Neural Network Architectures: CNN and Training Nets

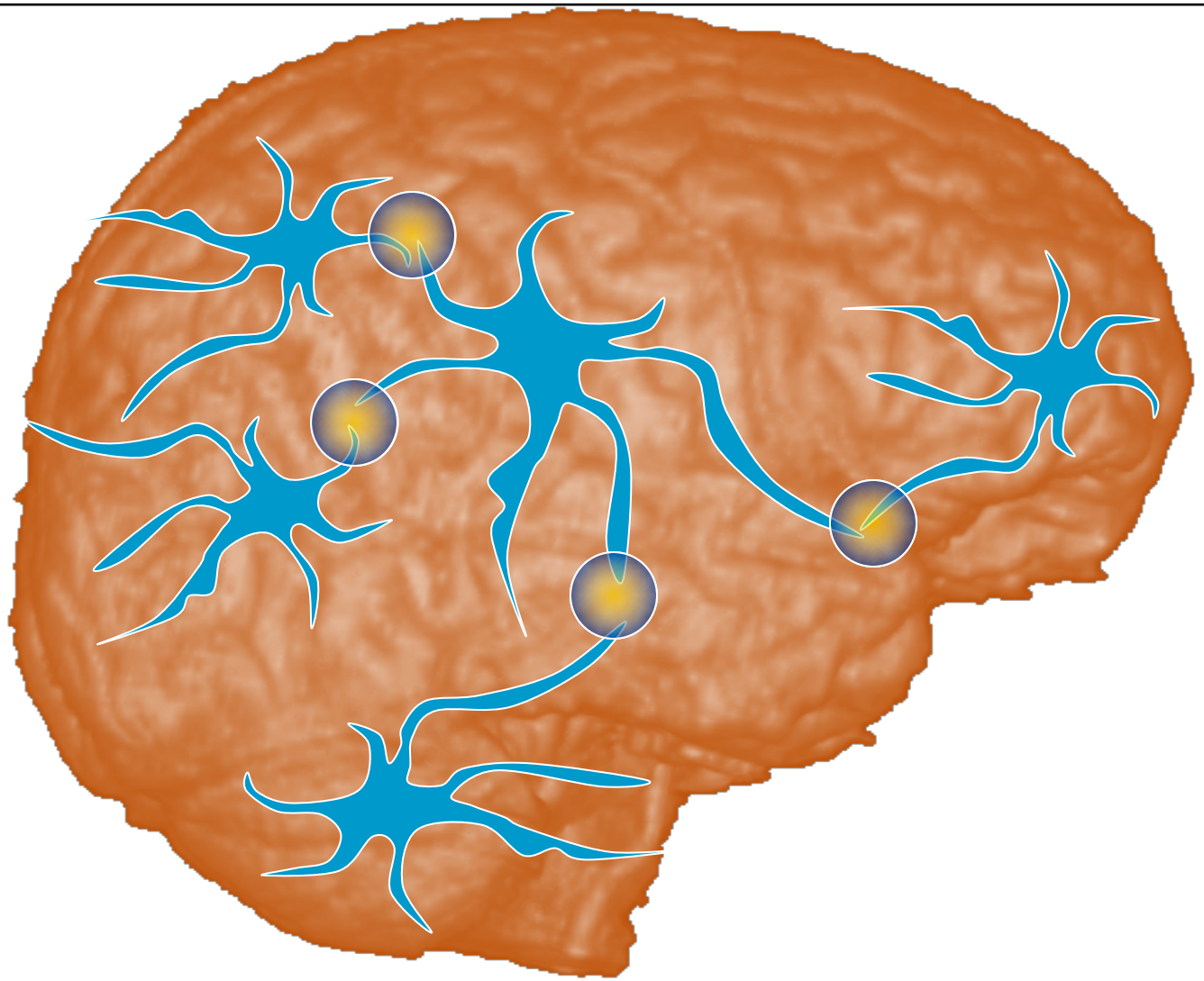
Instructor: Dr. Muhammad Fahim

INNOPOLIS UNIVERSITY

Contents

- Information Processing in Neural Networks
- Artificial Neural Network
- Convolutional Neural Network (ConvNets)
- Training Networks
  - Preprocessing
  - Activation Function
  - Weight Initialization
  - Hyperparameter Optimization
  - Evaluation model ensemble
- Summary

Information Processing in a Neural Network



Artificial Neural Network

- An artificial neural network is an information processing system that has certain performance characteristics in common with biological neural networks.

- An ANN can be characterized by:
  1. **Architecture:** The pattern of connections between different neurons.
  2. **Training or Learning Algorithms:** The method of determining weights on the connections.
  3. **Activation Function:** The nature of function used by a neuron to become activated.

Convolutional Neural Networks

Yann LeCun – 1989

Yann LeCun
Director of AI Research
Facebook

Key Idea!!

Convolutional networks are simply feed-forward neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Convolutional Neural Networks (CNN or ConvNets)

- It is a feed-forward network that can extract topological properties
- The network employs a mathematical operation called convolution
- Convolution is a specialized kind of linear operation
- **Examples**
  - **Time-series data:** which can be thought of as a 1-D grid taking samples at regular time intervals
  - **Image data:** which can be thought of as a 2-D grid of pixels.

The Convolution Operation

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

The region in the image – where this kernel will be applied is known as **receptive field**

1	-1	-1
-1	1	-1
-1	-1	1

Kernel1

-1	1	-1
-1	1	-1
-1	1	-1

Kernel 2

⋮ ⋮

Each kernel detects a small pattern (3 x 3).

These are the network parameters to be learned.

The Convolution Operation

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Dot product →

3 -1

1	-1	-1
-1	1	-1
-1	-1	1

Kernel1

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

The Convolution Operation

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

1	-1	-1
-1	1	-1
-1	-1	1

Kernel1

NOTE: Stride is one of the design choice of architecture

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

The Convolution Operation

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

1	-1	-1
-1	1	-1
-1	-1	1

Kernel 1

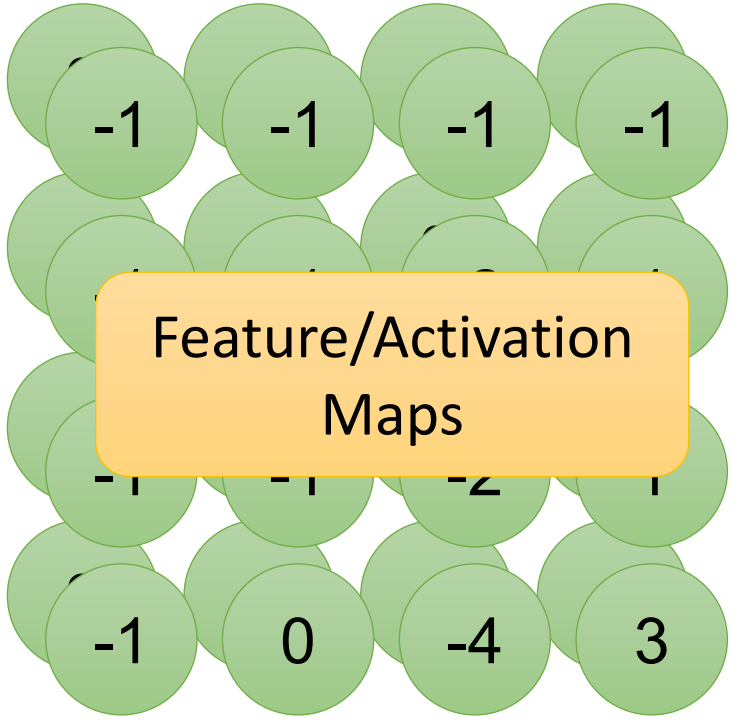
The Convolution Operation

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Repeat this for each kernel

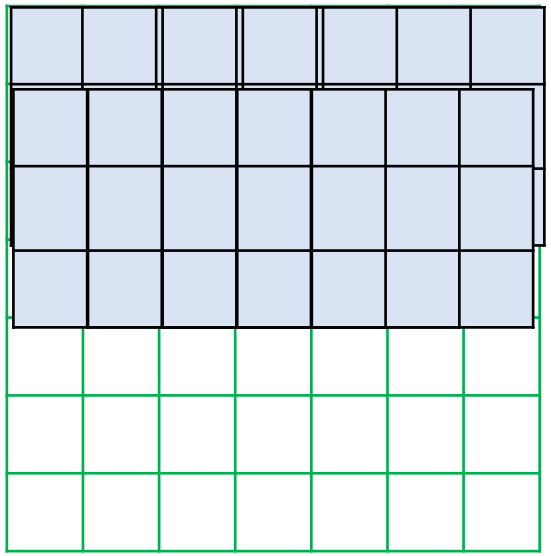


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

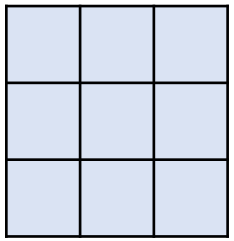
-1	1	-1
-1	1	-1
-1	1	-1

Kernel 2

The Convolution Operation



7 x 7
Input



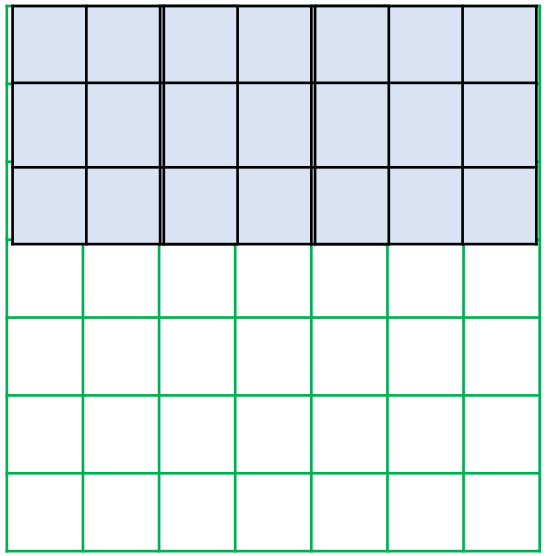
3 x 3
Kernel

Stride 1

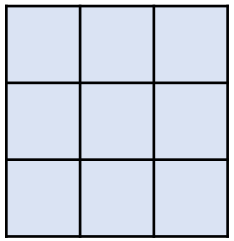
Output = ??

