Deep Learning Workshop

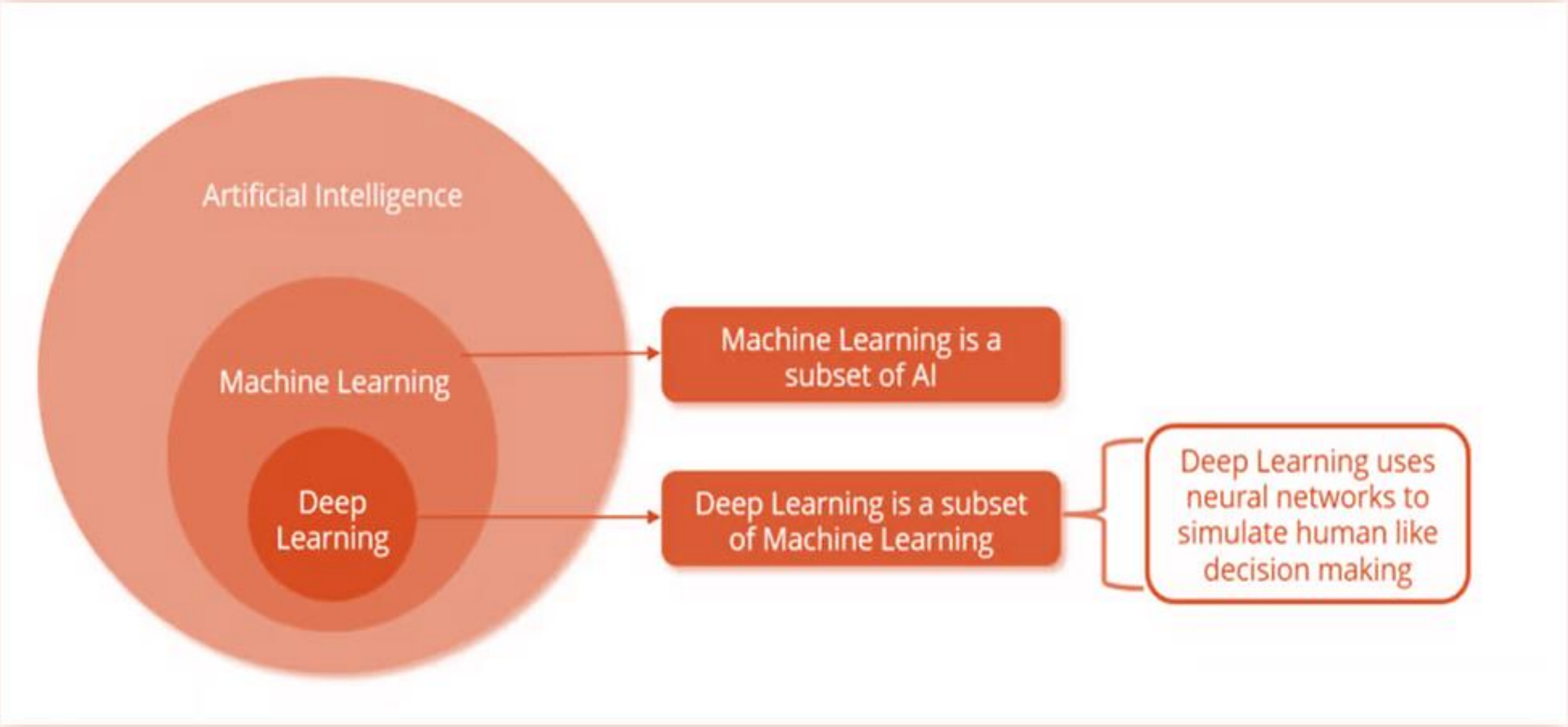
Deep Learning and Computer vision applications

Hind Azegrouz

Over the next 120 minutes, you will learn about..

- The basic concepts of Neural Networks
- Neural Networks Training
- Neural Networks Architectures
- Deep Learning Frameworks
- Deep Learning Applications in Computer vision
    - Convolutional neural networks

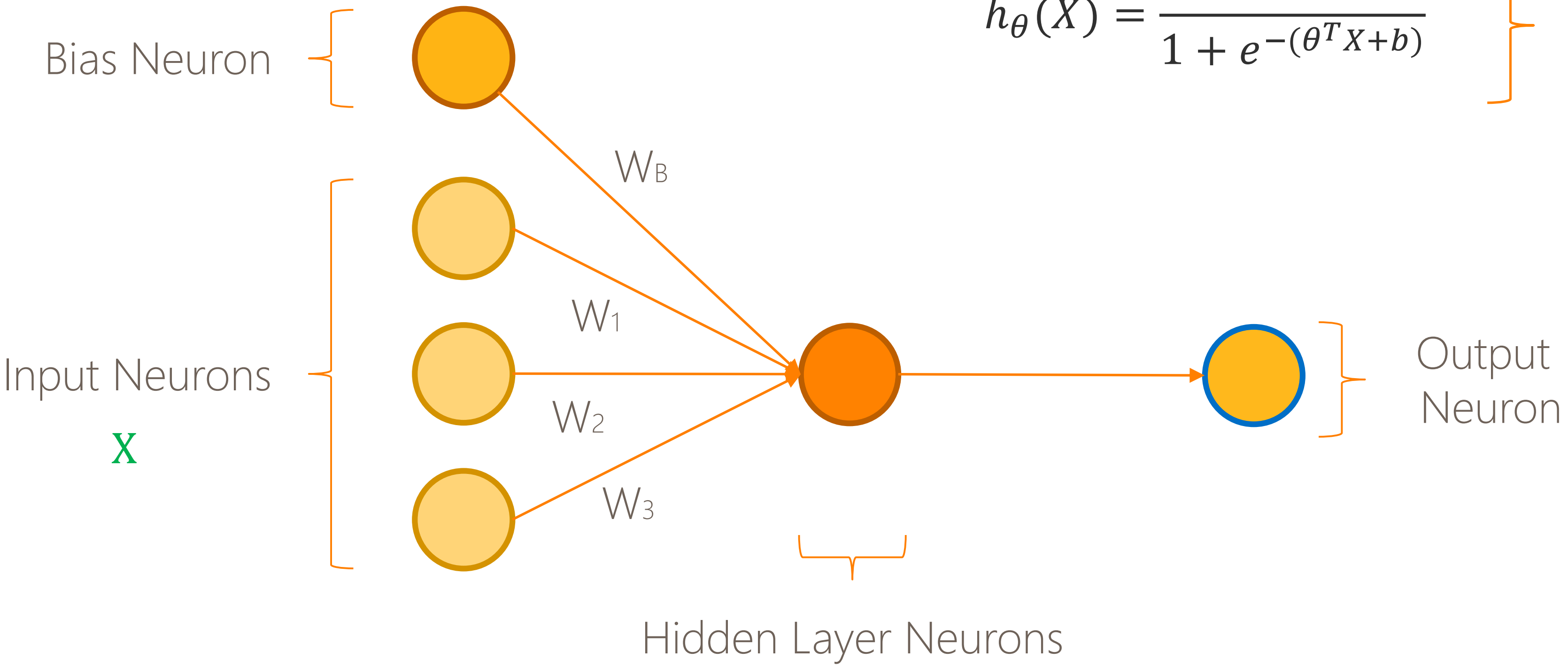
Subsets of Artificial Intelligence



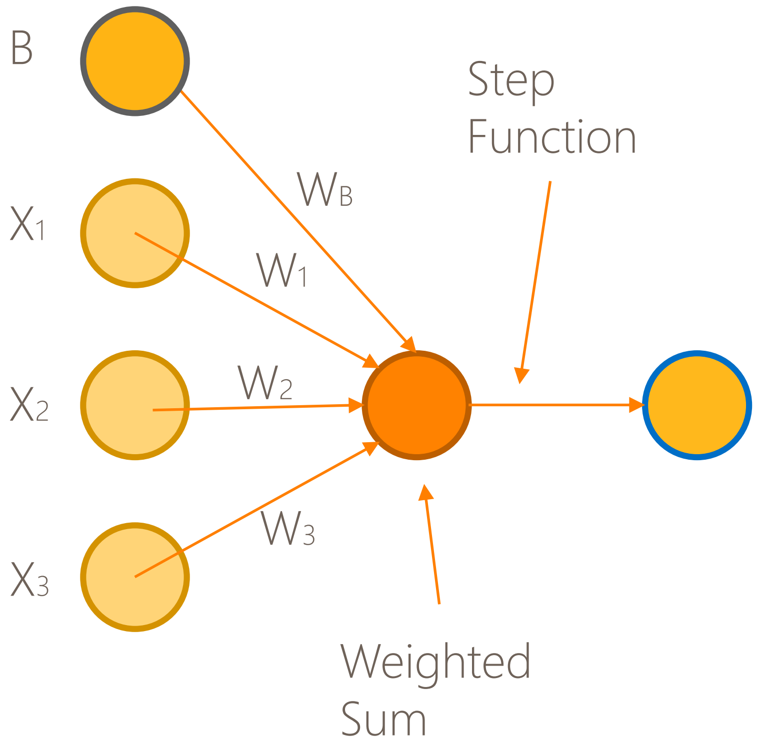
What's a Neural Network anyway?

$$h_\theta(X) = \frac{1}{1 + e^{-(\theta^T X + b)}}$$

Activation Function



Neural Networks: Humble Beginnings (1/2)

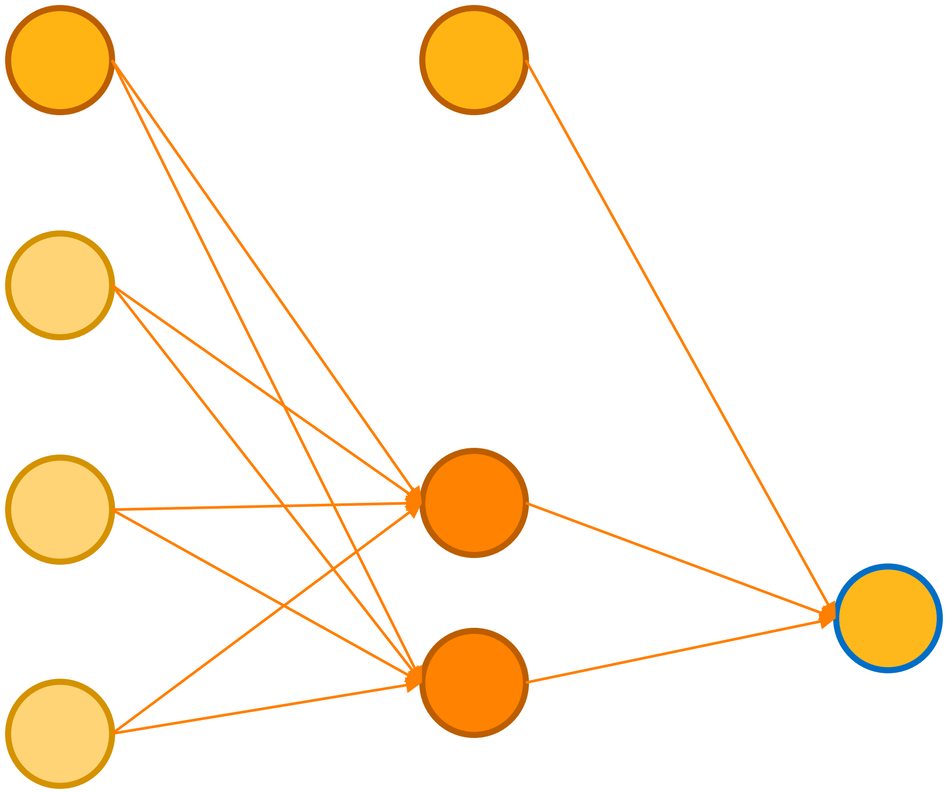


Sum inputs,
apply non-linear activation function,
get output,
simple training procedure

Sounds great… but…

- Simple perceptrons are limited (e.g. XOR which isn't linearly separable)
- Perceptron learning algorithm can run into infinite linear boundaries

Neural Networks: Humble Beginnings (2/2)



Hidden layer with two neurons helps!

