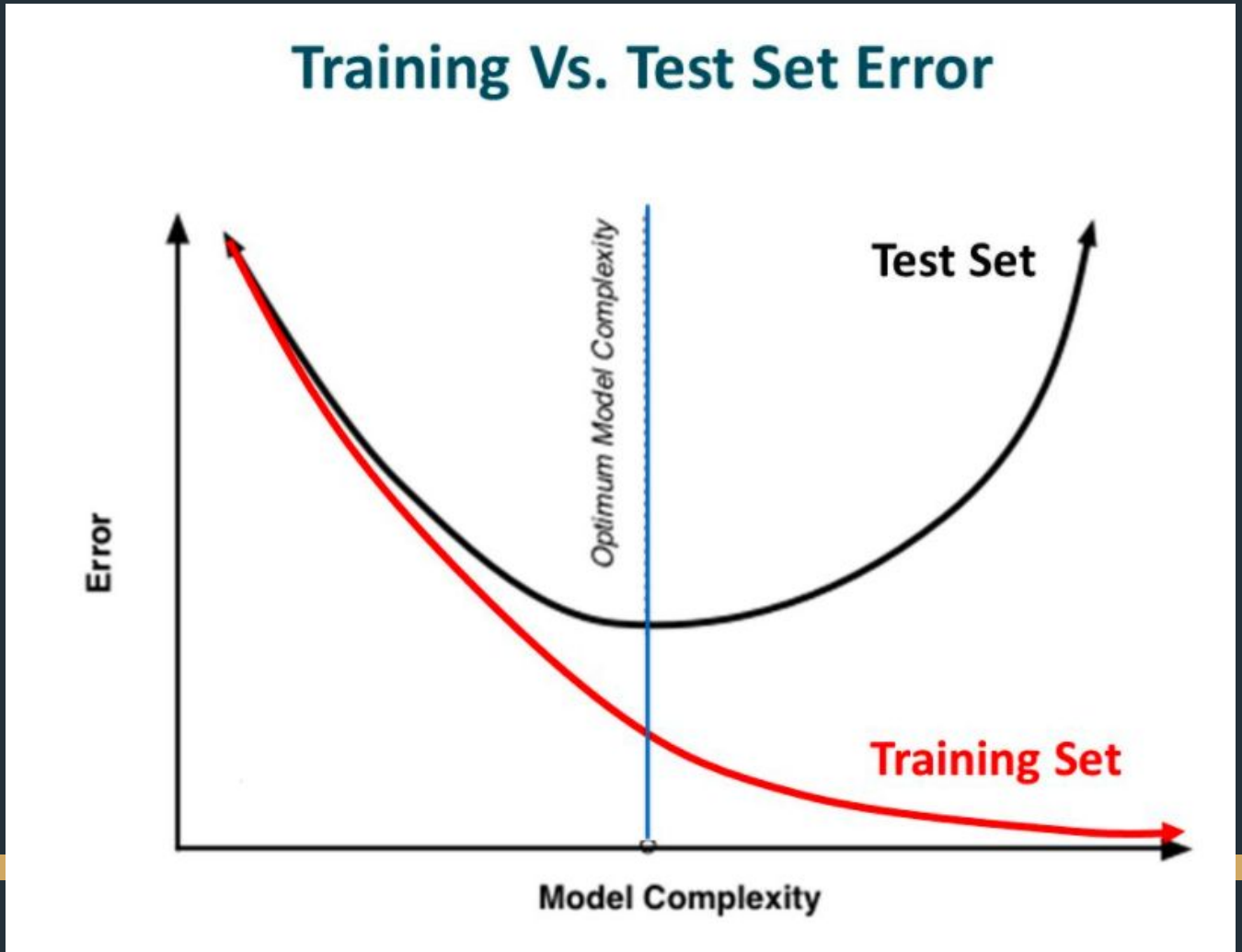
   (generalize from good training set performance to "dev" set performance)

   a. Get more data
   b. Add regularization (esp. when can't get more data)
      i. might increase bias a **little**)
   c. Try different NN architectures (helps sometimes)

1. No bias-variance **tradeoff**
   b. In this era of modern deep learning, big data
   c. One of the reasons why DL is useful for supervised learning