Output = 5 x 5

The Convolution Operation



7 x 7
Input



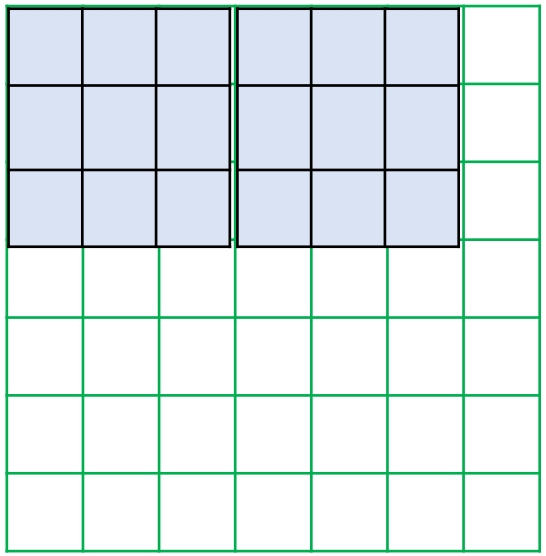
3 x 3
Kernel

Stride **2**

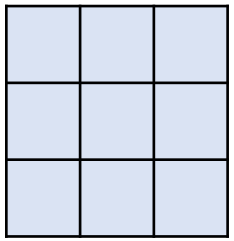
Output = ??

Output = 3 x 3

The Convolution Operation



7 x 7
Input



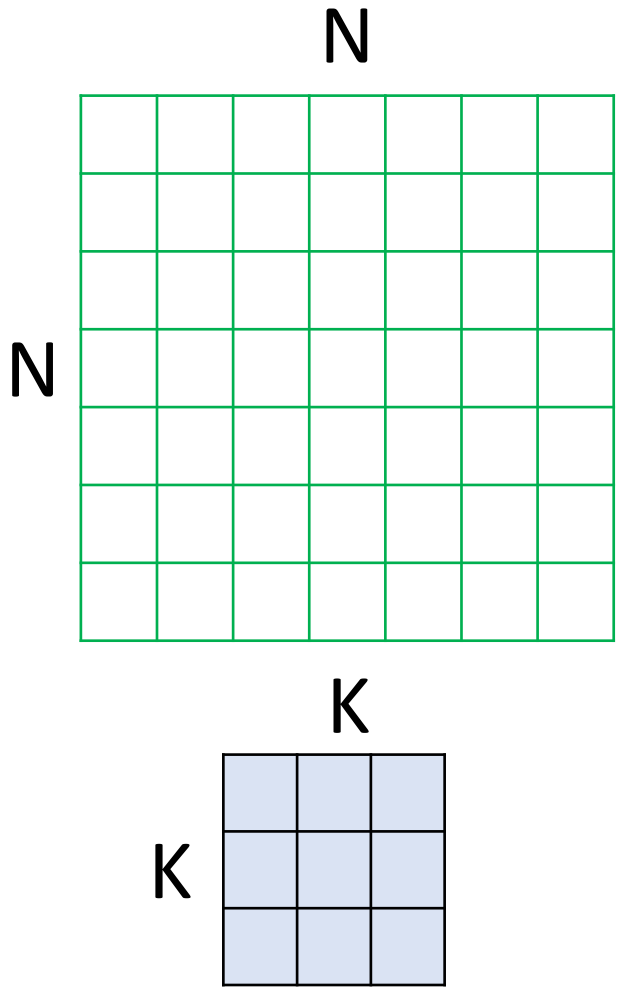
3 x 3
Kernel

Stride **3**

Output = ??

Output = doesn’t fit!!

The Convolution Operation



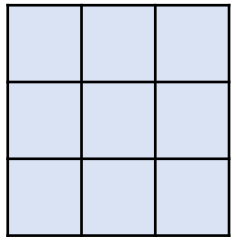
- Output size = [(N - K) / stride] + 1

- Fore Example: N = 7, K = 3:
  - Stride 1 => [(7 - 3)/1] + 1 = 5
  - Stride 2 => [(7 - 3)/2] + 1 = 3
  - Stride 3 => [(7 - 3)/3] + 1 = 2.33 ☹

What to do now?

The Convolution Operation – Zero Padding

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0



Output size = [(N - K) / stride] + 1

Here!!
Input = 7 x 7
Stride = 1
Kernel = 3 x 3
Output size = ??

Output Size = 7 x 7

Because input is 9 x 9 after zero padding

The Convolution Operation – Zero Padding

- In general, common to see convolutional layers with
  - Stride 1
  - Filters of size FxF and zero-padding with (F-1)/2

  - For Example:
    - F = 3 => zero pad with 1
    - F = 5 => zero pad with 2
    - F = 7 => zero pad with 3

- Adding zero-padding is also called wide convolution, and not using zero-padding would be a narrow convolution.

Convolution Layer – Parameters

- Input Vector = 7 x 7
- Kernel = 3 X 3
- Number of kernels = 5 (it means that we'll have 5 activation maps)
- Number of parameters in this layer?

- Answer = 50 parameters
  - Explanation
    - 3 X 3 = 9 + 1 (bias) = 10 x 5 = 50

Convolution Layer – Output Size

- For Convolution Layer: Suppose
  - Input size : W_1 x H_1 x D_1
  - Kernel size: K
  - Stride: S
  - Output: W_2 x H_2 x D_2
  - It is calculated as:

$$W_2 = \frac{W_1 - K}{S} + 1 \text{ and } H_2 = \frac{H_1 - K}{S} + 1$$

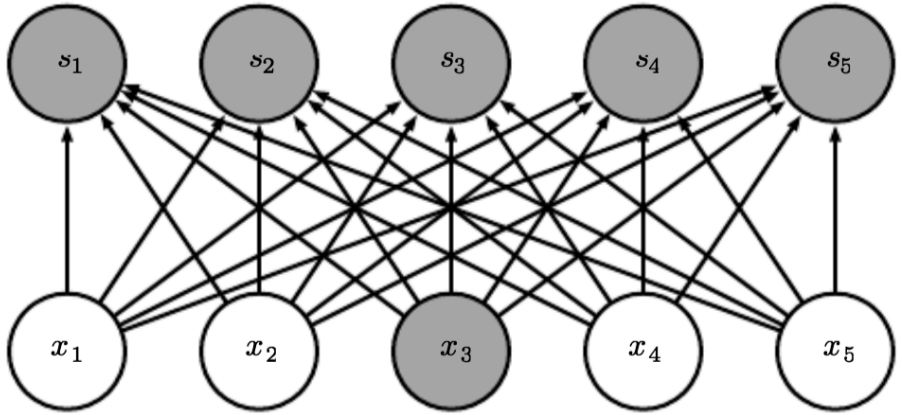
  - **Note:** depth will be the same
  - What will happen if we do padding? Does Formula Change?

$(w_1 + 2*\text{padding} - K)/S + 1$

Why Convolution Operation?

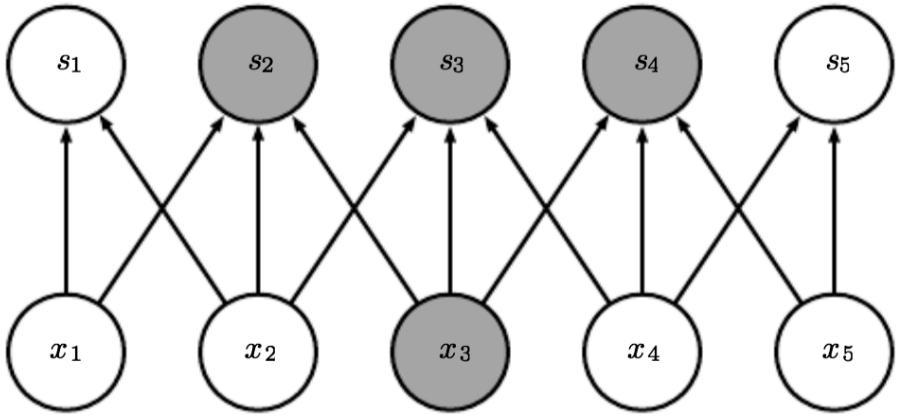
- Convolution leverages three important ideas that can help improve a machine learning system
  - Sparse interactions
  - Parameter sharing
  - Equivariant

Sparse Interaction



No Sparse Connectivity

When s is formed by matrix multiplication, connectivity is no longer sparse, so all the outputs are affected by x3