Sounds great… but…

- XOR problem still complex to solve with gradient descent (though can be solved)
- Though weight initialization can help

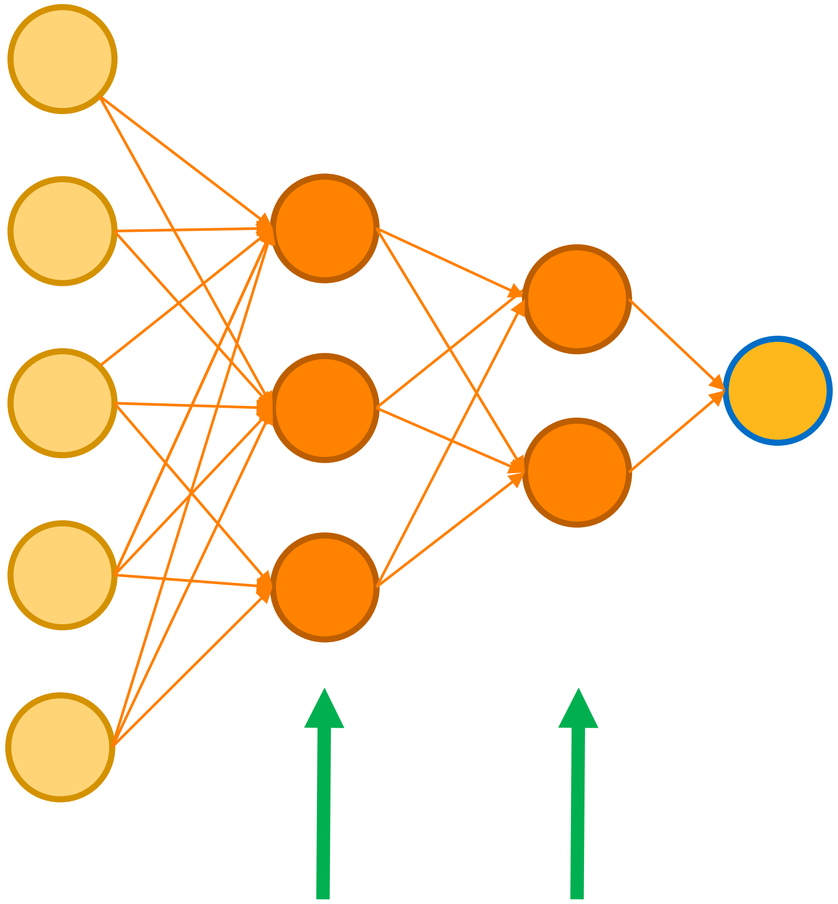
Neural networks ended up going out of fashion in late 1960's

Backpropagation

- History goes back to **optimal control theory** in 1960
- Basic derivation via chain rule in 1962
- Potential applicability to neural networks realized in early 1970's
- Successfully applied in early 1980's
- Shown to generate useful internal representations of data in 1986
- First international pattern recognition contest won via backprop in 1993
- Fell out of favor in 2000's
- Hugely popular today!

Neural Network Training

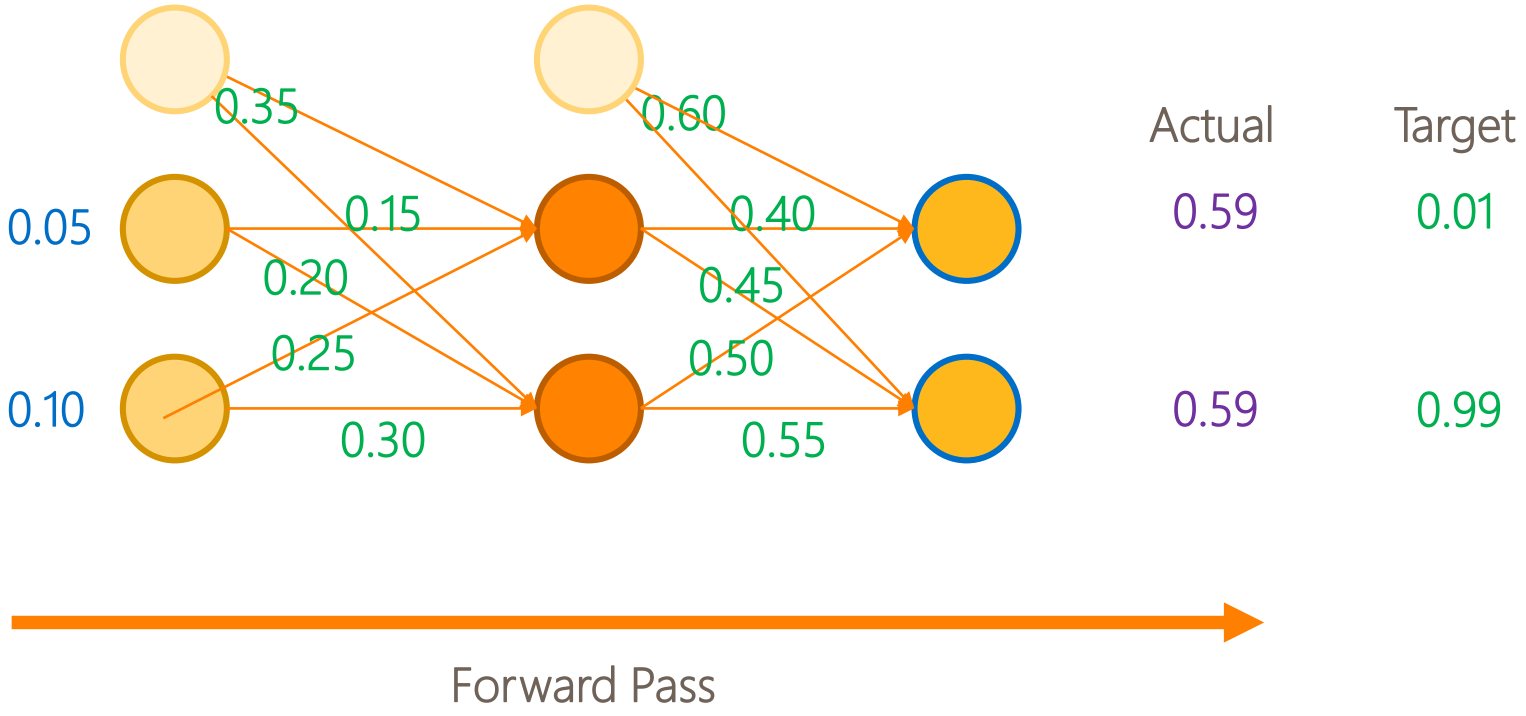
Training Much Deeper Neural Networks



- In 2006, Hinton et al. showed that neural networks, like **Restricted Boltzmann Machines**, stacked into **Deep Belief Nets** could be pre-trained, layer-by-layer, in an **unsupervised** fashion – then – fine-tuned using **supervised** backpropagation

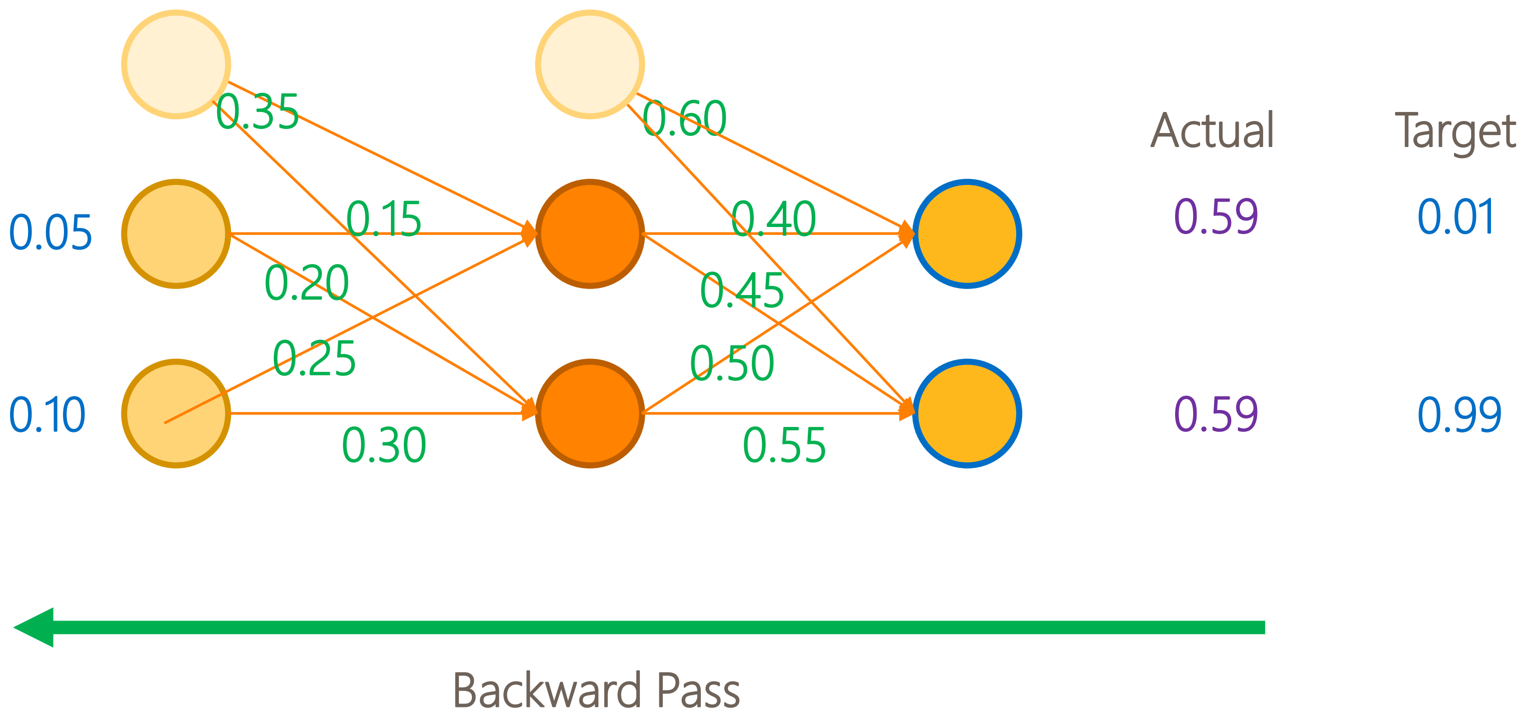
- Pre-training era 'ended' in early 2010's – when other backprop-driven approaches (including ReLUs, dropout, and better weight initialization) were discovered, ultimately reduced need for pre-training
  - Resulted in explosion of backprop-powered deep learning advancements

Backpropagation (1/6)