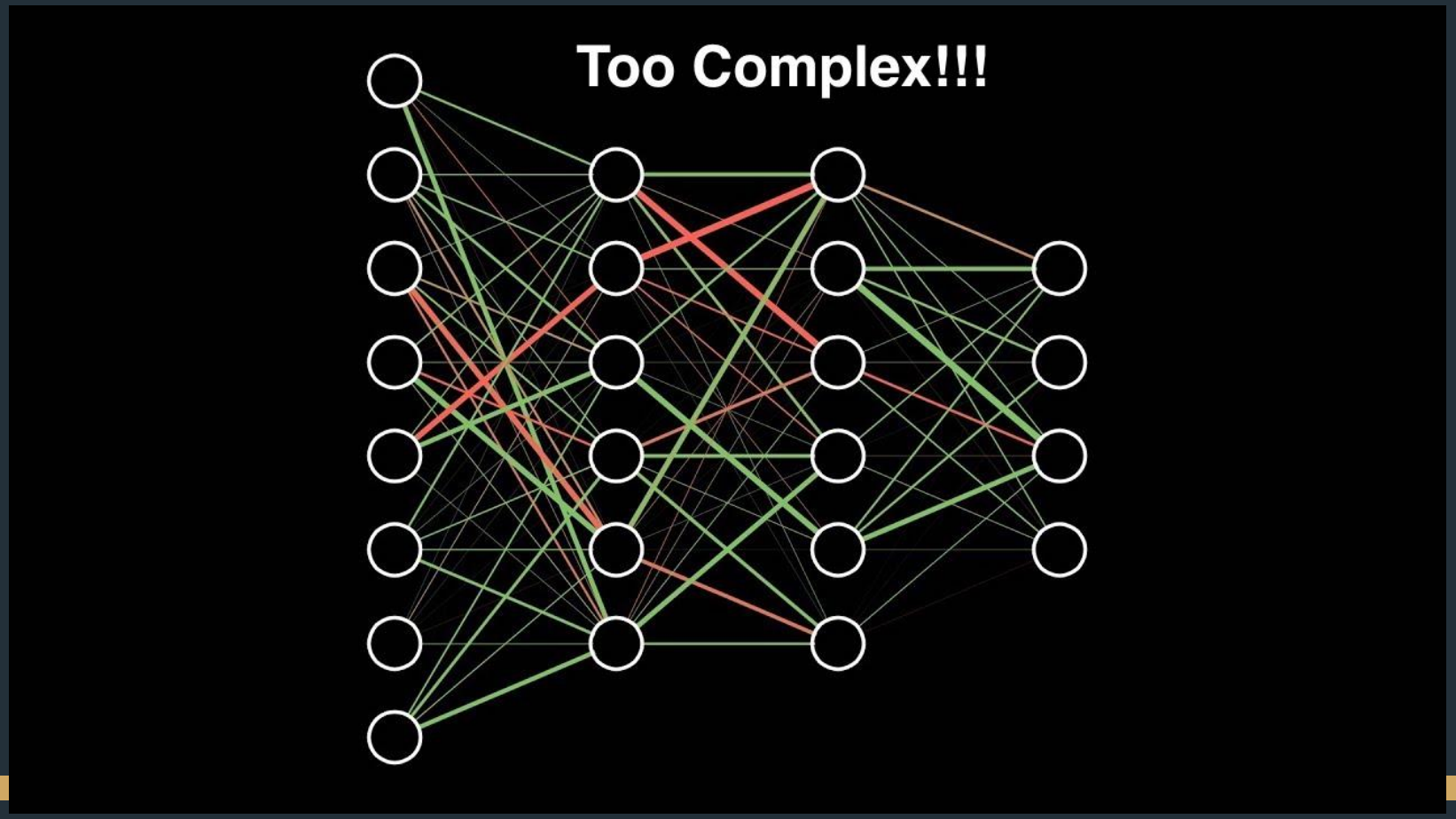
Regularization

The complexity of the model increases such that the training error reduces but the testing error doesn't.