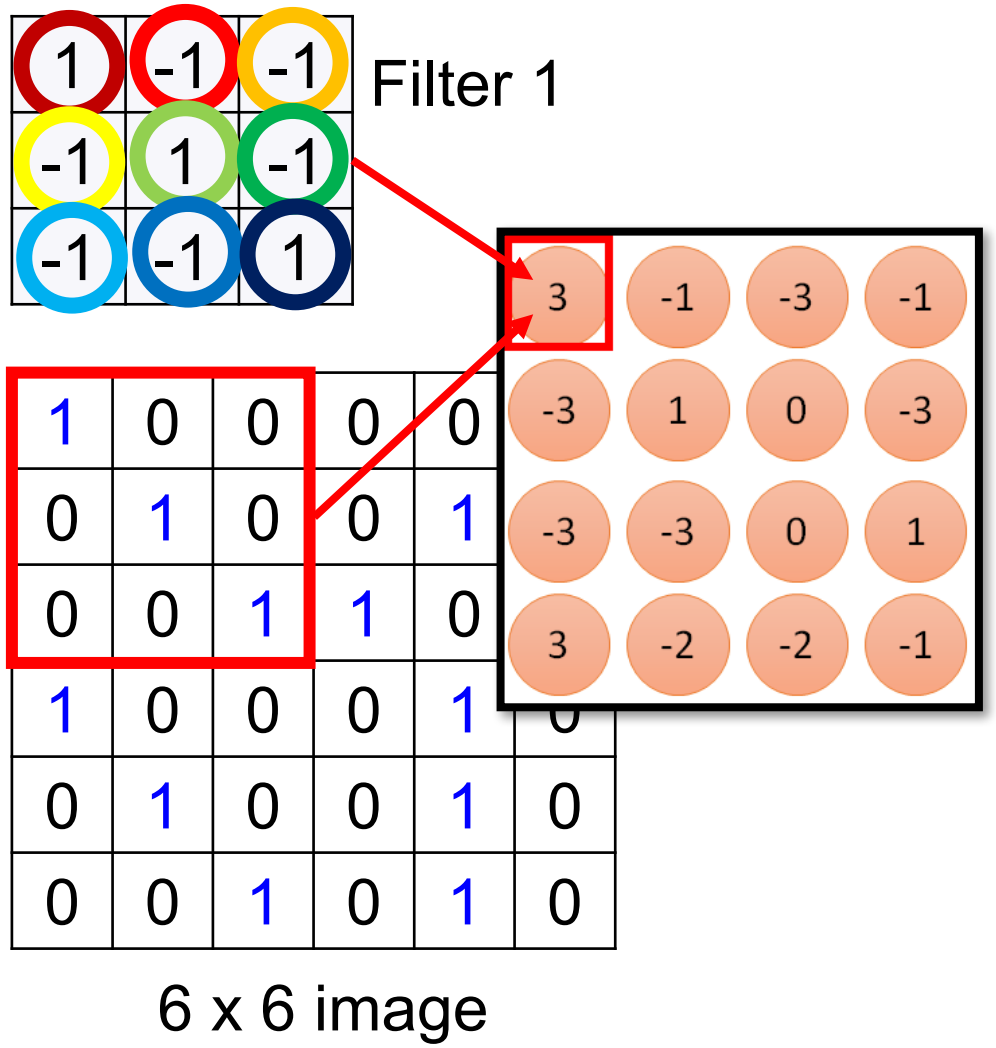


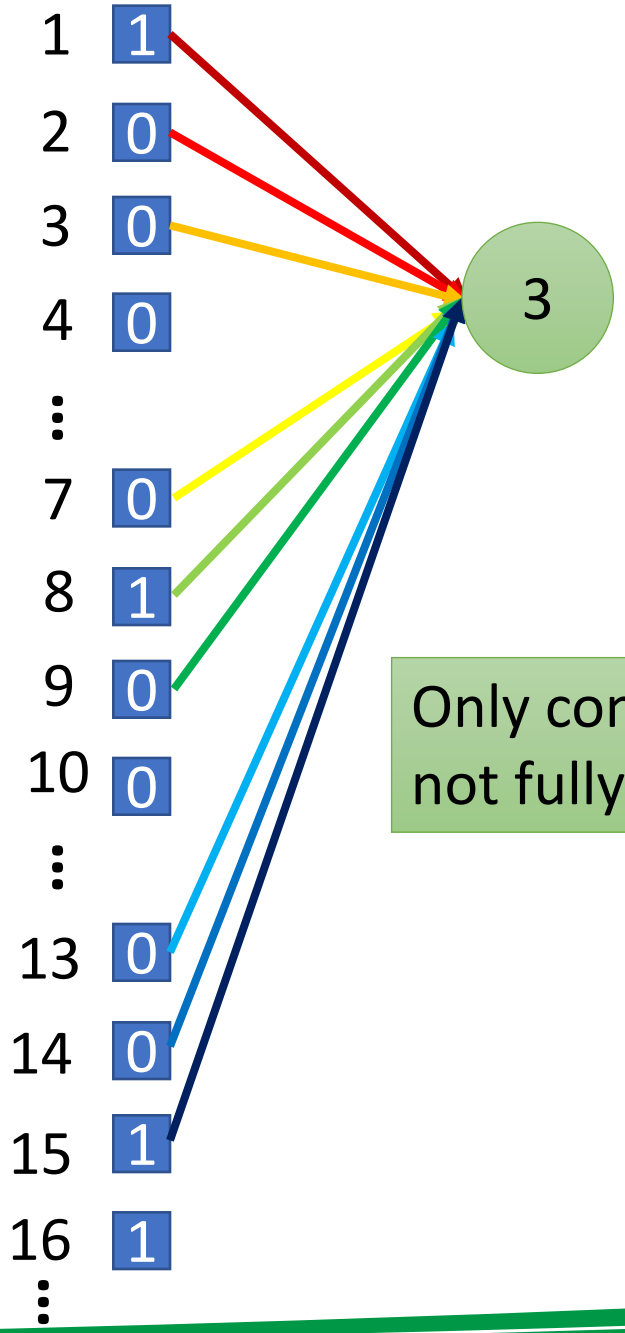
Sparse Connectivity

When s is formed by convolution with a kernel of width 3, only three outputs are affected by x

It is also referred as sparse connectivity or sparse weights.

Sparse Interaction





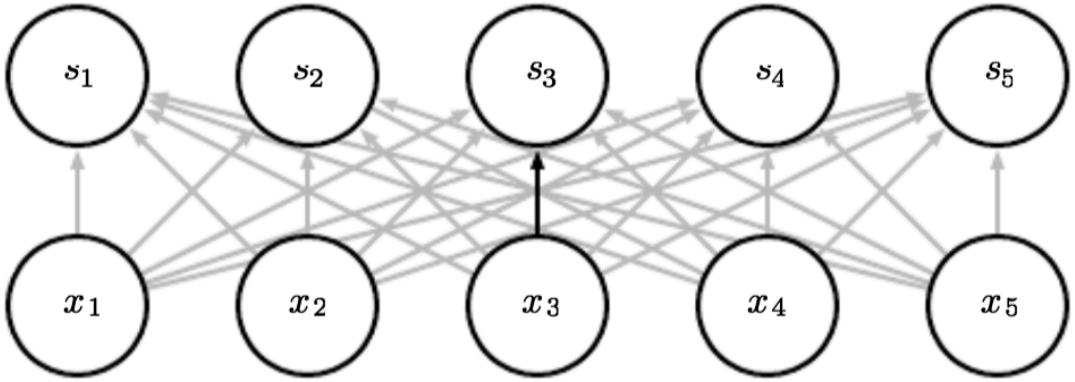
Only connect to 9 inputs, not fully connected

Remarks on Sparse Interaction

- It means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency.

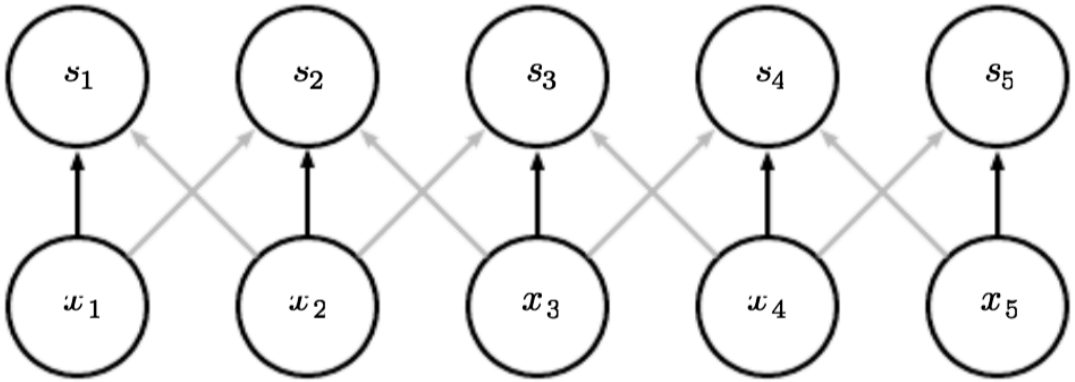
- Fewer operation requires to compute the output

Parameter Sharing



No Sharing between Weights

The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing, so the parameter is used only once