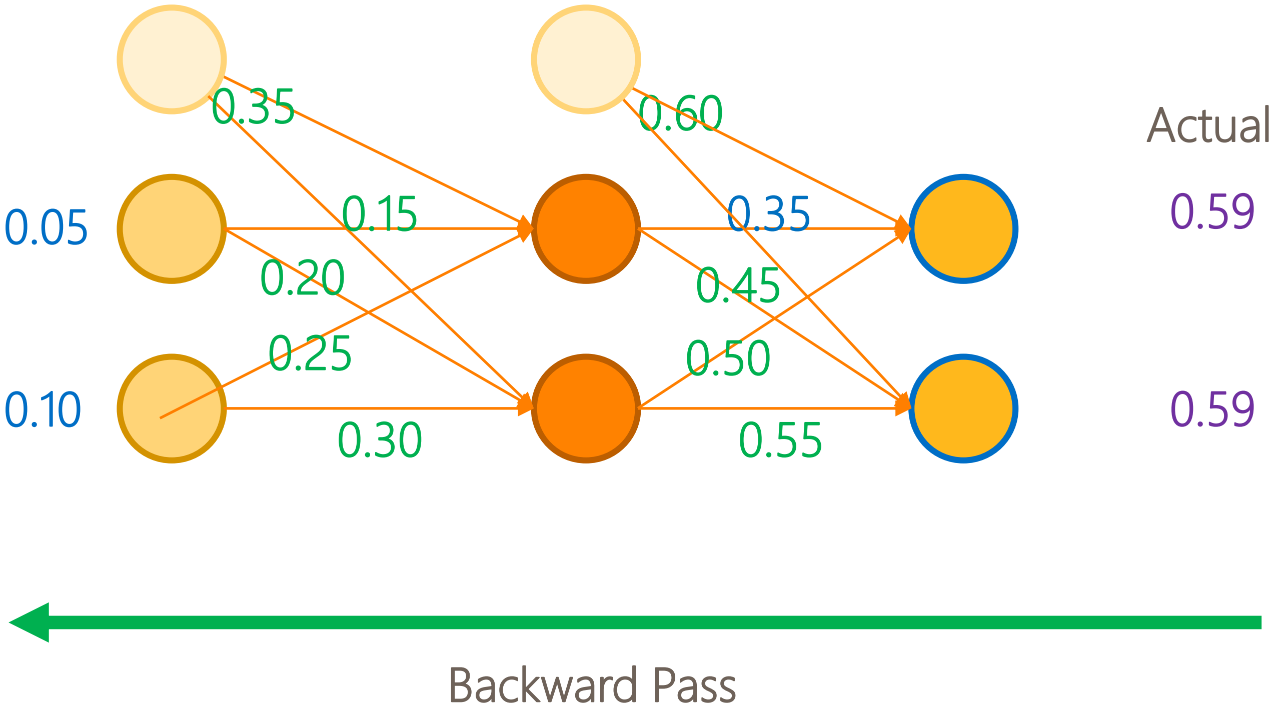
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

Backpropagation (2/6)



$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

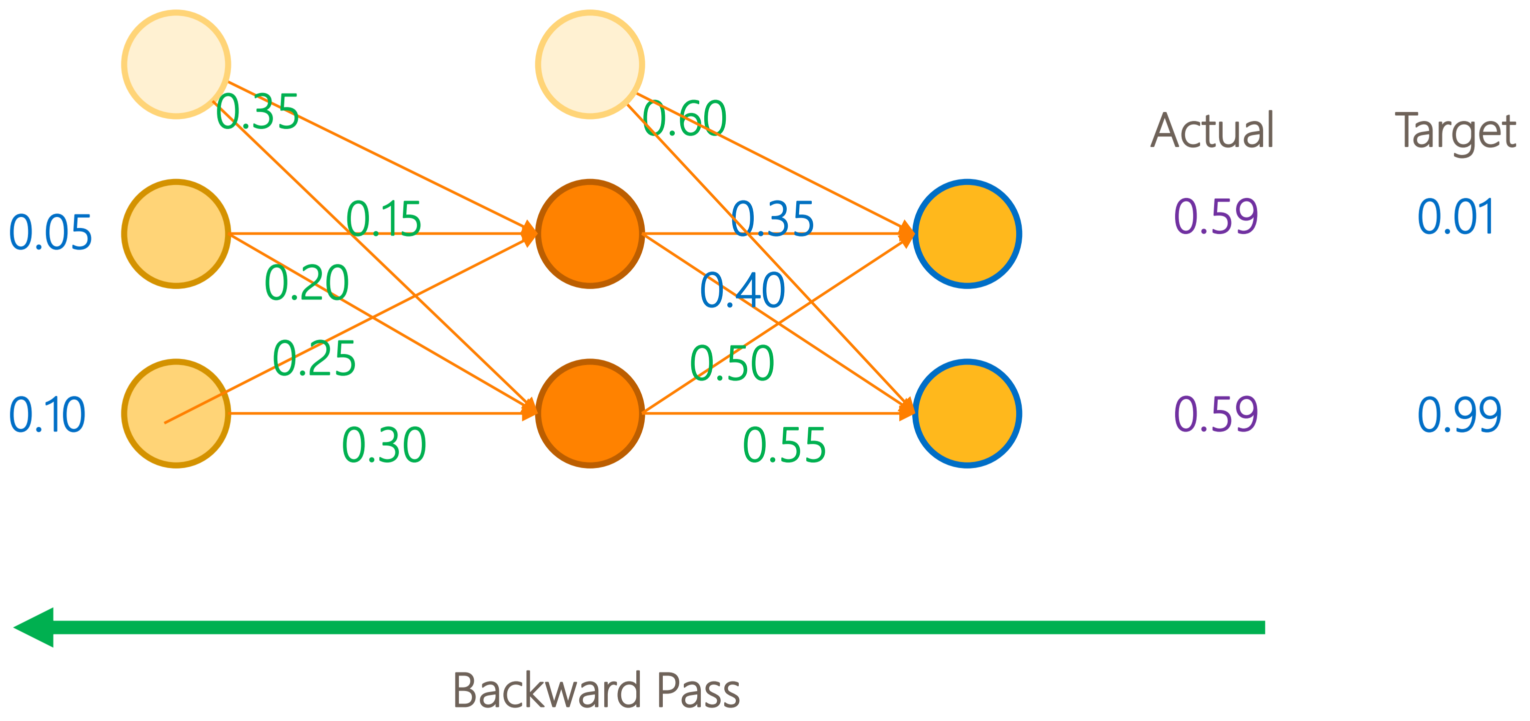
Backpropagation (3/6)



Actual	Target
0.59	0.01
0.59	0.99

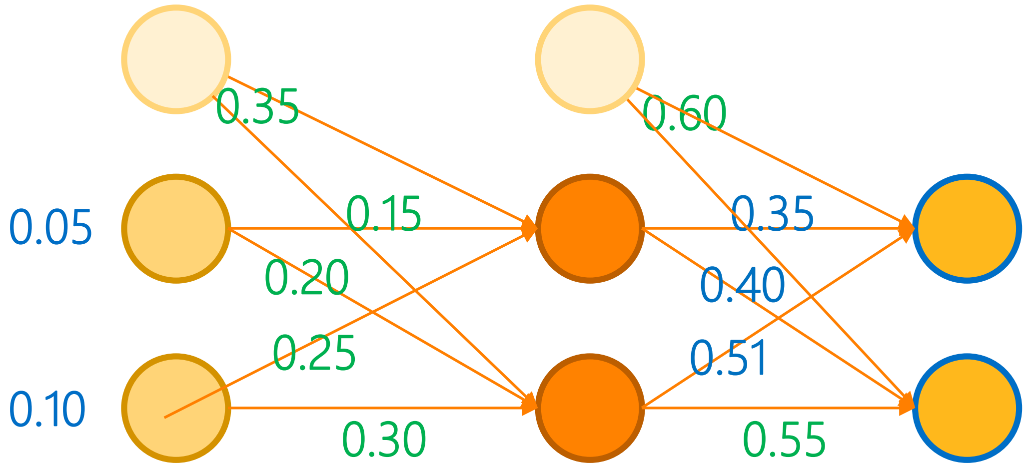
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

Backpropagation (4/6)



$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

Backpropagation (5/6)

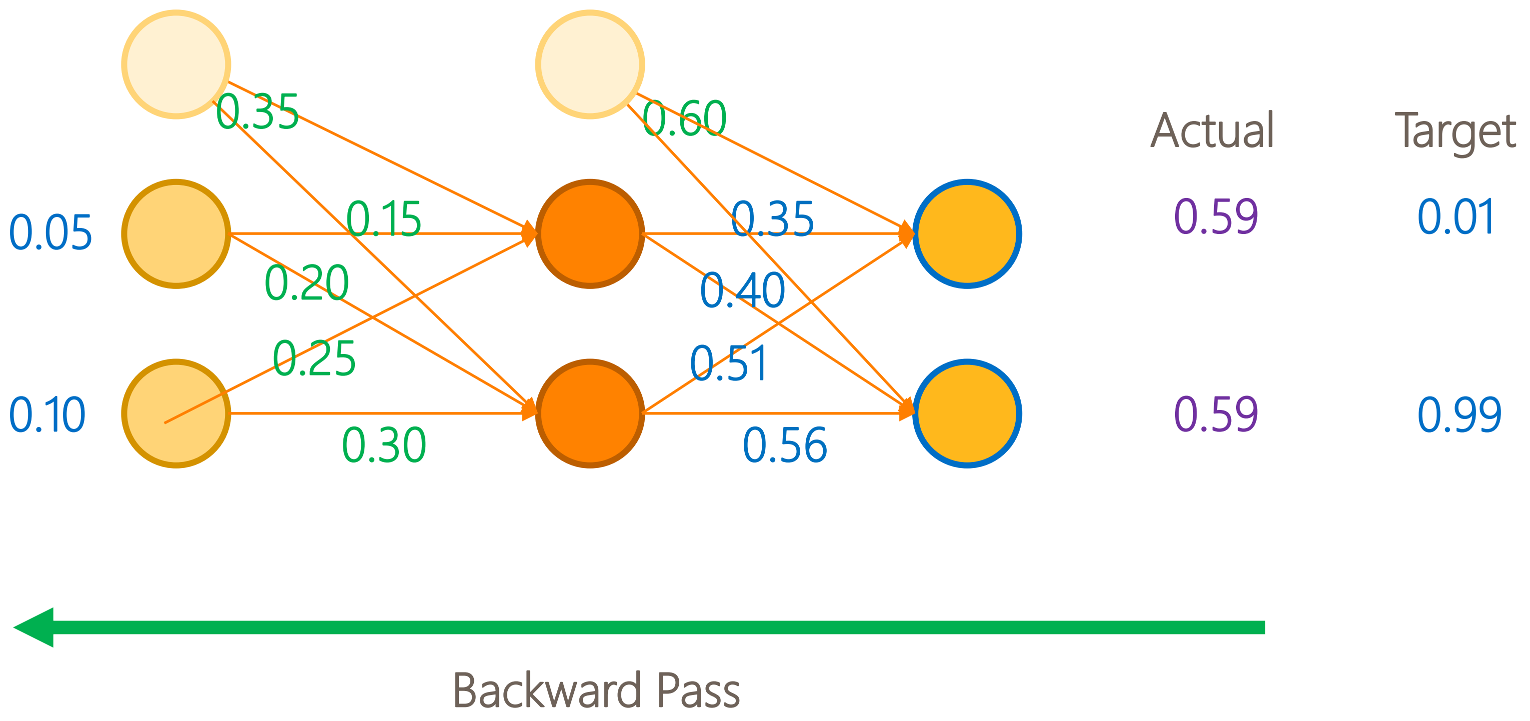


Actual	Target
0.59	0.01
0.59	0.99

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

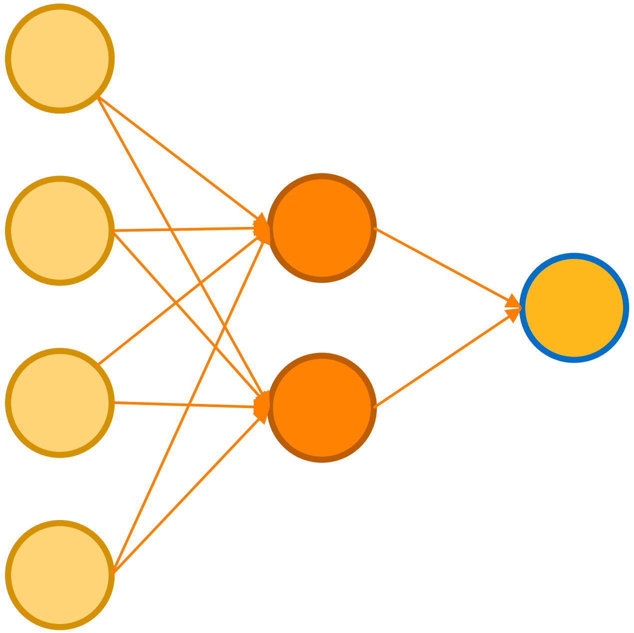
Backward Pass

Backpropagation (6/6)