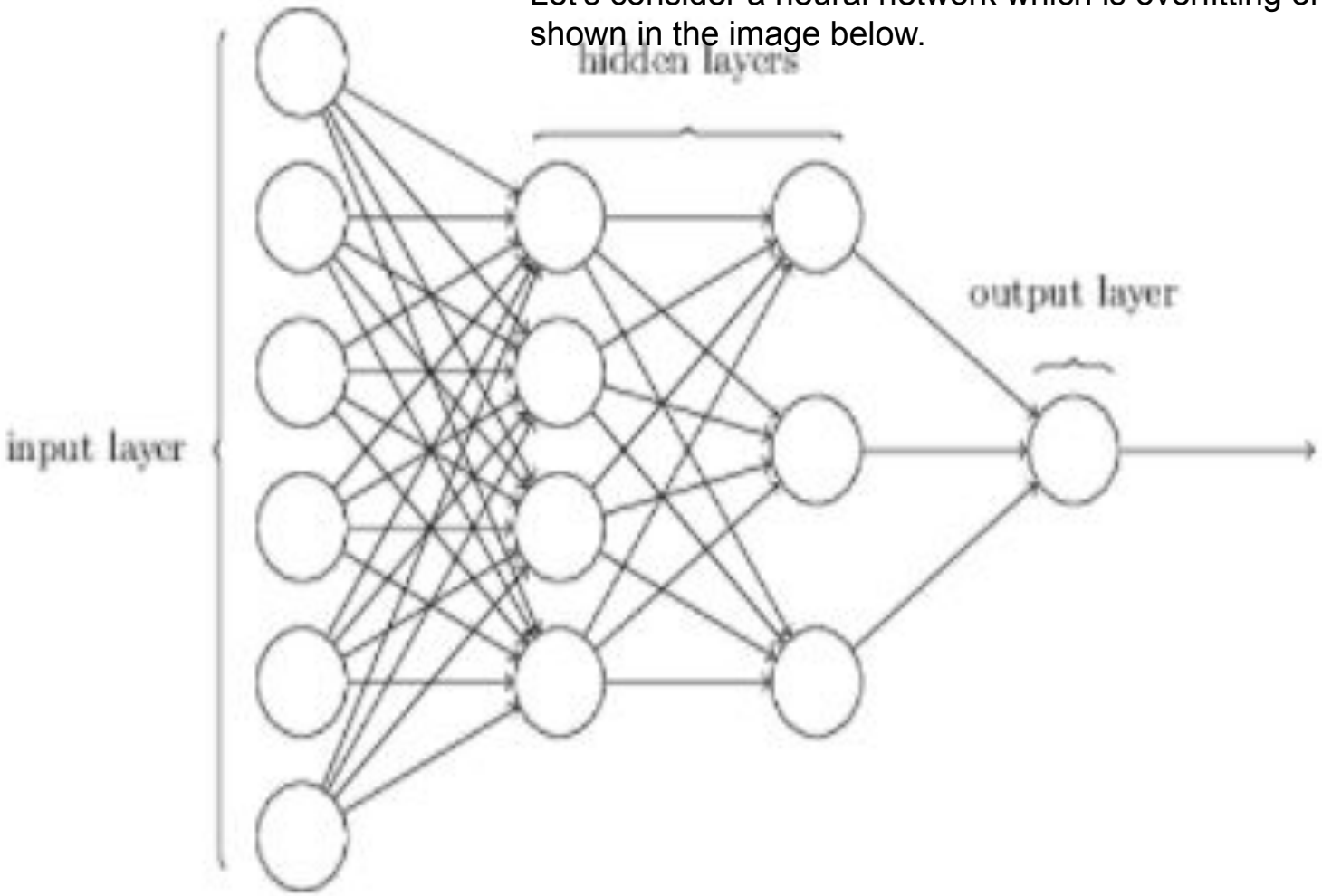
Regularization

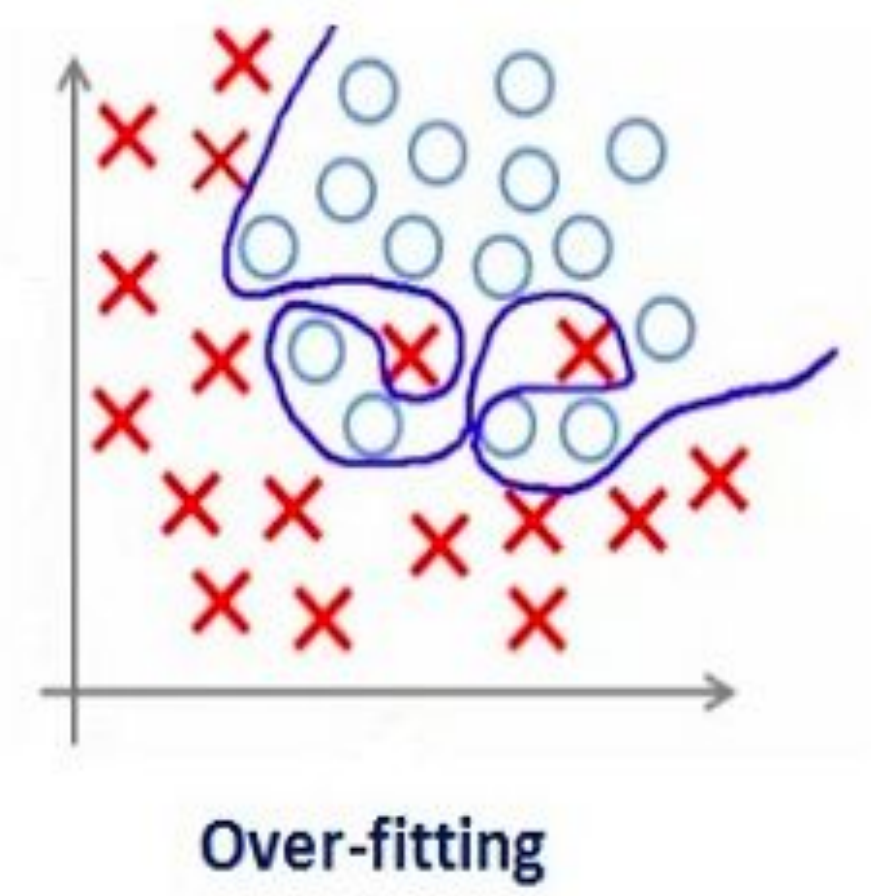
Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This in turn improves the model's performance on the unseen data as well.