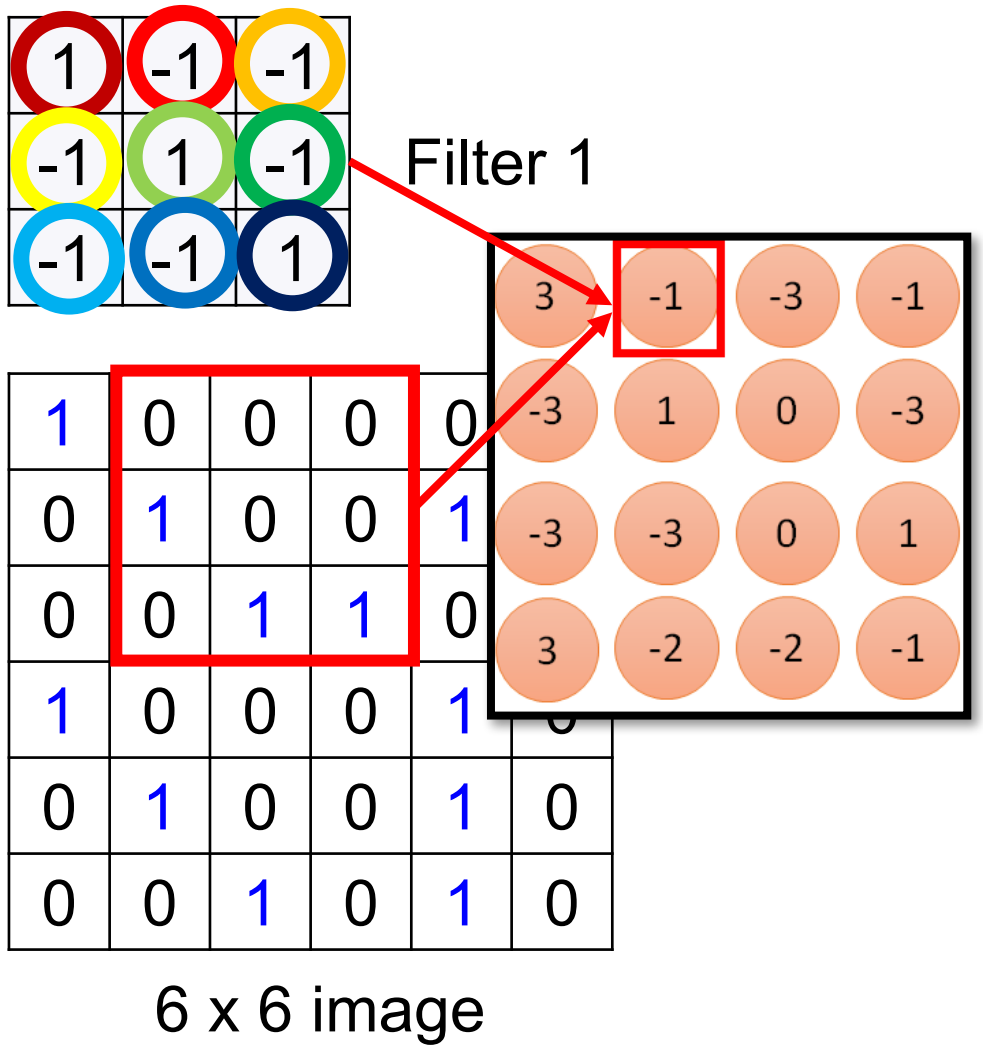


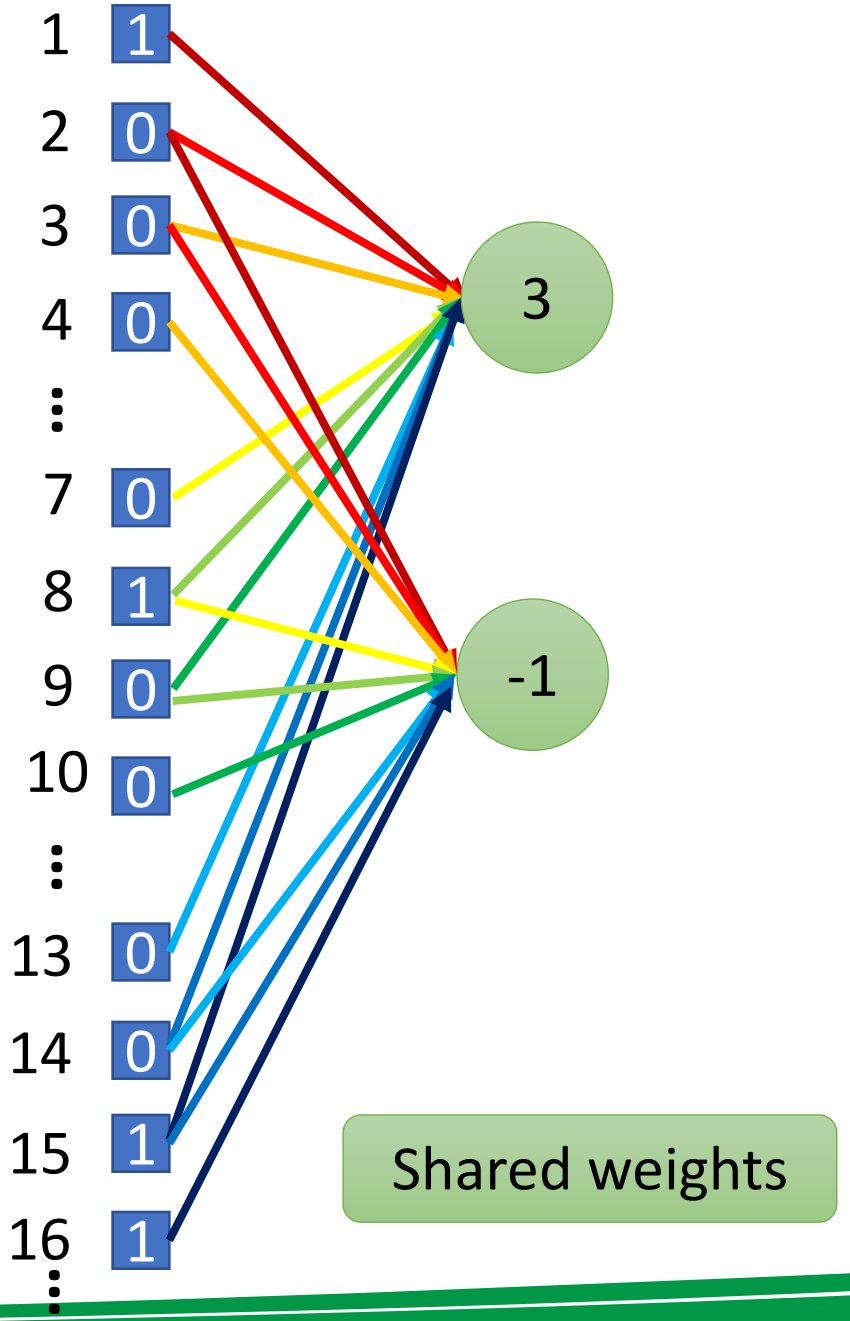
Shared Weights

The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Because of parameter sharing, this single parameter is used at all input locations.

- As a synonym for parameter sharing, one can say that a network has tied weights

Parameter Sharing



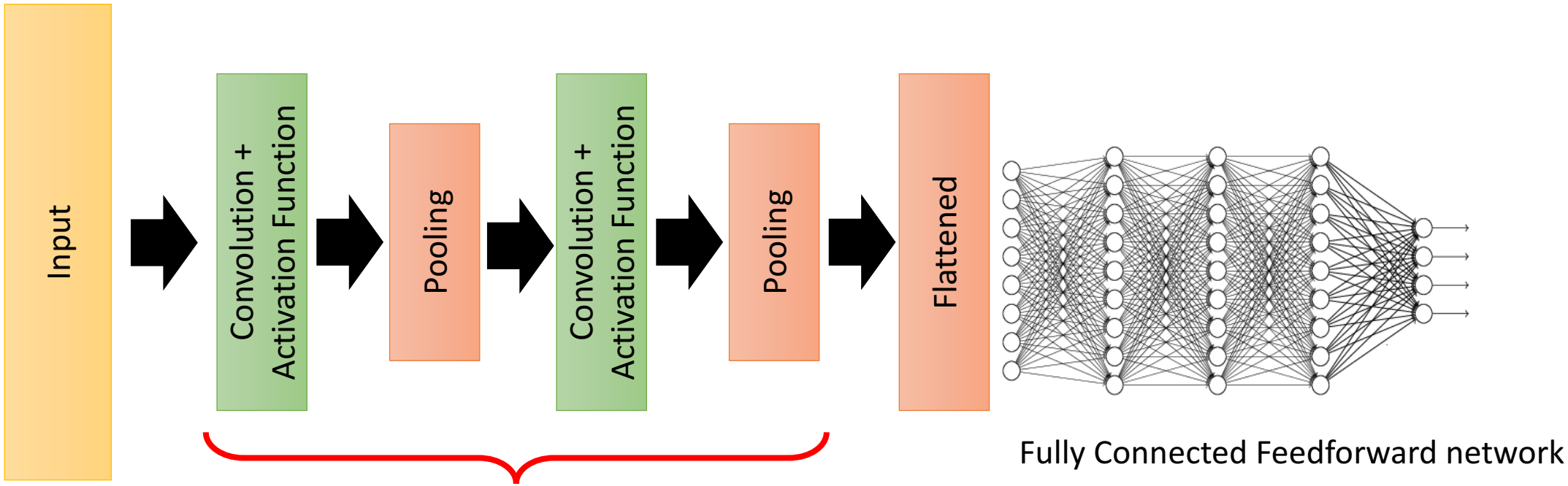


Equivariant

- In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance

- To say a function is equivariant means that if the input changes, the output changes in the same way

A Complete Picture



NOTE:

- It is not necessary you have pooling layer after each convolution.
- It is design choice of the architecture.

A Complete picture

- A typical layer of a convolutional network consists of three stages
- **First stage:** The layer performs several convolutions in parallel to produce a set of linear activations.
- **Second stage:** Each linear activation is run through a nonlinear activation function, such as ReLU. This stage is sometimes called the detector stage.
- **Third stage:** A pooling function to modify the output of the layer further.

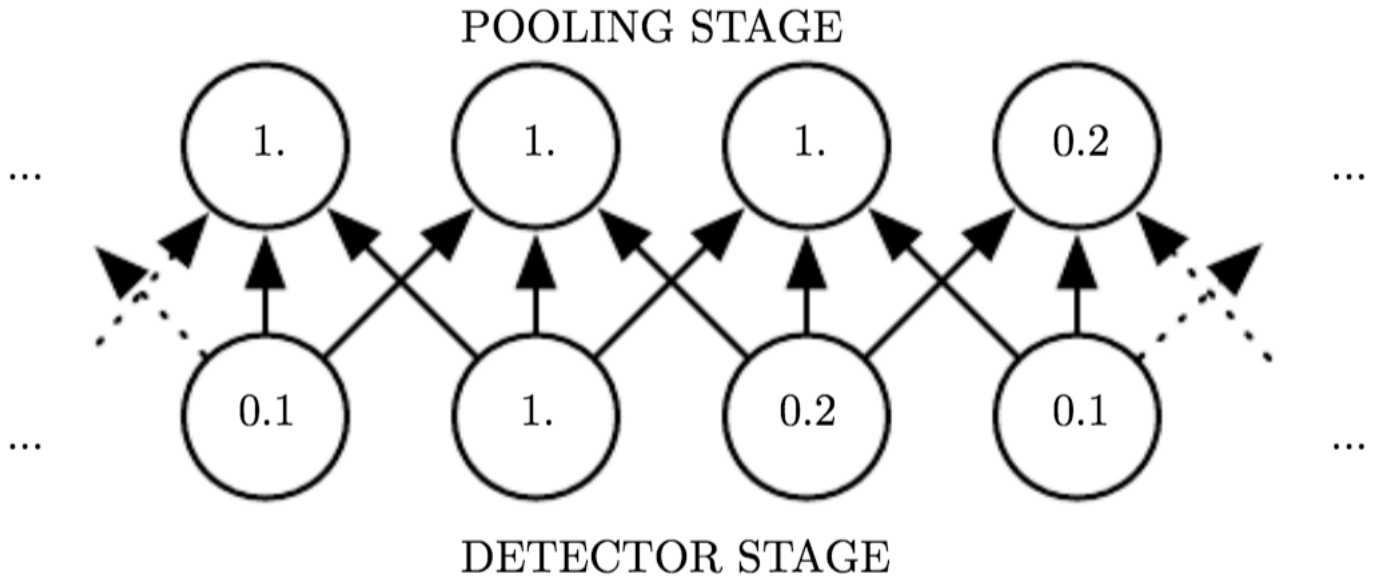
Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.