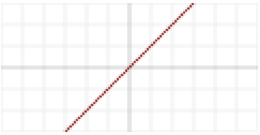
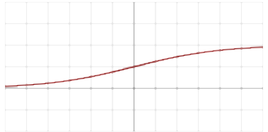
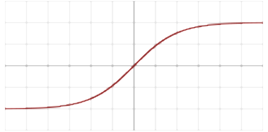
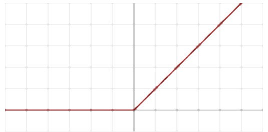
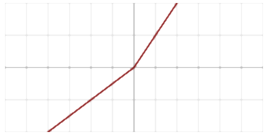
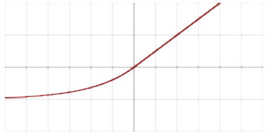
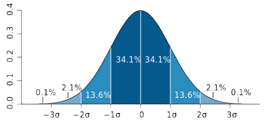
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

Activation Functions



$$h_\theta(X) = \frac{1}{1 + e^{-(\theta^T X + b)}}$$

Activation Function

Identity		$f(x) = x$
Logistic (sigmoid)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ReLU		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
ELU		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Softmax		$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$

REPSOL

Stochastic Gradient Descent

Optimizer

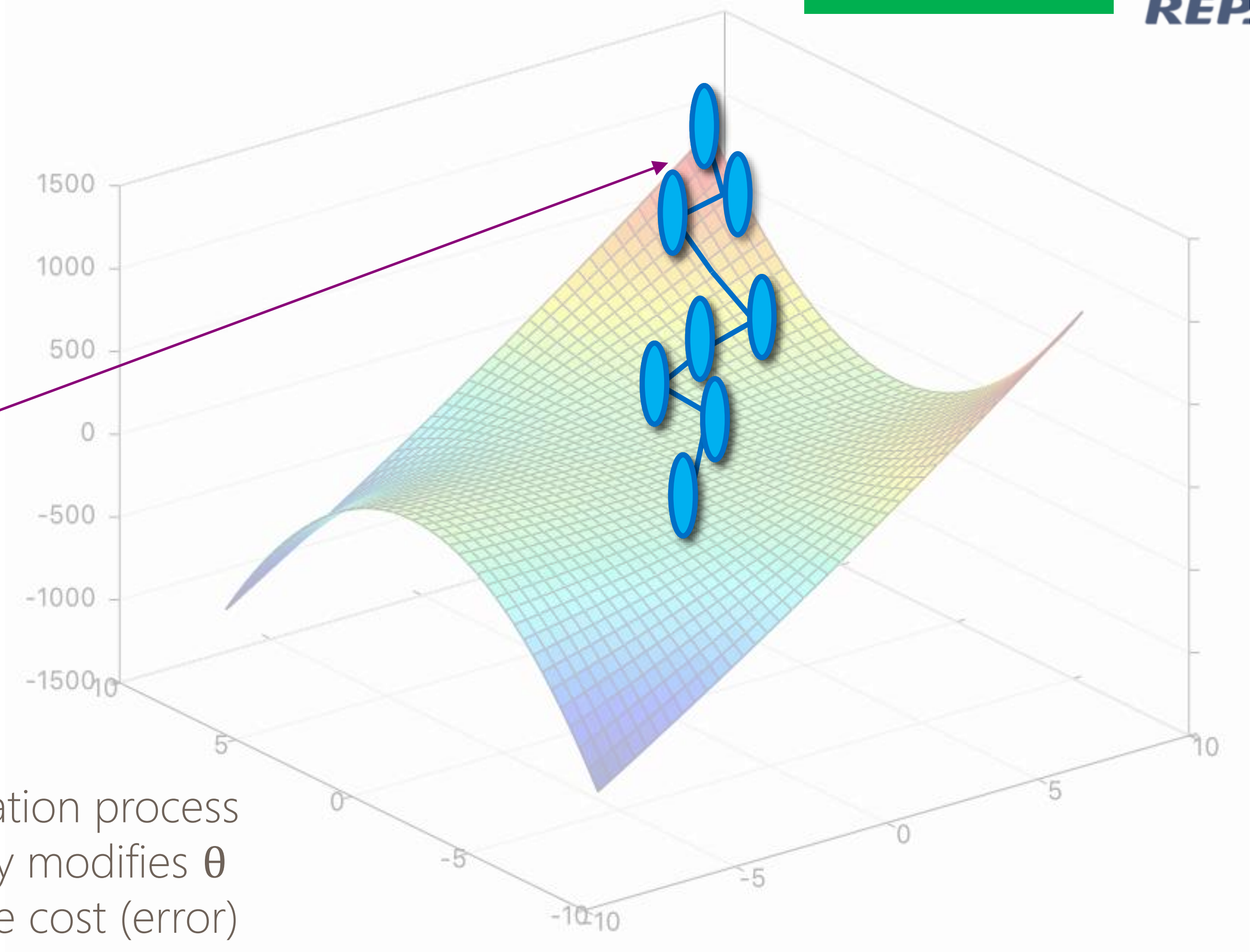
REPSOL

Model $h_\theta(x)$

When model is wrong, the “cost” of that error can be measured as:

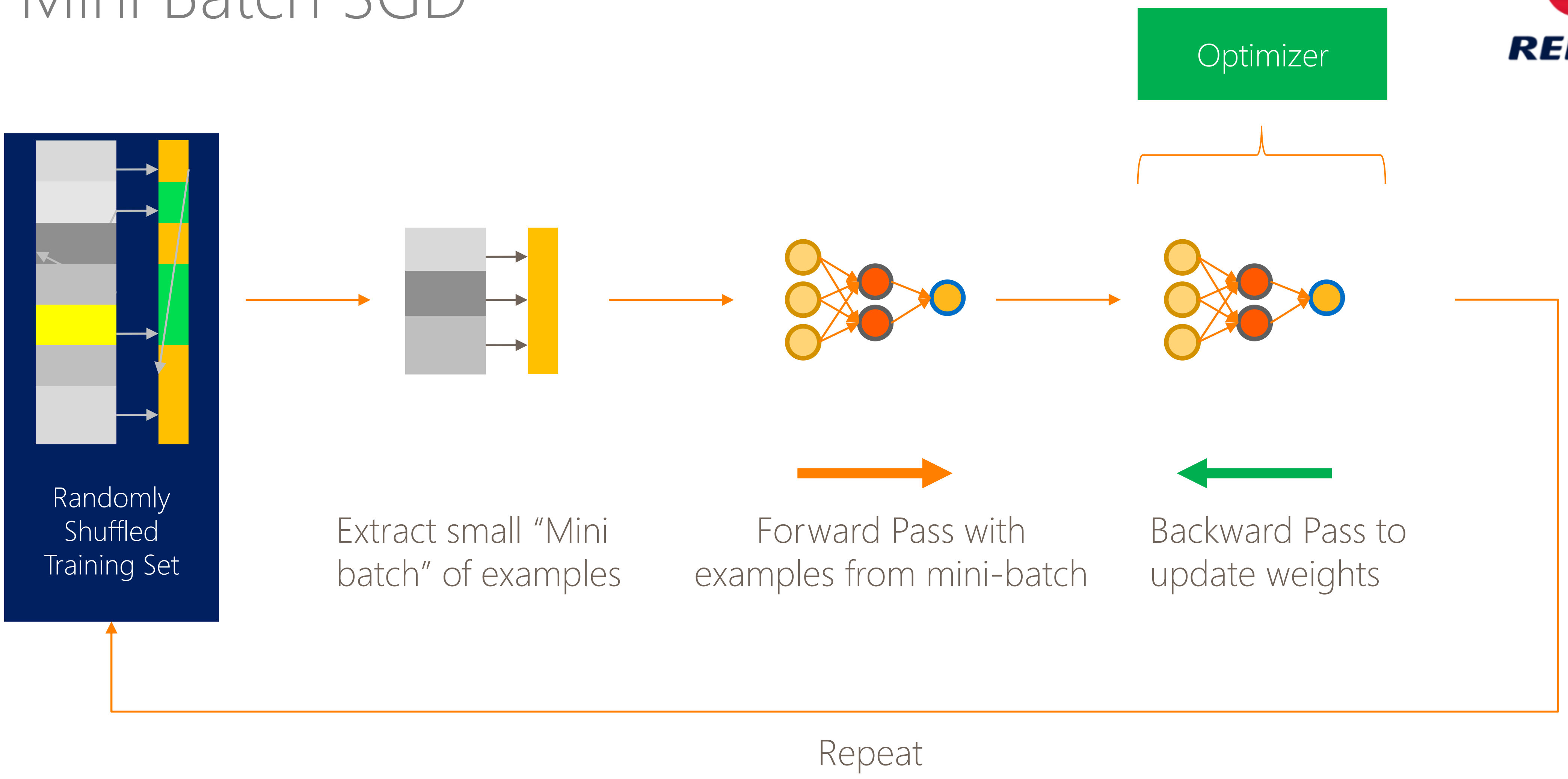
$J(\theta)$

Cost Function



Optimization process iteratively modifies θ to reduce cost (error)

Mini Batch SGD



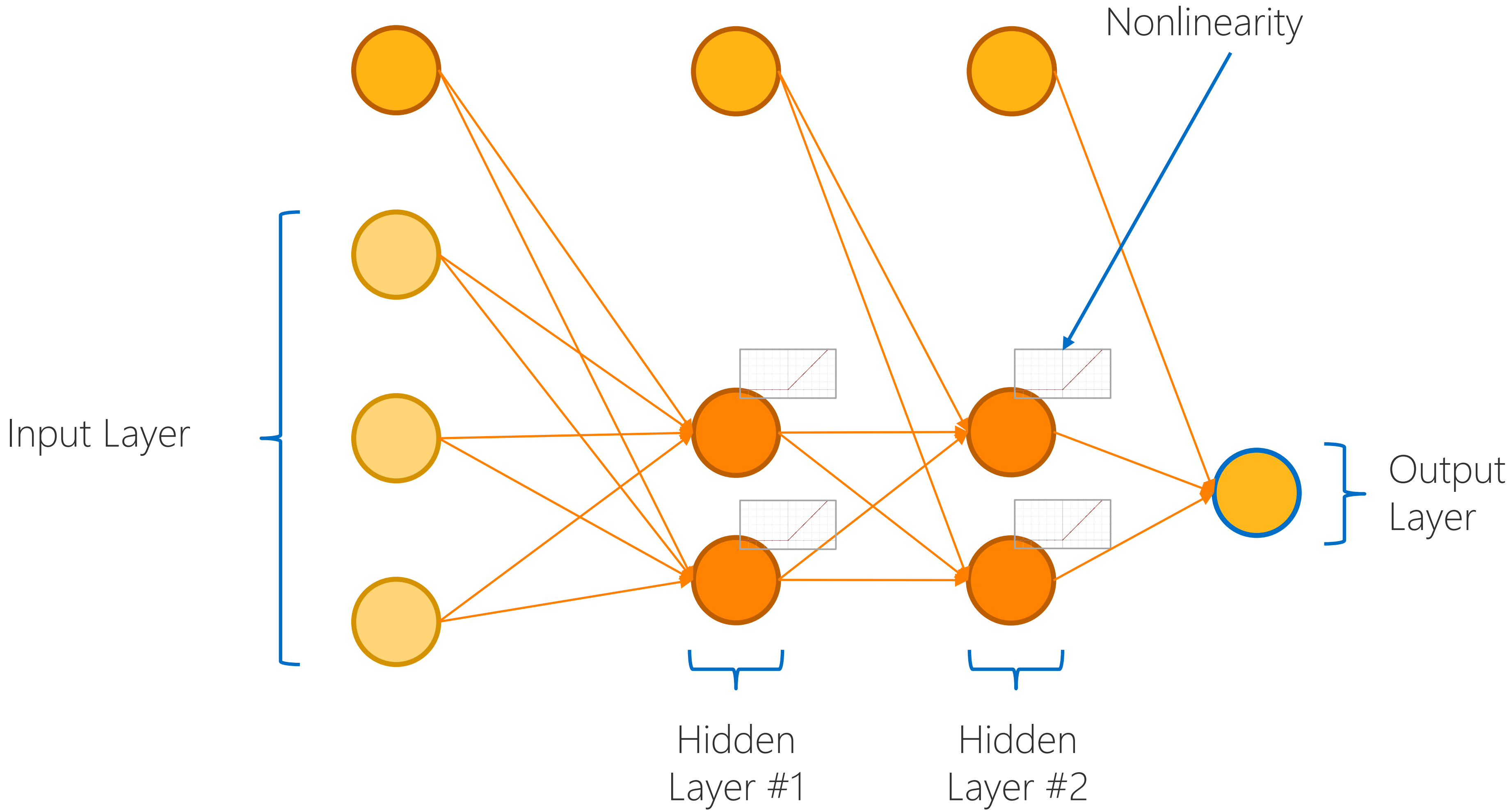
Optimization

Some common optimization algorithms:

- Stochastic Gradient Descent
- Adam (ADAptive Moment estimation)
  - *"The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients"*
- Adagrad (ADAptive GRADient algorithm)
  - " maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems)."
- RMS Prop (Root Mean Square PROPagation)
  - "maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing)."