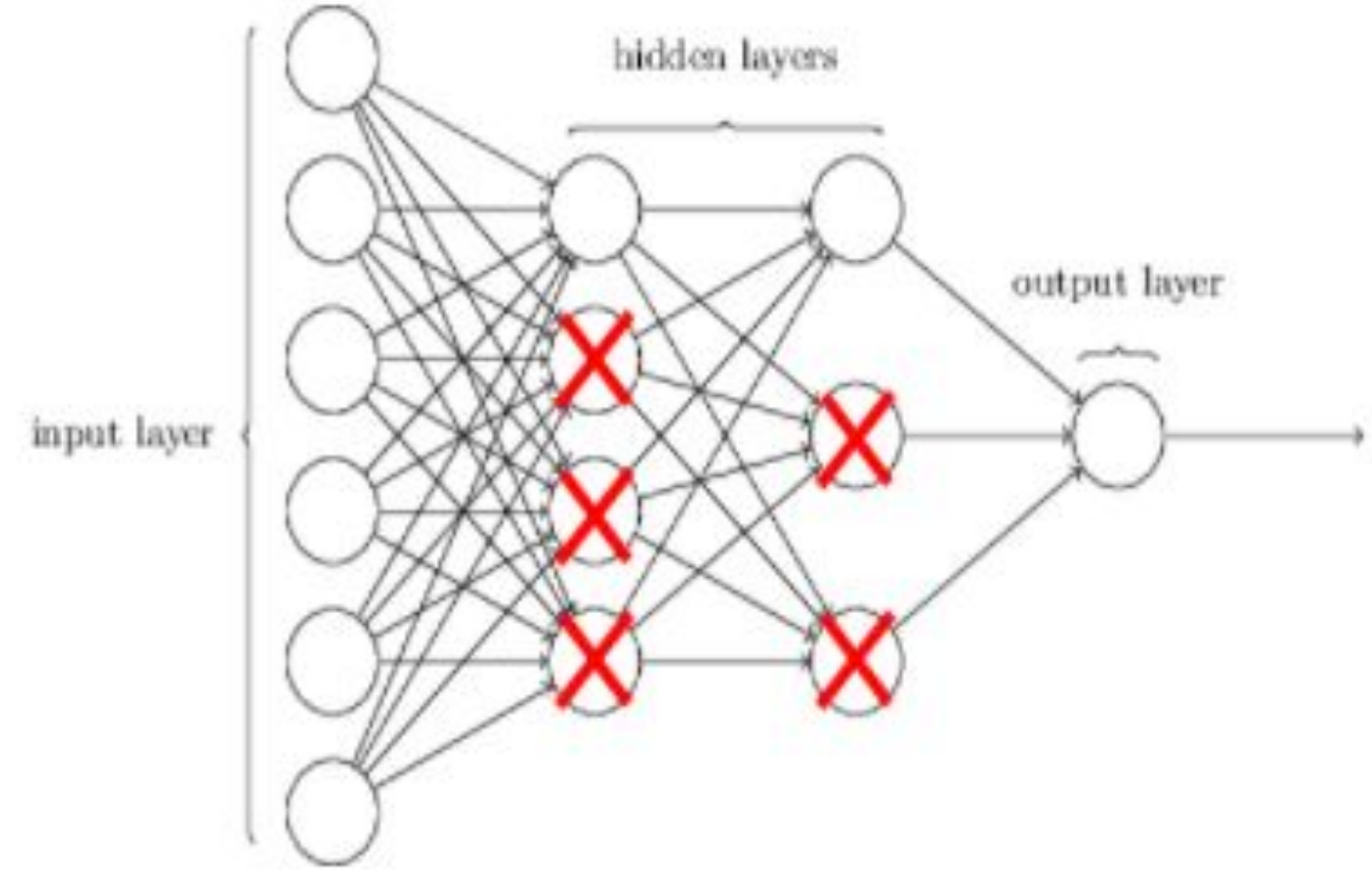
How does Regularization help reduce Overfitting?