- It reduce the number of parameters in your network – also known as "down-sampling"

- **Popular pooling functions**
  - Max pooling
  - Average of a rectangular neighborhood
  - A weighted average based on the distance from the central pixel
  - Many others…

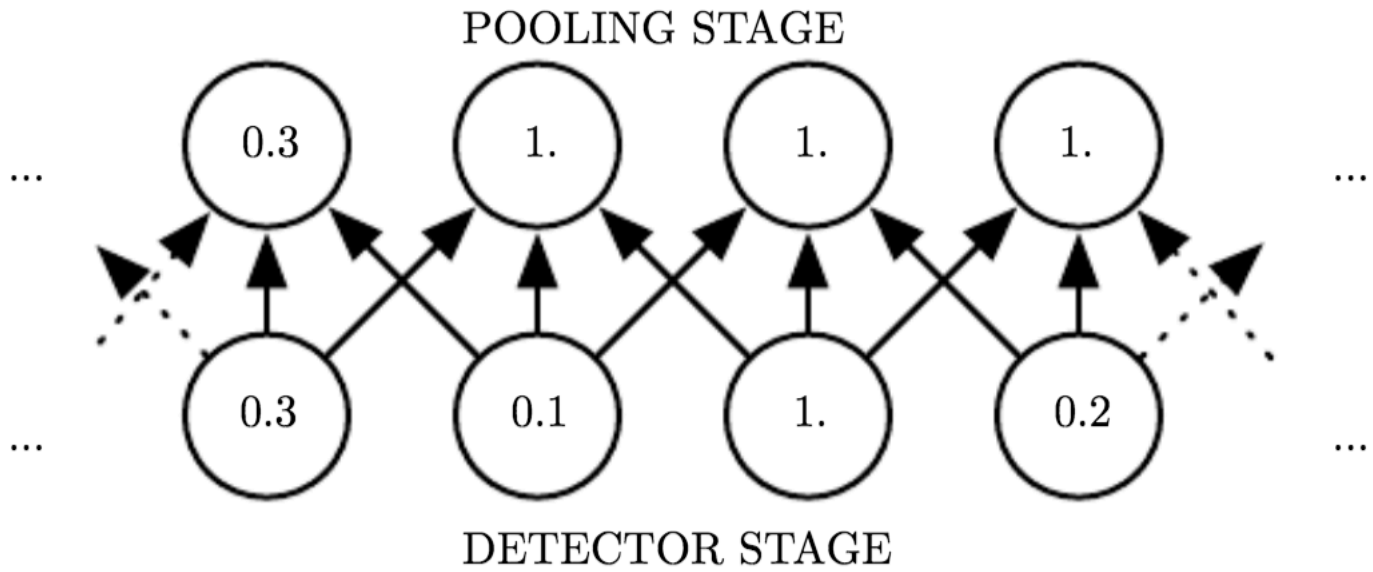
Max Pooling



- A view of the middle of the output of a convolutional layer.
- The bottom row shows outputs of the nonlinearity.
- The top row shows the outputs of max pooling, with pooling region width of three.

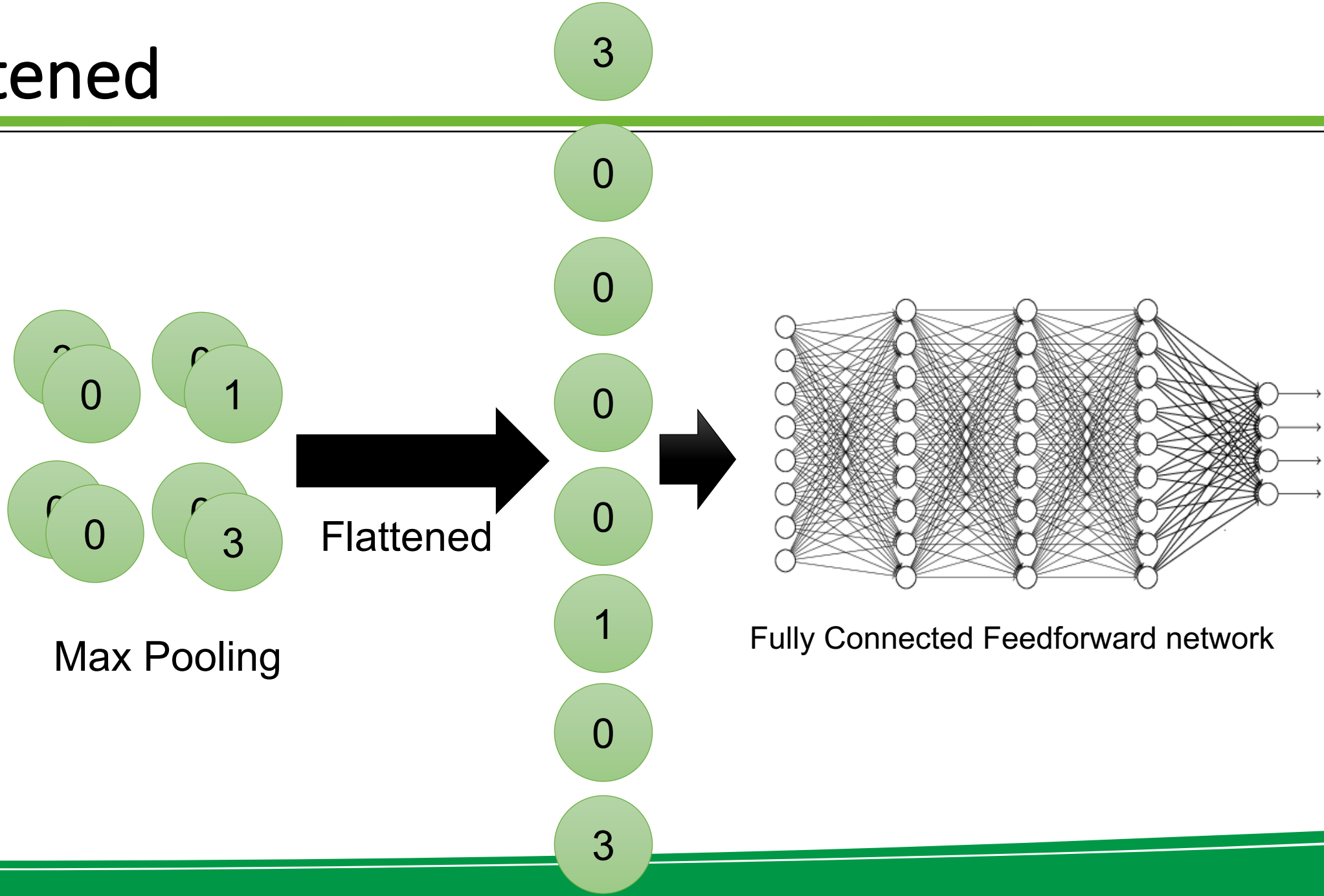
Max pooling introduces invariance

Max Pooling



- A view of the same network, after the input has been shifted to the right by stride 1.
- Every value in the bottom row has changed, but only half of the values in the top row have changed
  - **Reason:** Because the max pooling units are sensitive only to the maximum value in the neighborhood, not its exact location.