Neural Networks Architectures

Neural Network "Architecture"



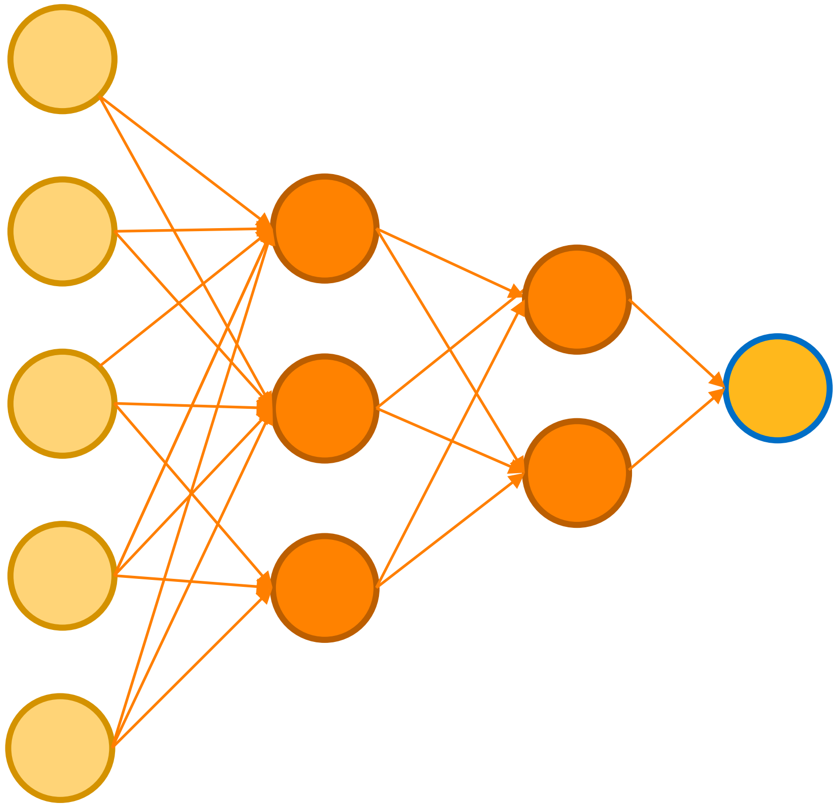
©

Universal Approximation Theorem

A feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate **continuous functions** on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.

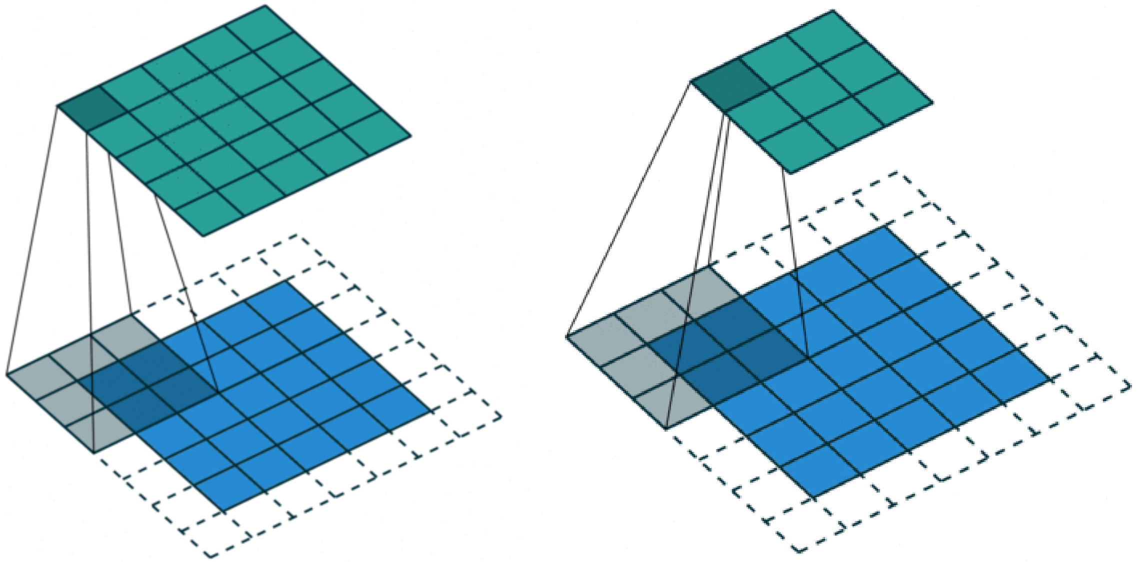
[Universal Approximation Theorem]

Multilayer **Perceptrons**

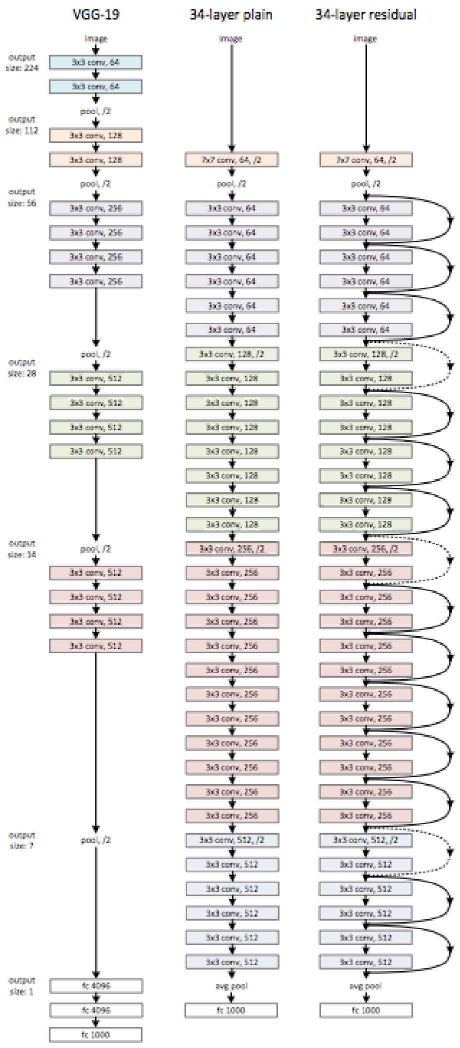


- ***Fully connected*** or ***dense*** layers
- **Limited utility – why?**
  - Big! Computationally expensive.
  - Extremely sensitive to shifts in input – consider image recognition.

Convolutional Networks



- [A Guide to Receptive Field Arithmetic for Convolutional Neural Networks]



- Shift/space invariant
- Effective for computer vision

Sparse vs. Distributed Representations (1/3)

Say you have 5 different classes, represented by vectors:

$$A = [\,1\;0\;0\;0\;0\,]$$
$$B = [\,0\;0\;0\;1\;0\,]$$
$$C = [\,0\;0\;1\;0\;0\,]$$
$$D = [\,0\;0\;0\;0\;1\,]$$
$$E = [\,0\;1\;0\;0\;0\,]$$

Sparse representation

What if you wanted to represent a new class, F ?

Sparse vs. Distributed Representations (2/3)

What if you wanted to represent words?

$$W('cat') = [\,1\;0\;0\;0\;0\,]$$
$$W('dog') = [\,0\;0\;0\;1\;0\,]$$
$$W('bat') = [\,0\;0\;1\;0\;0\,]$$

Feels like we're going to run out of space pretty quickly...

Sparse vs. Distributed Representations (3/3)

Still 5 elements per vector, but far more classes of data can be represented.

$$A = [\, 0.2, 0.7, 0.1, 0.3, -0.5 \,]$$
$$B = [\, -0.3, 0.2, 0.1, 0.9, -0.7 \,]$$
$$C = [\, 0.5, 0.1, -0.6, -0.2, 0.8 \,]$$
$$D = [\, -0.7, -0.3, -0.4, 0.2, 0.1 \,]$$
$$E = [\, 0.8, 0.2, 0.3, -0.4, 0.3 \,]$$
$$F = [\, 0.9, 0.2, 0.7, -0.3, -0.4 \,]$$

Dense representation

Now - what if you wanted to represent a new class, *F* ?

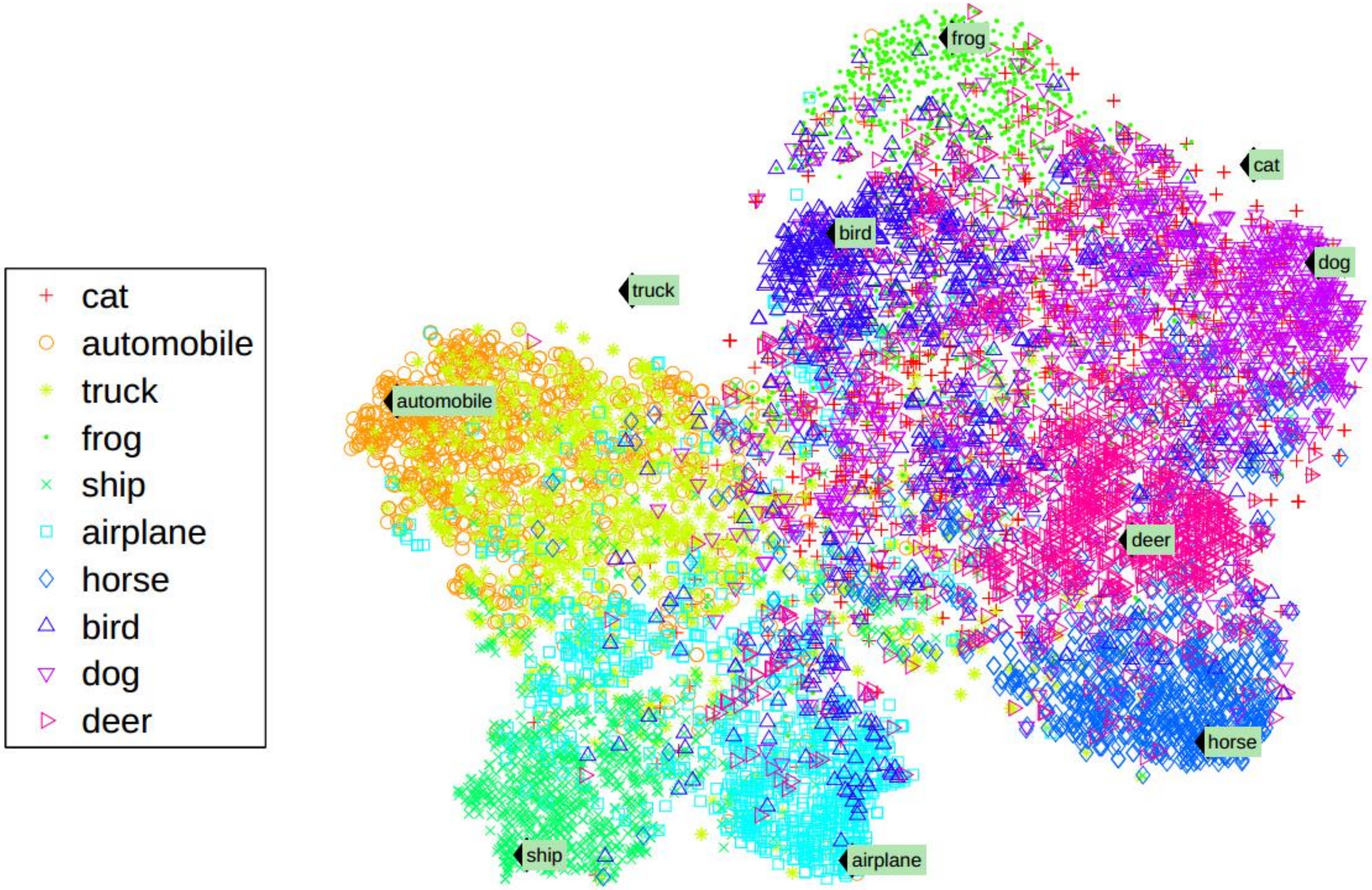
Word Embeddings (1/3)