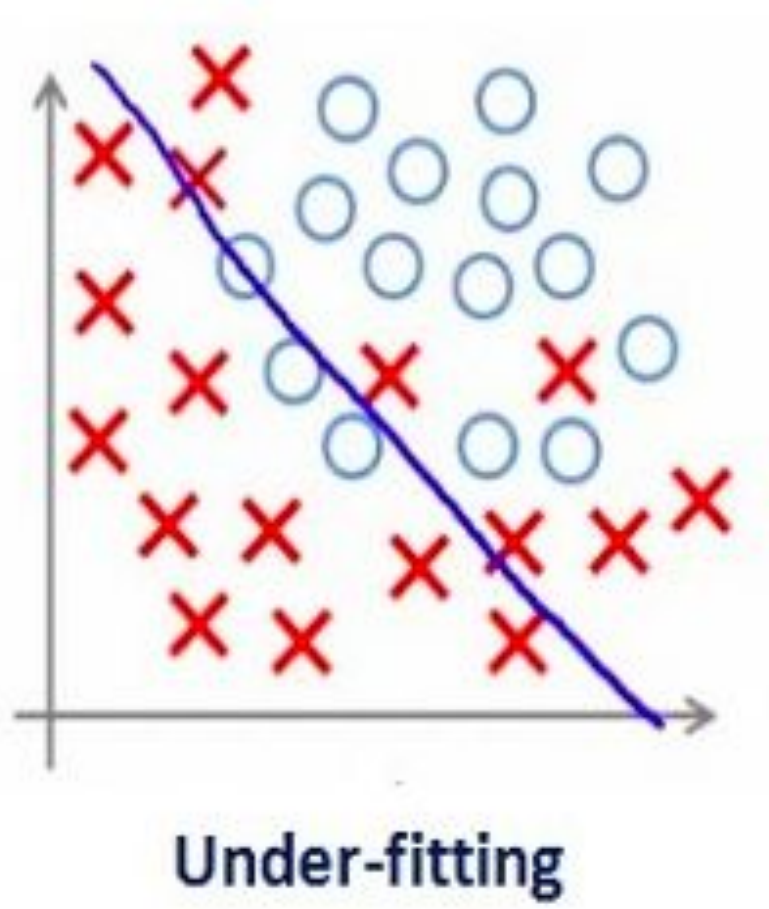
Let’s consider a neural network which is overfitting on the training data as shown in the image below.



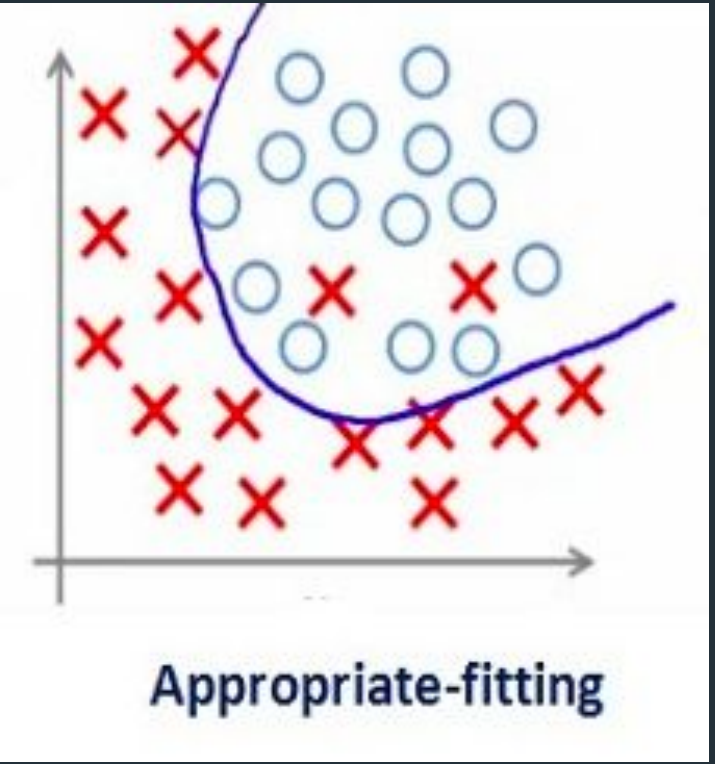


- Assume that our regularization coefficient is so high that some of the weight matrices are nearly equal to zero.





Large value of the regularization coefficient is not that useful. We need to optimize the value of regularization coefficient in order to obtain a well-fitted model as shown in the image below.



1. L2 & L1 regularization

- L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.
- Cost function = Loss (say, binary cross entropy) + Regularization term
- Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.