Flattened

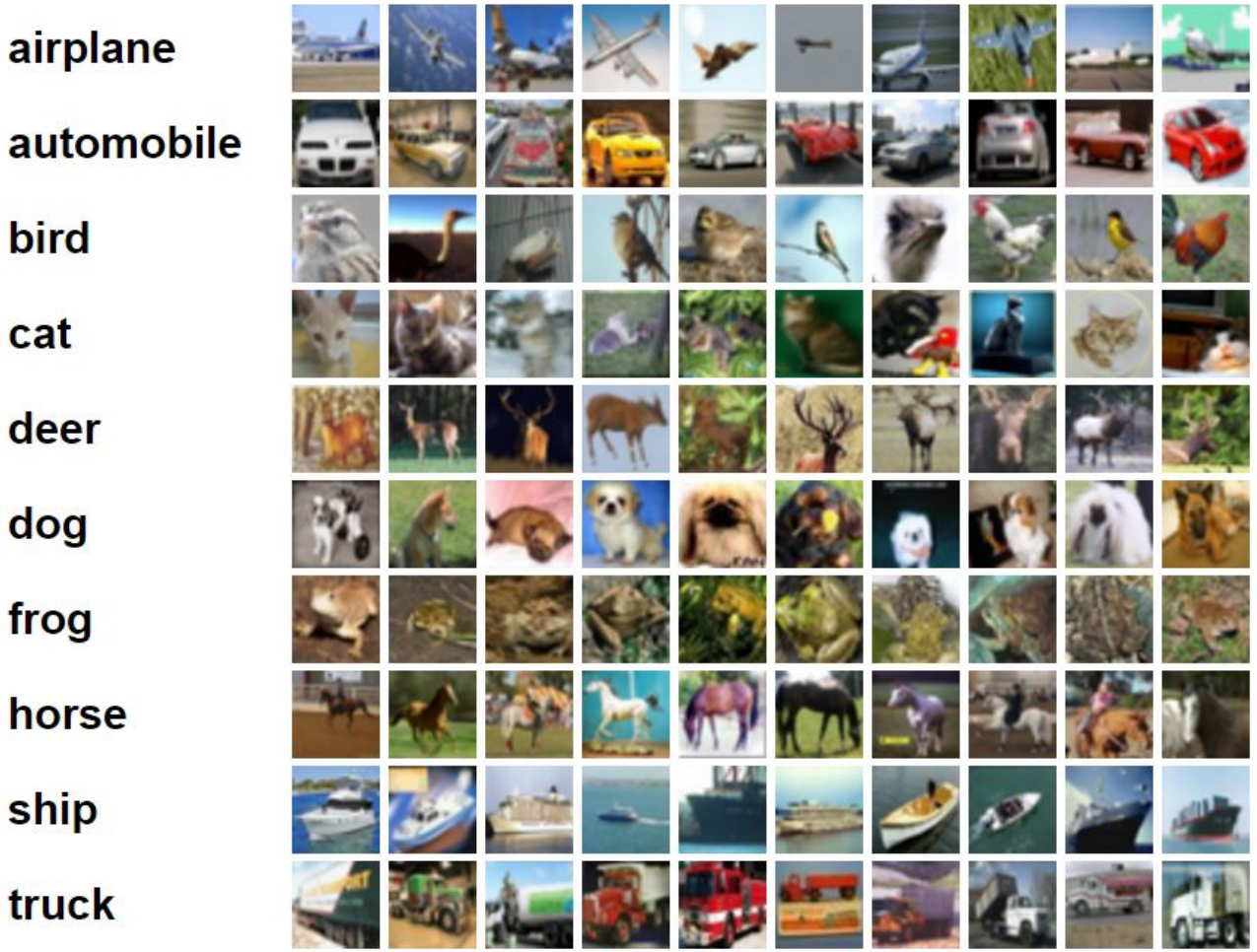

0
1
0
3
Max Pooling
Flattened
3
0
0
0
0
1
0
3
Fully Connected Feedforward network

Check Point!!

- Each Layer may or may not have parameters
  - CONV/FC **(do)**
  - RELU/POOL **(don’t)**

- Each Layer may or may not have additional hyperparameter
  - CONV/FC/POOL **(do)**
  - RELU **(don’t)**

Example: CIFAR-10 image dataset



CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Self-Check

Top 1% Error

Top 5% Error

Famous Datasets

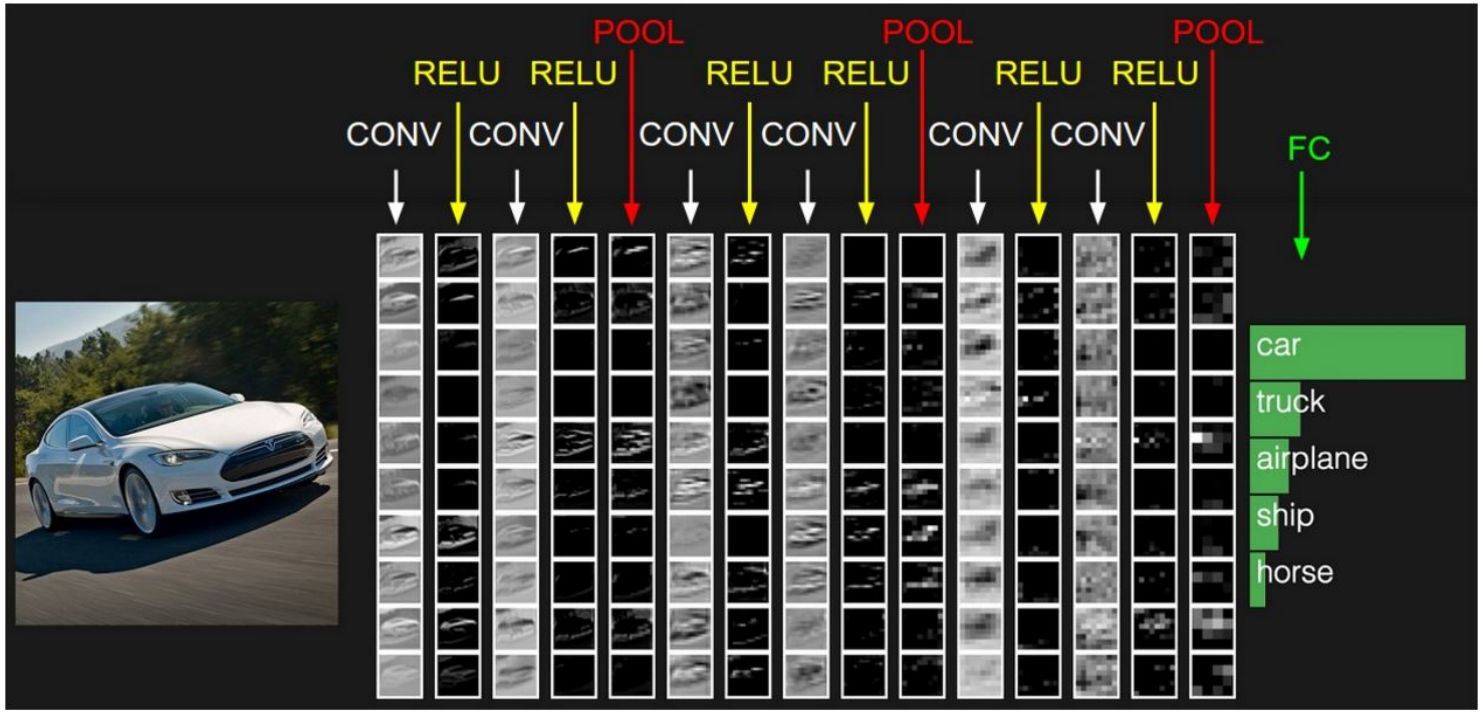
MNIST

ImageNet

CIFAR-10 / CIFAR-100

Places

Example (CIFAR-10 images)



Evolution of CNN – ImageNet (ILSVR)

- AlexNet (2012): 15.4%
- ZFNet (2013): 11.2%
- VGGNet (2014): 7.3%
- GoogLeNet (2014): 6.7%
- ResNet (2015): 3.57%
- CUImage (2016): 2.99%
- SENet (2017): 2.251%

Evolution of CNN – LeNet, AlexNet

- **LeNet**
  - The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990
    - **Application Area:** Used to read zip codes, digits, etc.

- **AlexNet**
  - The first work that popularized CNN in Computer Vision was the AlexNet
    - Developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton.
  - The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012.
  - It is deeper, bigger, and featured Convolutional Layers stacked on top of each other

Evolution of CNN — ZFNet, VGGNet

- **ZFNet**
  - The ILSVRC 2013 winner was a CNN from Matthew Zeiler and Rob Fergus.
  - Expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.

- **VGGNet**
  - The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet.
  - Its main contribution was in showing that the depth of the network is a critical component for good performance.

Evolution of CNN — ResNet, SENet

- **ResNet**
  - Residual Network was the winner of ILSVRC 2015.
  - It features special skip connections and a heavy use of batch normalization.
  - ResNets are currently by far state of the art CNN models and are the default choice for using ConvNets in practice (as of May 10, 2016).

- **SENet**
  - Squeeze-and-Excitation Network (state of the art in 2017)
    - **It adds a parameter to each channel in a convolutional block** so that the network can adjust the weighing of each feature map.
    - This allows to parametrize every step of the network.
  - This method of adding parameters can be applied to other architectures

Training Nets

1. Preprocessing
2. Activation functions
3. Weight initialization
4. Regularization
5. Hyperparameter optimization
6. Evaluation model ensembles

1. Data Preprocessing

- **Mean subtraction** – It has the geometric interpretation of centering the data around the origin along every dimension

- **Normalization** – It refers to normalizing the data dimensions so that they are of approximately the same scale

- **Principal Component Analysis** – the data is first normalized. Then, we can compute the covariance matrix that tells us about the correlation structure in the data

3. Activation Function

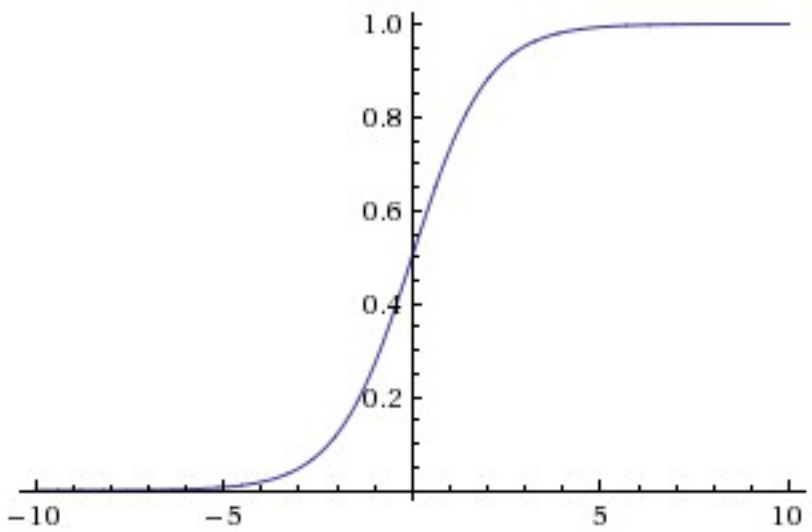
- Activation functions are an extremely important feature of the artificial neural networks.
- They basically decide whether a neuron should be activated or not.
- Every activation function (or *non-linearity*) takes a single number and performs a certain fixed mathematical operation on it.
- The nonlinear activation functions are mainly divided on the basis of their range or curves

Activation Function – Sigmoid or Logistic

- The sigmoid is non-linearity activation function.
- Mathematically:

$$f(z) = \frac{1}{1 + e^{(-z)}}$$

  - It takes a real-valued number and “squashes” it into range between 0 and 1
  - In particular, large negative numbers become 0 and large positive numbers become 1.