What if you wanted to represent words?

$$W('cat') = [\,0.2, 0.7, 0.1, 0.3, -0.5, \ldots \quad]$$
$$W('dog') = [\,-0.3, 0.2, 0.1, 0.9, -0.7, \ldots \quad]$$
$$W('bat') = [\,0.5, 0.1, -0.6, -0.2, 0.8, \ldots \quad]$$

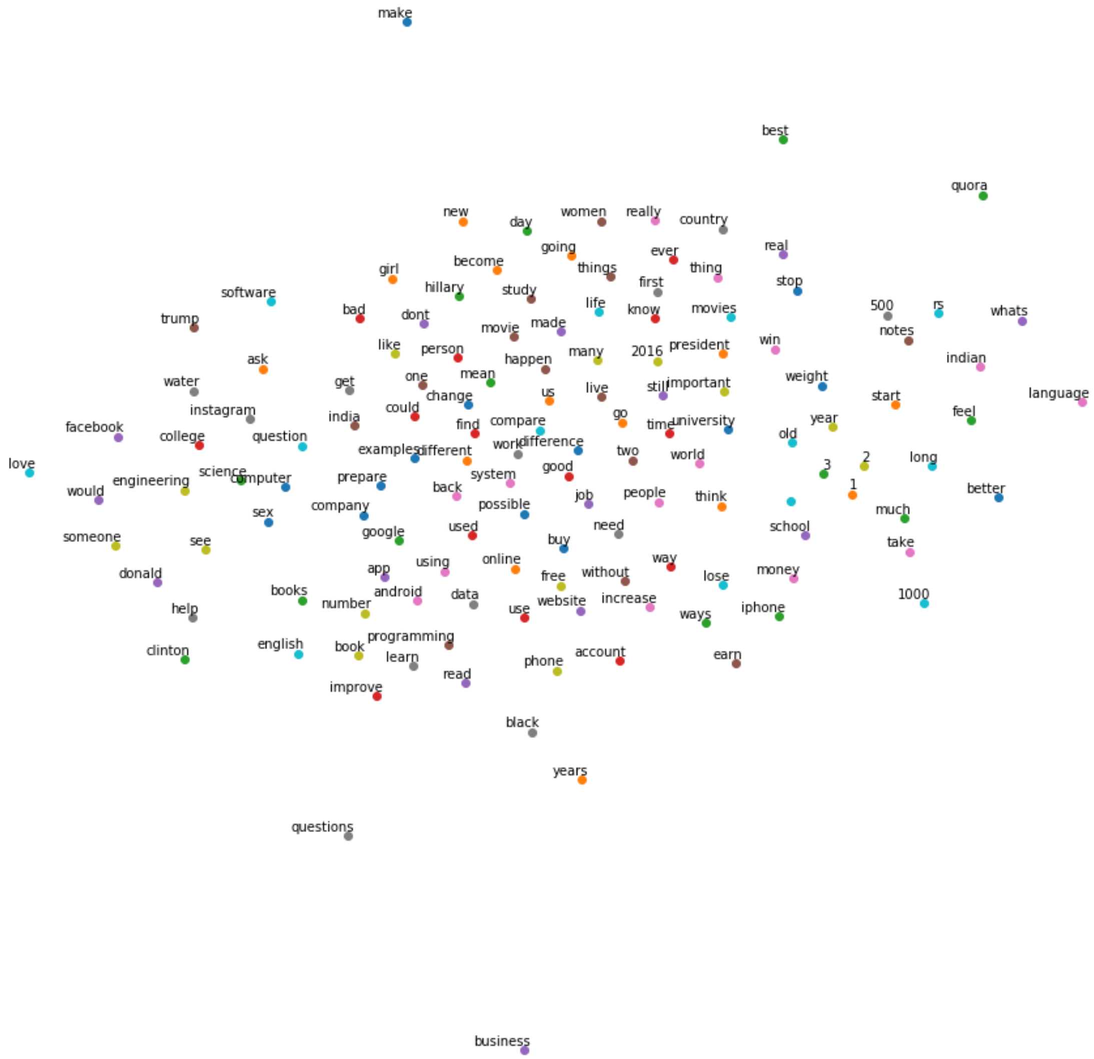
Significantly more room for growth!

Word Embeddings (2/3)



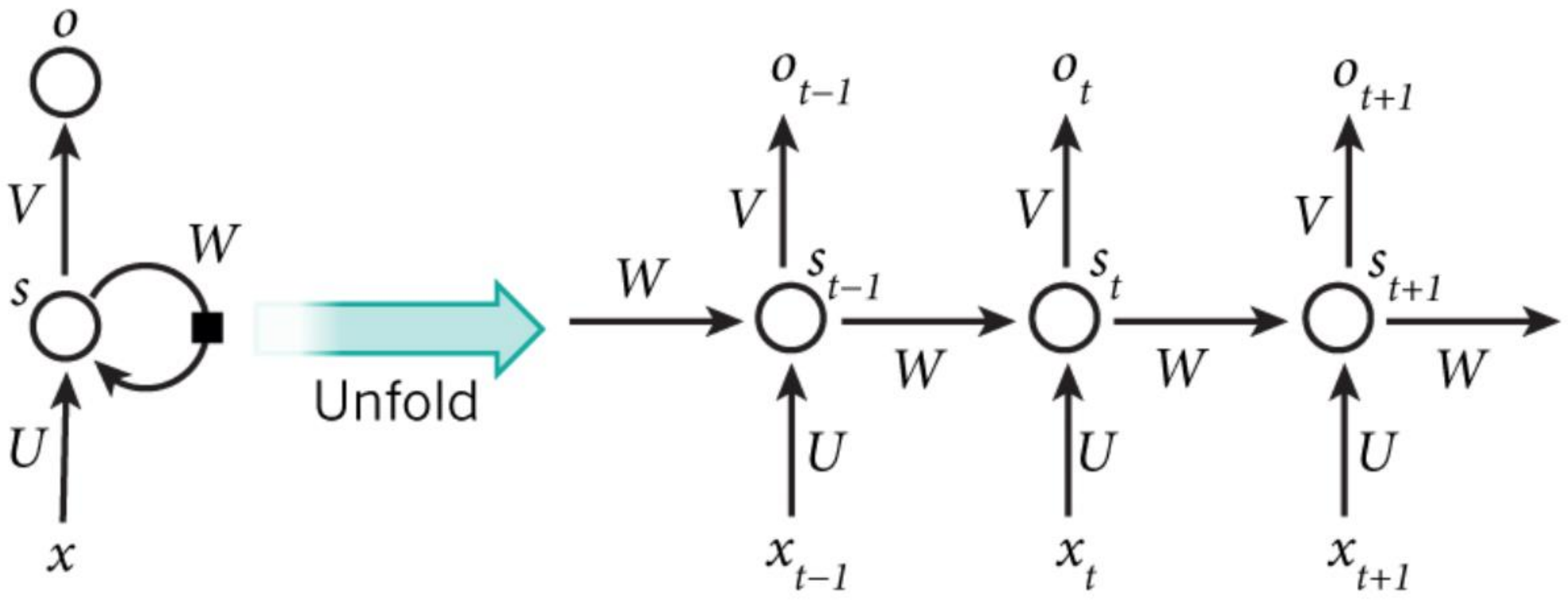
[Deep Learning, NLP, and Representations]

Word Embeddings (3/3)

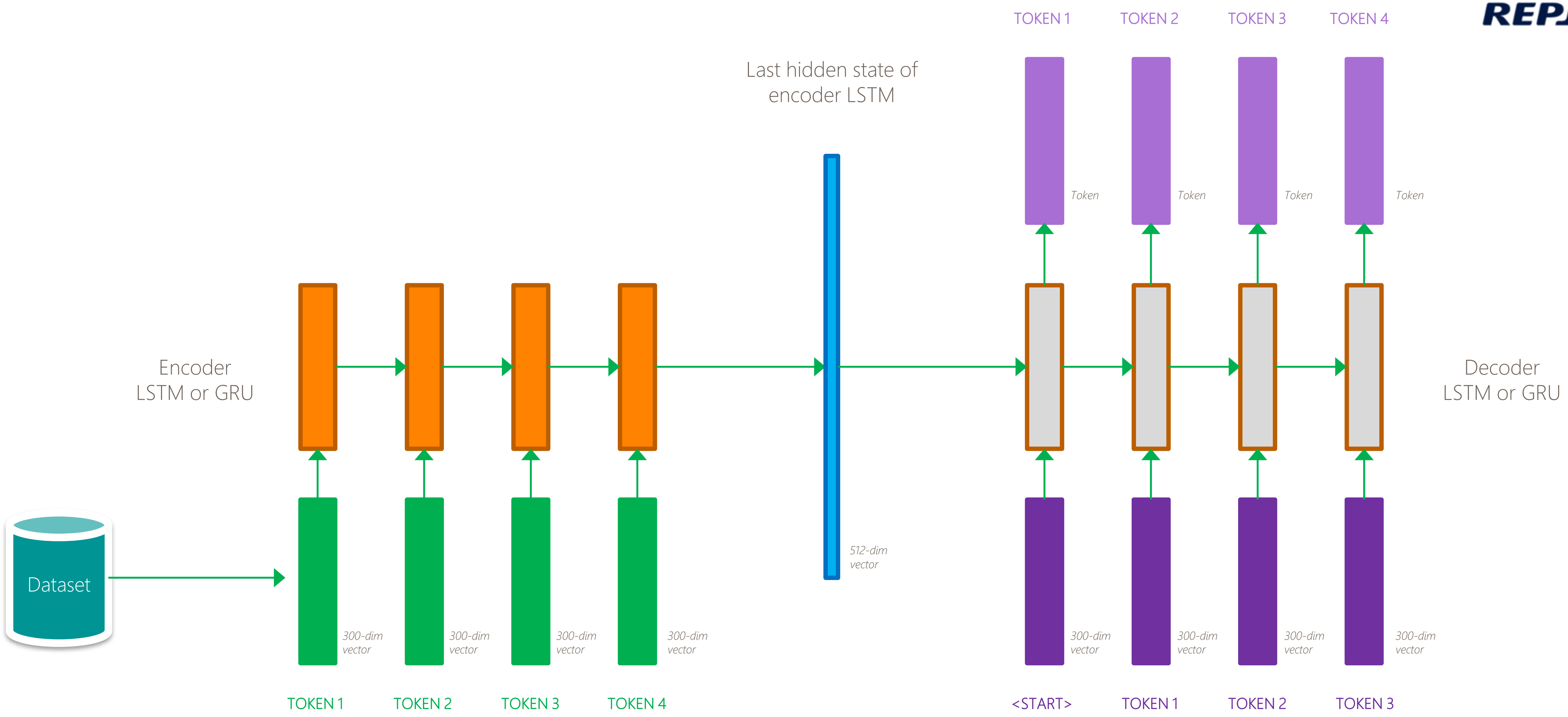


[Visualizing Word Vectors with t-SNE]()

Recurrent Networks



RNN Example: seq2seq Models



Generative Models

Zhang et al. (2016).