1. L2 & L1 regularization

$$Cost\ function = Loss + \frac{\lambda}{2m} * \sum ||w||^2$$

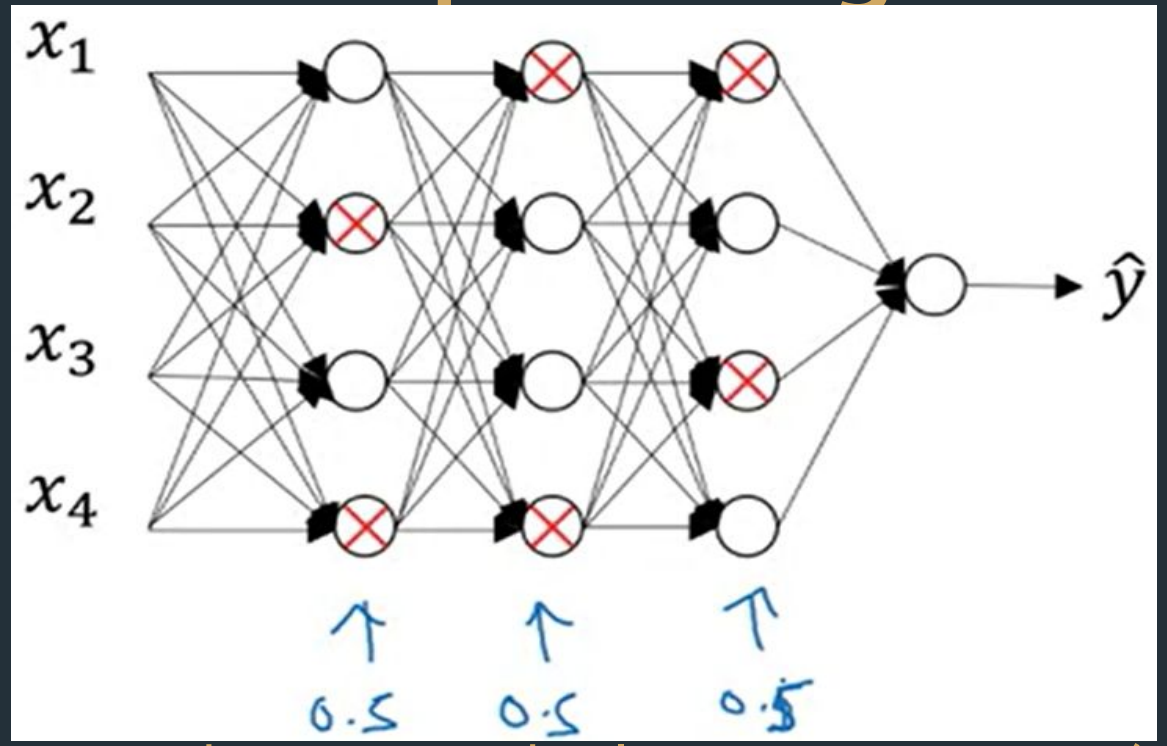
$$Cost\ function = Loss + \frac{\lambda}{2m} * \sum ||w||$$

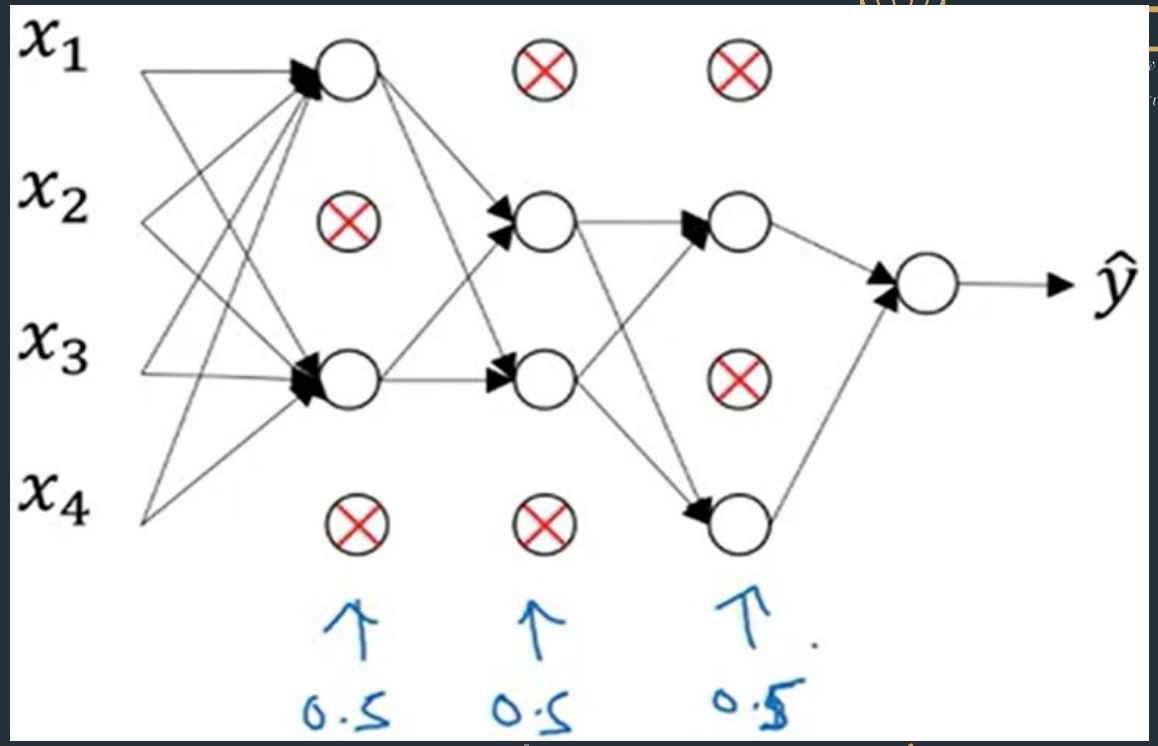
lambda is the regularization parameter.