- **Drawbacks**
  - *Sigmoids saturate and kill gradients*.
  - Sigmoid outputs are not zero-centered.
  - exp() is a bit compute expensive

The **softmax function** is a more generalized logistic activation function

Activation Function – Softmax

- It converts linear inputs to probabilities
- It will calculate the probabilities of each target class over all possible target classes
- Mathematically:

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \qquad i = 1,2,3,..j$$

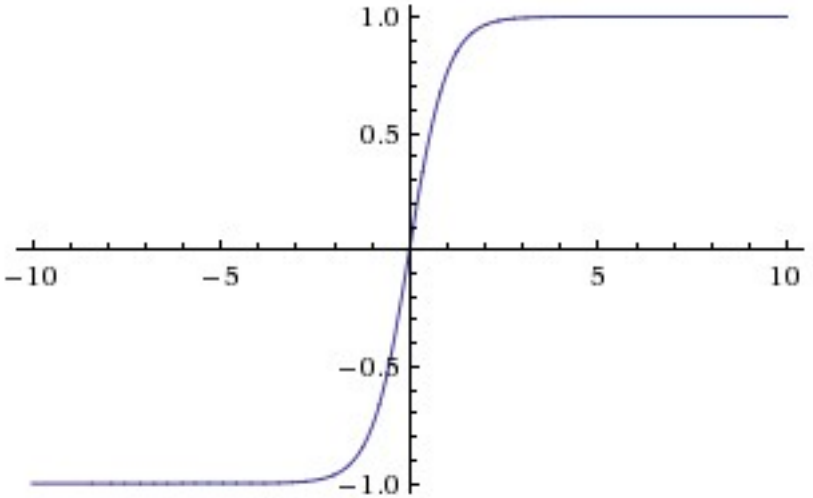
- **Properties**
  - The calculated probabilities will be in the range of 0 to 1.
  - The sum of all the probabilities is equals to 1.
  - The high value of *'z'* have the high probability.

Difference Between Sigmoid Function and Softmax Function

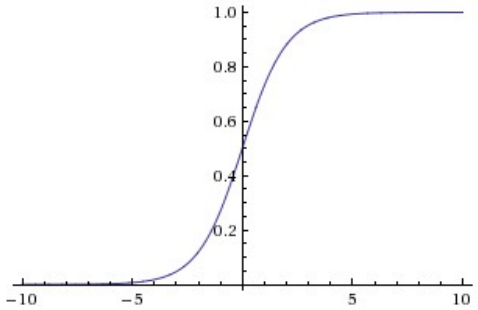
Softmax Function	Sigmoid Function
The probabilities sum will be 1	The probabilities sum need not be 1.
Used in the different layers of neural networks.	Used as activation function while building neural networks.
The high value will have the higher probability than other values.	The high value will have the high probability but not the higher probability.

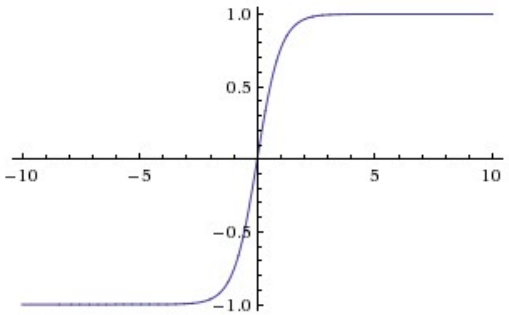
Activation Function – Tanh

- It squashes a real-valued number to the range [-1, 1].
- Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered.
- Therefore, in practice the *tanh* non-linearity is always preferred to the sigmoid nonlinearity.



- **Note:** *tanh* neuron is simply a scaled sigmoid



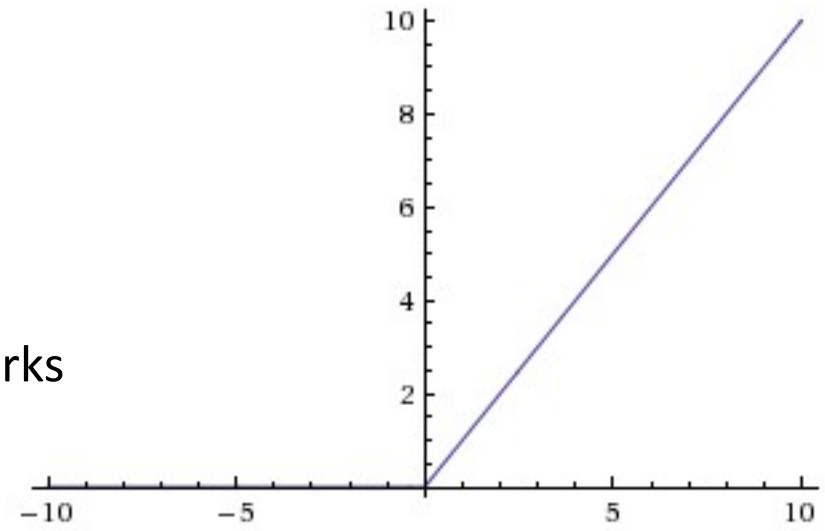


$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Activation Function – ReLU

- The Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$



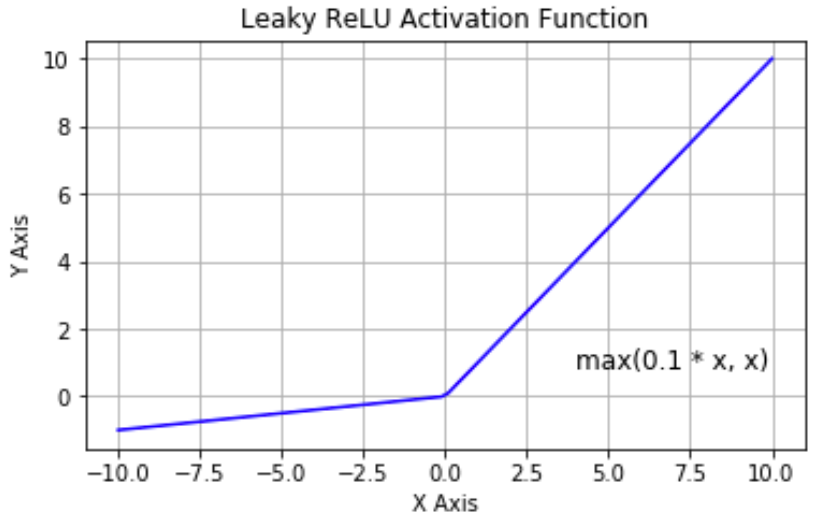
  - ReLu is a very popular – especially with Deep Neural networks

- **Pros:**
  - It learns faster and avoids vanishing gradient problem
- **Cons:**
  - It should only be used in hidden layers of a Neural Network.
  - ReLu could result in dead Neurons.

Activation Function – Leaky ReLU

- It is a modified version of ReLu.
- Leaky ReLu solves the problem of dying neurons.
- **How?**
  - By introducing a slight slope that keeps the updates alive.
- Mathematically:

$$\max(0.1x, x)$$



There are so many others activation functions…

3. Weight Initialization

- **First idea:** Small random numbers – sample from a distribution

- Usually standard Gaussian distribution and works well for small networks

- Random initialization have problems with deeper networks.

3. Weight Initialization

- Random initialization have problems with deeper networks.

Solution: “Xavier initialization” [Glorot et al., 2010]

Source: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

Good News: you have this function in TensorFlow.

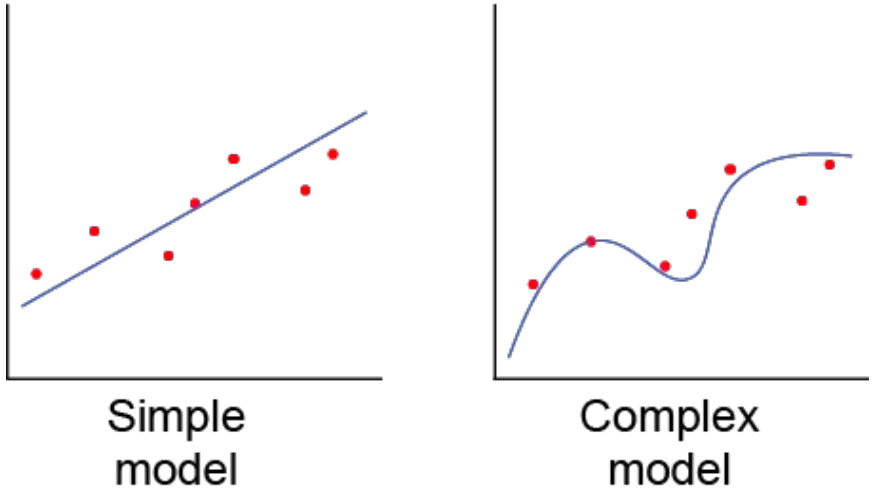
3. Weight Initialization – Check point!!

- What happens when you set all initialize the weights = 0?

- **Answer:**
  - All the neuron will do the same thing
  - Every neuron is going to be the same operation and same gradient.
  - We don’t want to do every neuron the same thing.

4. Regularization

- Regularization is a technique to penalize complex models.



- **Why Regularization?**
  - because simple models generalize better and are less prone to overfitting.

- Many regularization approaches are based on limiting the capacity of models

4. Regularization – Parameter Norm Penalties

- Neural networks, linear regression or logistic regression, by adding a parameter norm penalty Ω(θ) to the objective function J

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

$$\alpha \in [0, \infty)$$

- Setting α to 0 results in no regularization
- Larger values of α correspond to more regularization

Other Techniques

Dropout
Data Augmentation
Batch Normalization
Early stopping etc..

L^2 Parameter Regularization

- Also known as ridge regression or Tikhonov regularization

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

- The above term penalizes big coefficients and tries to minimize them to zero, although not making them exactly to zero

- The L^2 parameter norm penalty commonly known as **weight decay**

L^2 Parameter Regularization

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

Assume no bias parameter so θ is just w (for simplicity)

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

A single gradient step to update the weights

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\left(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})\right)$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Another Notation

$$w_i \leftarrow w_i - \eta\frac{\partial E}{\partial w_i} - \eta\lambda w_i$$

L^1 Regularization

- L^1 has one small change in the cost equation, it takes the absolute value.