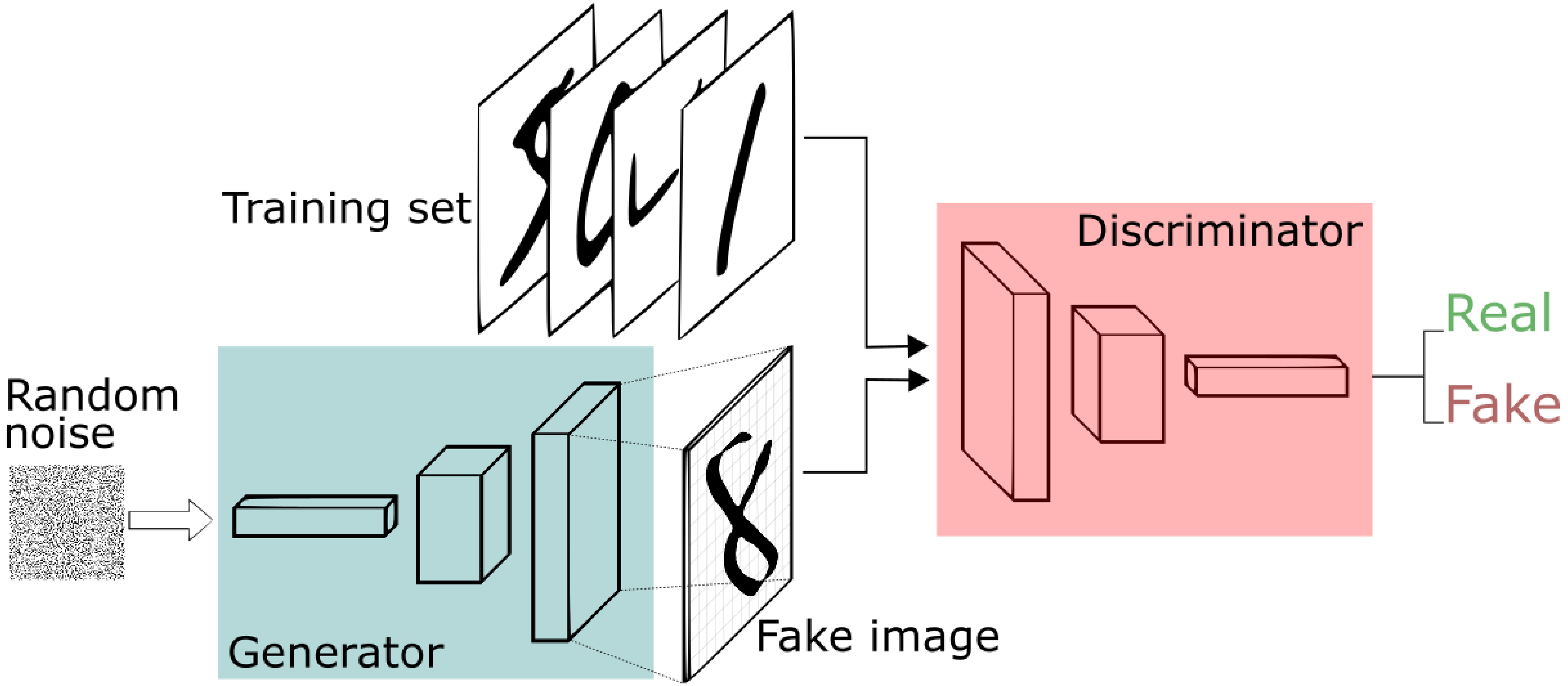
arXiv:1612.03242

https://github.com/hanzhanggit/StackGAN

Generative Models: Generative Adversarial Networks

- Two neural networks, pitted against each other in a zero-sum game framework
  - One network generates data, the other evaluates it for “realness”
- Used in unsupervised learning
- Learn to mimic arbitrary distributions of data
- Can generate superficially photorealistic images
- Can be very difficult to train

Generative Adversarial Networks

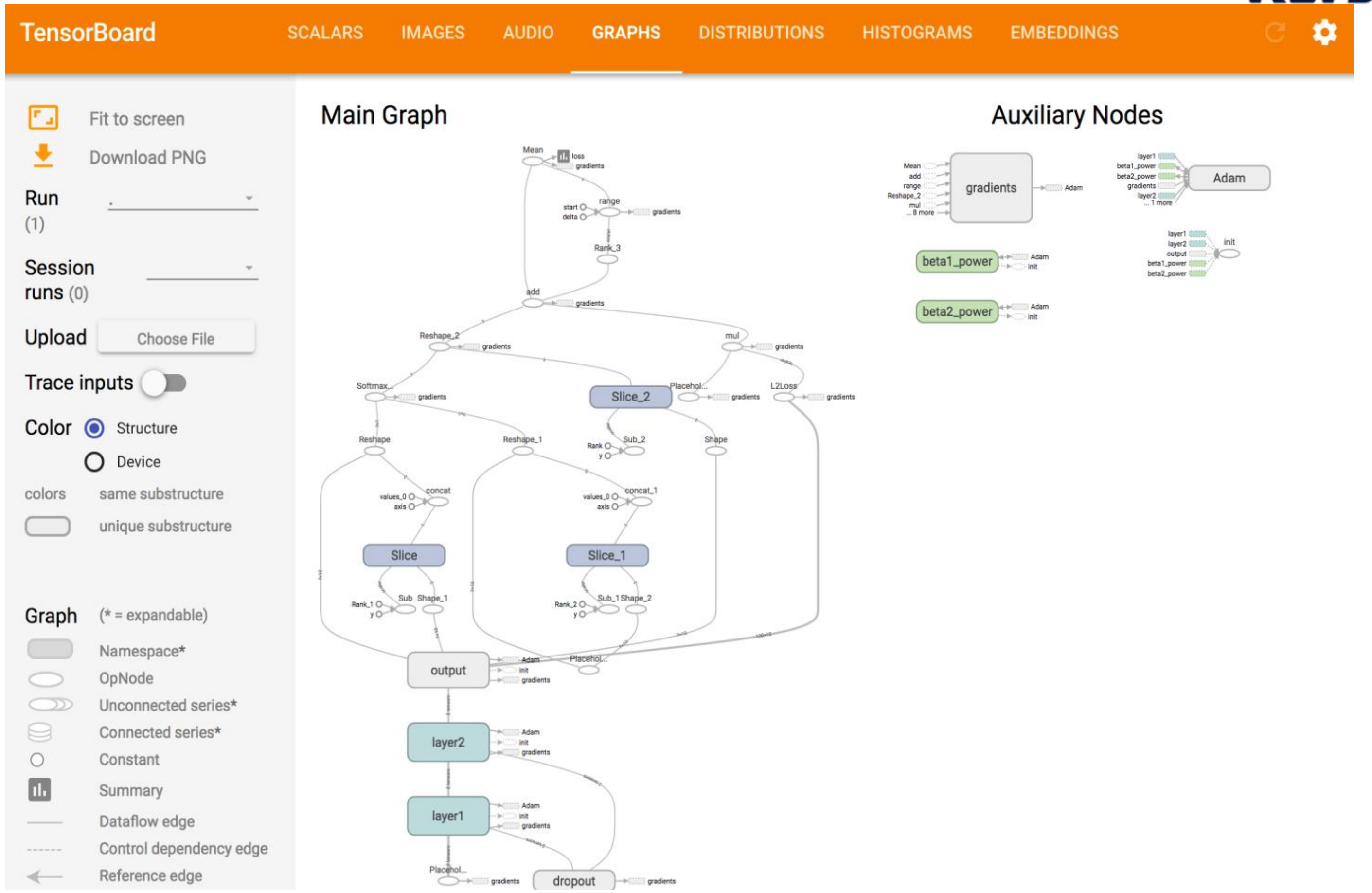


[GAN: A Beginner's Guide to Generative Adversarial Networks]

Deep Learning Frameworks

Deep Learning Frameworks (1/2)

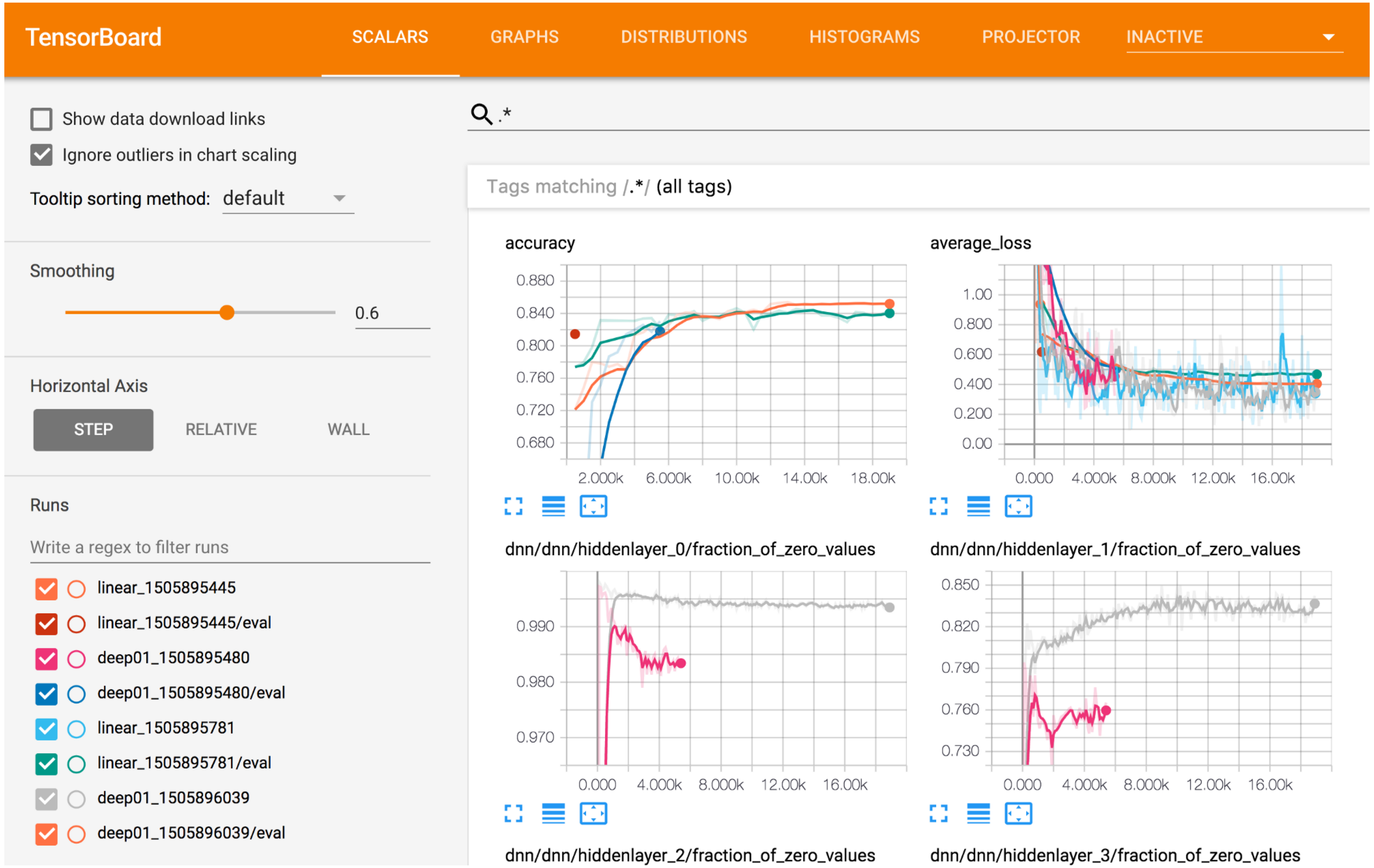
- Define models via differentiable computational graphs
- Learn models via minibatch stochastic gradient descent
- Provide pre-built components for models (e.g., LSTM cell)
- Leverage pre-built models (e.g. transfer learning)
- Model / learning visualization
- Model export for inference