It is the hyperparameter whose value is optimized for better results.

L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).

2. Dropout regularization

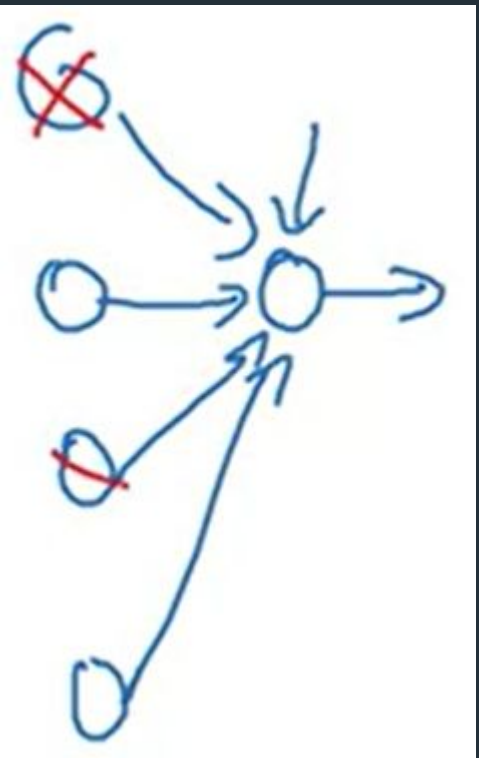




- Say, the NN on the left is overfitting (after training; not during testing)
- Go through each layer and set a probability per layer for eliminating a node (i.e.) reduce the network <- hyperparameter (so, can skip dropping from some layers)
  - Though nodes can be eliminated from the input layer, usually not
- Remove incoming, outgoing links for these nodes
- Train that example (different nodes may be dropped in different iterations
- Set probabilities again and repeat for the next example and so on
- *Inverted* dropout is used commonly

Why does drop-out work?

- Drop-out eliminates nodes at random in every iteration
  - A smaller network in every iteration
    - Regularizing effect
- Looking at it from a single unit's view
  - Inputs can eliminated randomly
  - => the unit can not rely on any one feature
  - Instead, at least give a small weight to all the inputs
  - => shrink weights, similar to L2 regularization
    - Prevents overfitting
  - An adaptive form of L2 regularization
    - But the L2 penalty on different weights are different depending o of the size of the activation multiplied into that weight



2. Drop-out regularization

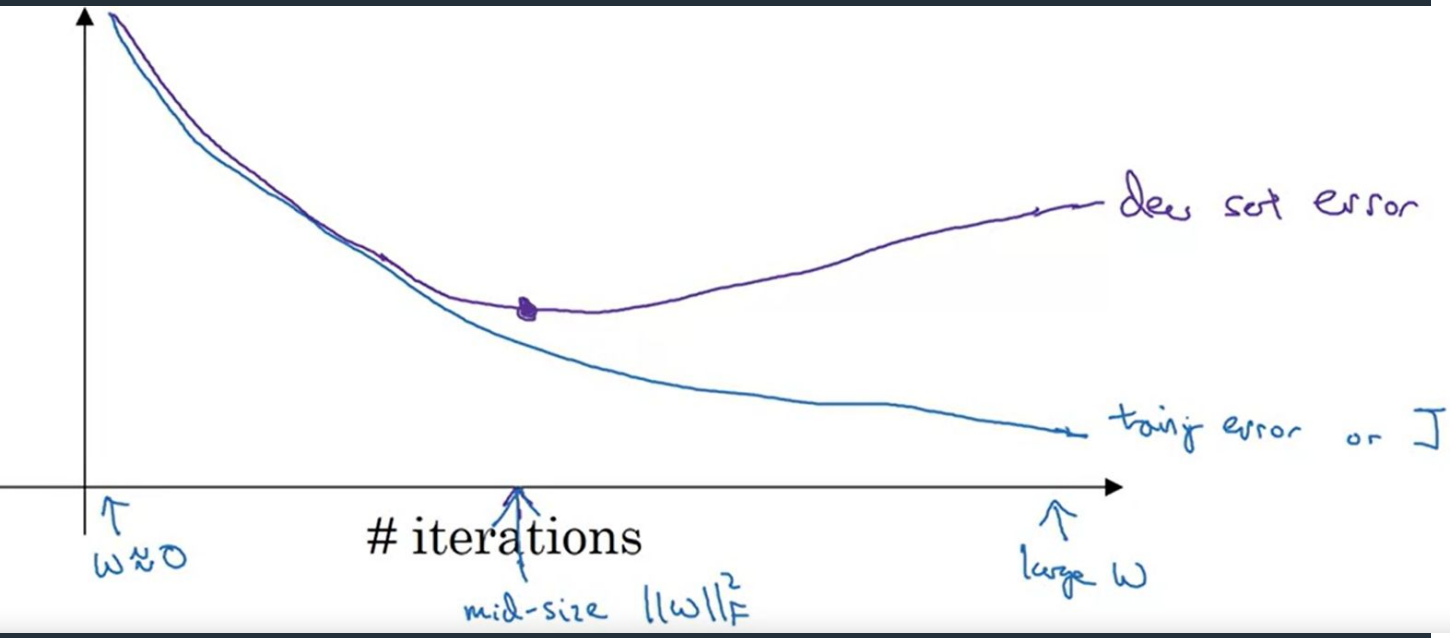
- Many drop-out implementations were in computer vision
  - Input size is large
    - Therefore, drop-out is used almost by default
  - But use drop-out only if there is overfitting
- Downsides of drop-out
  - The cost function J is no longer well defined because
    - At every iteration, nodes are eliminated randomly
    - => harder to double-check the performance of gradient descent
    - We lose the debugging tool to observe the cost function decreasing with every iteration
      - Turn off drop-out for debugging