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|$$

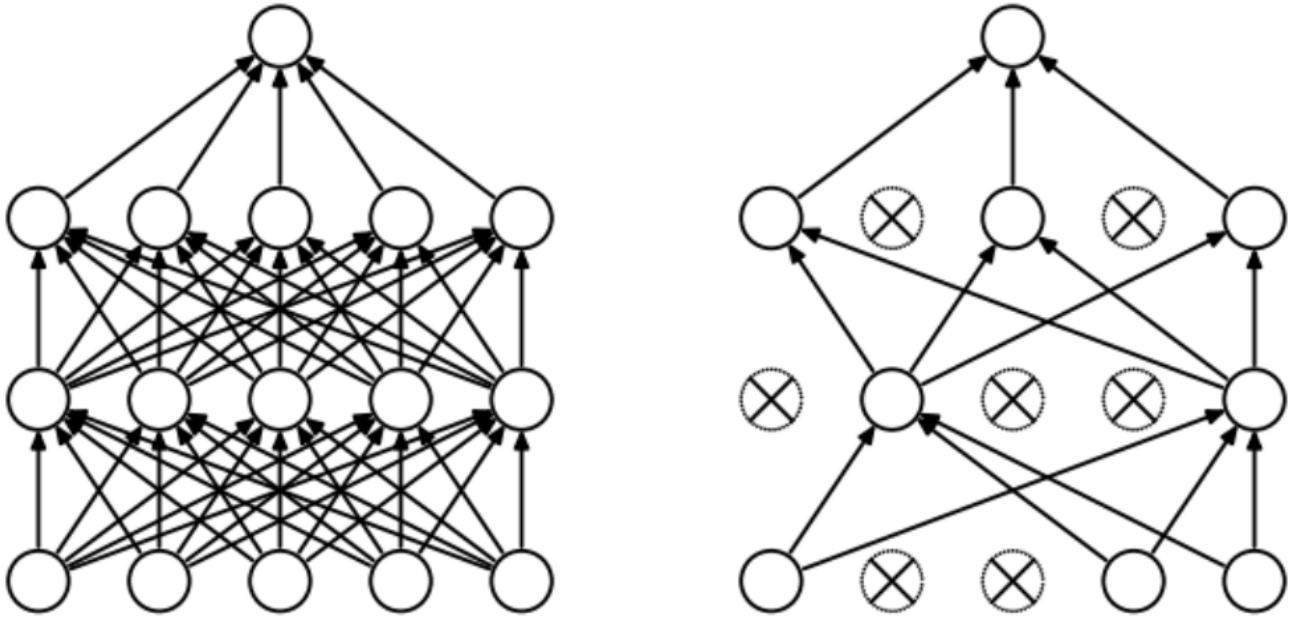
$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$$

- L1 regularization results in a solution that is more **sparse**
- Sparsity in this context refers to the fact that some parameters have an optimal value of zero.
- L1 regularization has been used extensively as a feature selection mechanism.

4. Regularization – Drop out

- It is a regularization technique for reducing overfitting
- In each forward pass, randomly set some neuron to zero
- Probability of dropping neuron is a hyperparameter (i.e., 0.5 is common)



(a) Standard Neural Net

(b) After applying dropout.

4. Regularization – Data Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.

- In practice (if) the amount of data is limited.
  - One way to get around this problem is to create fake data and add it to the training set.
  - There is no general recipe regarding how the synthetic data should be generated and it varies a lot from problem to problem.

4. Regularization – Data Augmentation

- Data augmentation increases the number of training examples
- It also increase the robustness against input variability.
- For images some common techniques include flipping, rotation, scaling, contrast, brightness etc.
- Injecting noise in the input signals to a neural network can also be seen as a form of data augmentation
  - For Example: Denoising autoencoder (Unsupervised learning algorithm)

Data Augmentation – Remarks!

- One must be careful not to apply transformations that would change the correct class.

- **For example:**
  - Optical character recognition tasks require recognizing the difference between “b” and “d” and the difference between “6” and “9”.
  - So horizontal flips and 180◦ rotations are not appropriate ways of augmenting datasets for these tasks.

4. Regularization – Early Stopping

- It is probably the most commonly used form of regularization in deep learning.

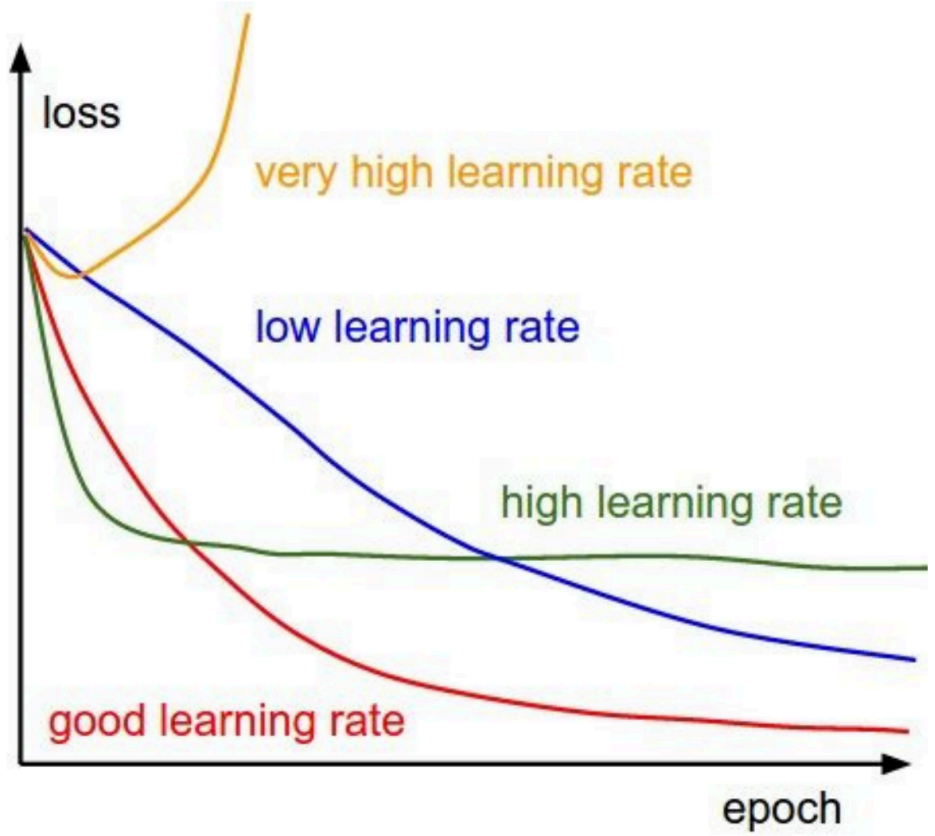
- Its popularity is due to both its effectiveness and its simplicity
- Early stopping may be used either alone or in conjunction with other regularization strategies

5. Learning Rate

- Find learning rate that makes the loss go down.

- **Loss is not going down**
  - Learning rate too low

- **Loss exploding**
  - Learning rate too high



6. Hyperparameter Optimization

- **First stage:** Only a few epochs to get rough idea of what parameters work
- **Second stage:** longer running time, finer search … (repeat as necessary)

7. Evaluation Model Ensembles

- Bagging (short for bootstrap aggregating) is a technique for reducing generalization error by combining several models.
- The idea is to train several different models separately, then have all the models vote on the output for test examples.
- This is an example of a general strategy in machine learning called model averaging.
- Techniques employing this strategy are known as ensemble methods
- Why it works?

7. Evaluation Model Ensembles

- Train multiple independent models
  - Instead of training independent models, use multiple snapshots of a single model during training!
  - Differences in random initialization, in random selection of minibatches, in hyperparameters, or in outcomes of nondeterministic implementations of neural networks are often enough to cause different members of the ensemble to make partially independent errors.

- At test time average their results

Remarks

- Model averaging is an extremely powerful and reliable method for reducing generalization error.

- Its use is usually discouraged when benchmarking algorithms for scientific papers
  - because any machine learning algorithm can benefit substantially from model averaging at the price of increased computation and memory

Application Areas

- ConvNets have been successful applied
  - Identifying faces
  - Objects
  - Traffic signs
  - Vision in robots
  - AlphaGo
  - Self driving cars

Summary

- Information Processing in Neural Networks
- Artificial Neural Network
- Convolutional Neural Network (ConvNets)
- Training Networks
  - Preprocessing
  - Activation Function
  - Weight Initialization
  - Hyperparameter Optimization
  - Evaluation model ensemble

Reference

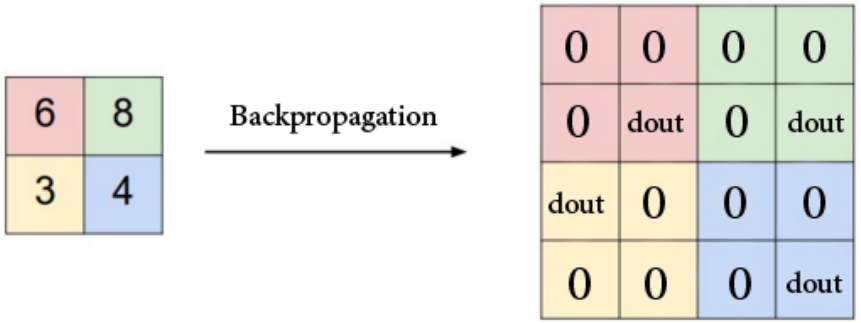
- This lecture is based on the following book chapter and other internet resources.
- Read Chapter 9: http://www.deeplearningbook.org/contents/convnets.html
- http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf
- http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Thank You ☺

Appendix

Backpropagation in Pooling Layer

- We learn that the max node simply act as a router, giving the input gradient "dout" to the input that has value bigger than zero
- So consider the backward propagation of the max pooling layer as a product between a mask containing all elements that were selected during the forward propagation and dout.



- In other words the gradient with respect to the input of the max pooling layer will be a tensor make of zeros except on the places that was selected during the forward propagation. or
- During the forward pass of a pooling layer it is common to keep track of the index of the max activation (sometimes also called *the switches*) so that gradient routing is efficient during backpropagation.