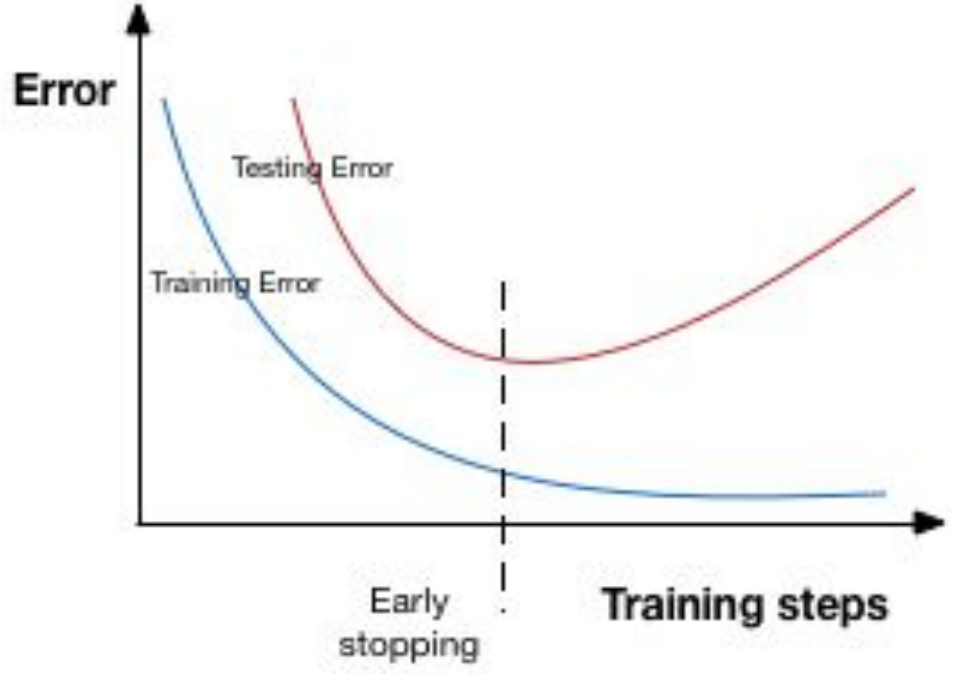
3. Data Augmentation regularization

- The simplest way to reduce overfitting is to include more training data.
  - Getting more training data can be expensive or may not be possible to get more data
- While dealing with images
  - A few ways of increasing the size of the training data – through augmentation, inexpensively
    - rotating the image randomly and zoom in too
    - flipping, scaling, shifting, random distortion etc. (random transformations)
  - In the below image, some transformation has been done on the handwritten digits dataset.

4. Early stopping

- Early stopping is a kind of cross-validation strategy where we keep one part of the training set as the validation set.
- When the performance on the validation/ dev set is getting worse, immediately stop the training on the model <- early stopping.
  - Stop training at the dotted line since after that our model will start overfitting on the training data.





4. Early stopping – why it works

RV UNIVERSITY
Go, change the world

- When not yet many iterations
  - Weights will be close to 0 as we start with small random values for them
  - Weights get larger with more iterations
- Early stopping
  - Stops half-way
  - Mid-size values for weights
    - Similar to L2 regularization, prevents overfitting
- Downsides of early stopping
  - Couples tasks that are separated in ML (orthogonalized)
    - Instead of different tools for the two tasks, one tool and so neither may be done well
      - In ML, cost function is optimized
        - gradient descent etc.
      - After optimizing J, there should be no overfitting
        - Regularization,, getting more data etc.
- Alternative to early stopping
  - L2 regularization – need to try different values of λ though (but preferred)
  - Lets us train for as long as possible