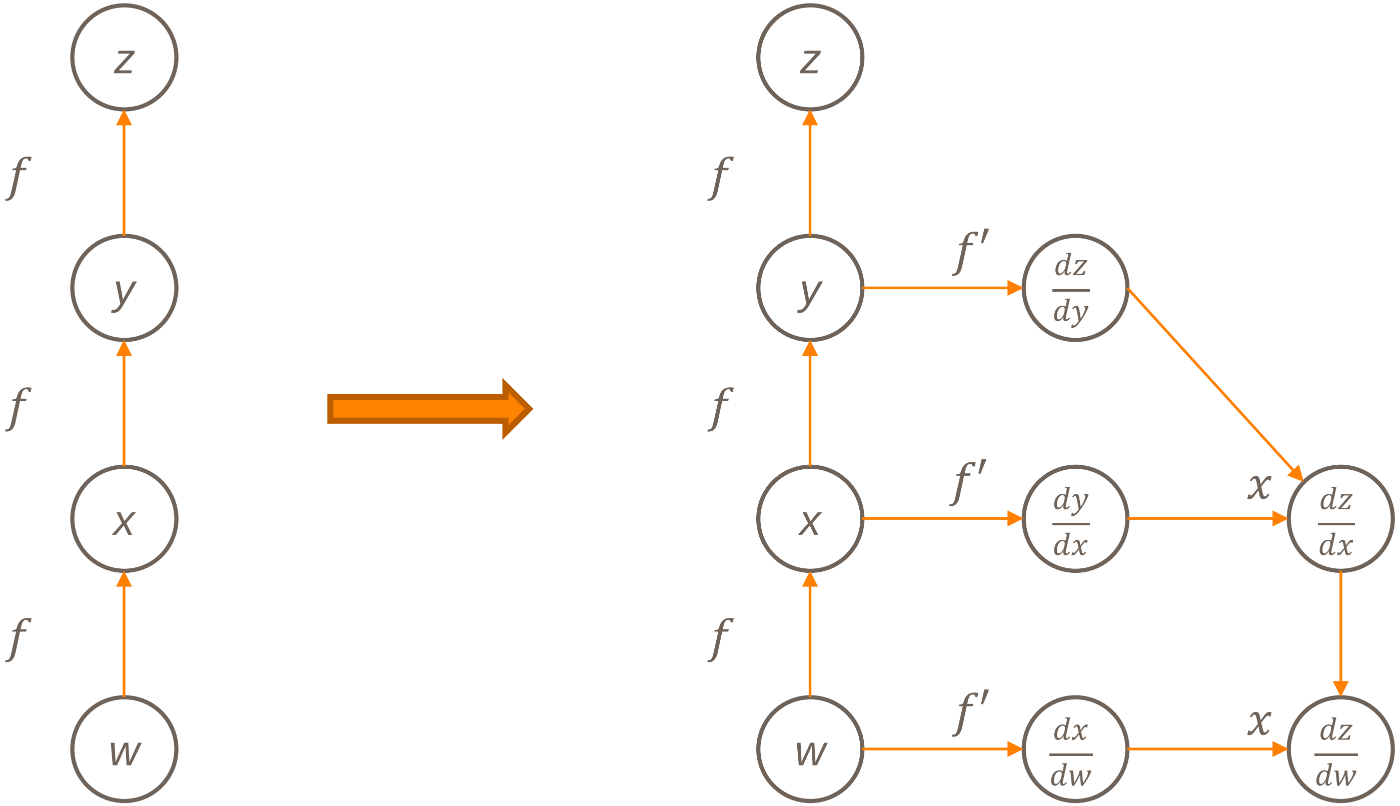
Deep Learning Frameworks (2/2)

A few...

- TensorFlow
- Keras
- Microsoft Cognitive Toolkit (CNTK)
- PyTorch
- Caffe2



Differentiable Graphs



$$\frac{dz}{dw} = \frac{dz}{dy}\frac{dy}{dx}\frac{dx}{dw}$$

Tensors (1/5)

- Generalization of vectors and matrices
- Not quite the same as tensors in physics!
- TensorFlow tensors have:
  - A **data type** (float32, int32, string, etc.)
  - A **shape**

Rank	Math Entity
0	Scalar
1	Vector
2	Matrix (table of numbers
3	3-Tensor (cube of numbers)
n	n-Tensor

Tensors (2/5)

Tensor Shape	Example
[784]	Single 28x28 grayscale image, flattened

Tensors (3/5)

Tensor Shape	Example
[784]	Single 28x28 grayscale image, flattened
[3, 784]	Single 28x28 image, three color channels

Tensors (4/5)

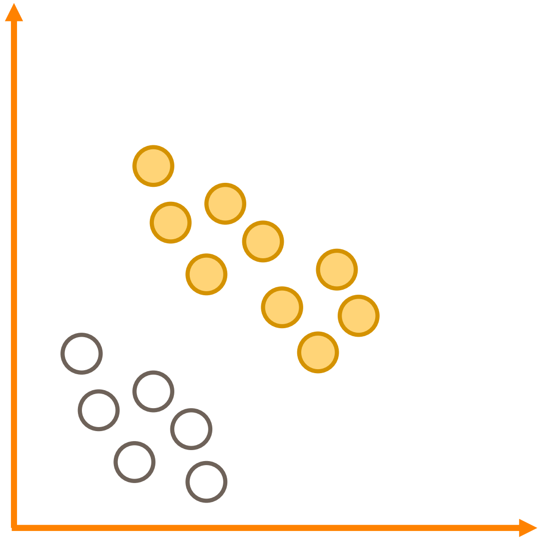
Tensor Shape	Example
[784]	Single 28x28 grayscale image, flattened
[3, 784]	Single 28x28 image, three color channels
[10, 3, 784]	Ten 28x28 images, each with three color channels

Tensors (5/5)

Tensor Shape	Example
[784]	Single 28x28 grayscale image, flattened
[3, 784]	Single 28x28 image, three color channels
[10, 3, 784]	Ten 28x28 images, each with three color channels
[10, 100, 3, 784]	Ten videos, each with 100 frames, each frame with three color channels

Logistic Regression

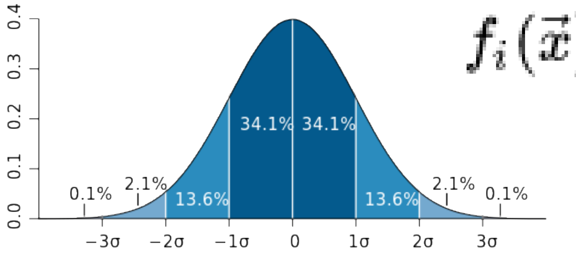
- Regression model where dependent variable is categorical
- Looking for linear decision boundary to separate classes



$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

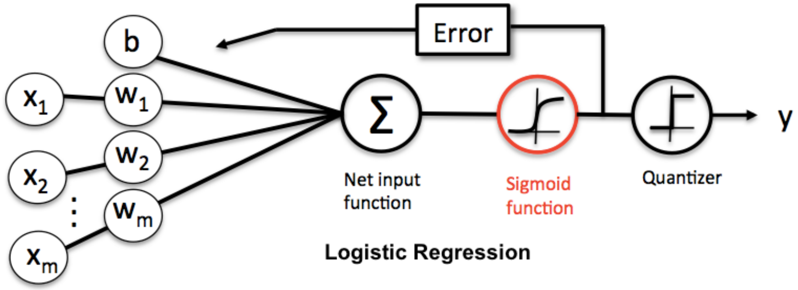
Softmax Regression

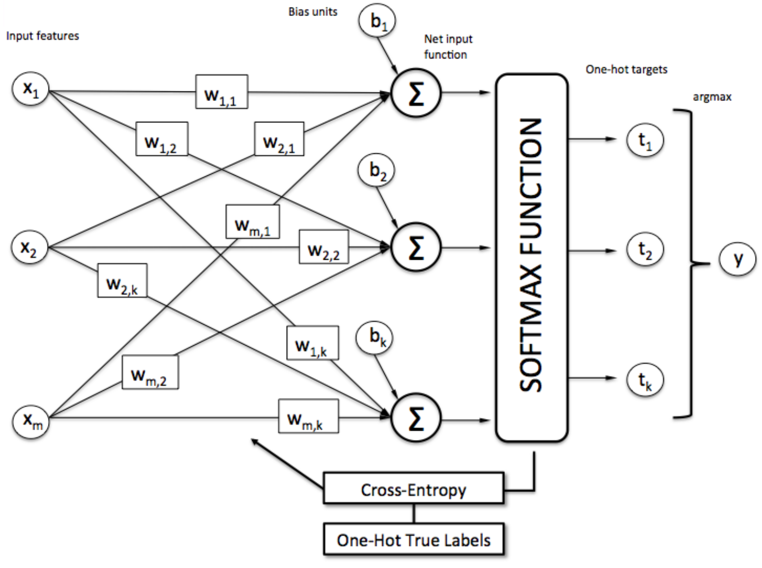
- Generalization of logistic regression that can be used for multi-class classification
- Replace logistic (sigmoid) function with the softmax function:



$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$$

- Softmax function is a generalization of logistic (sigmoid) function that can output a probability distribution over classes





Remember:

Deep learning is a class of machine learning algorithms that

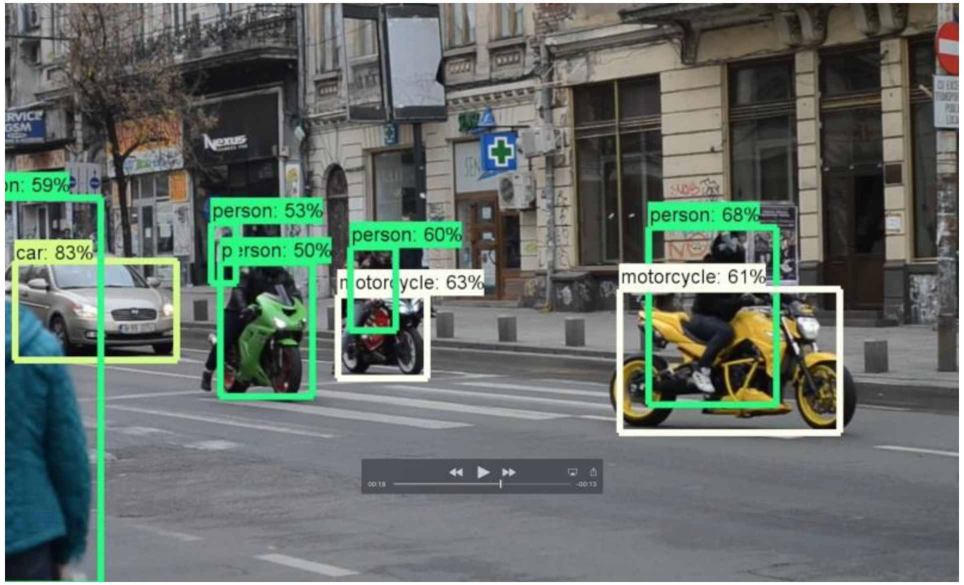
- use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
- learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners.
- learn multiple levels of representations that correspond to different levels of abstraction

Convolutional Neural Networks

Convolutional Neural Networks

Limitations of Machine Learning in computer vision

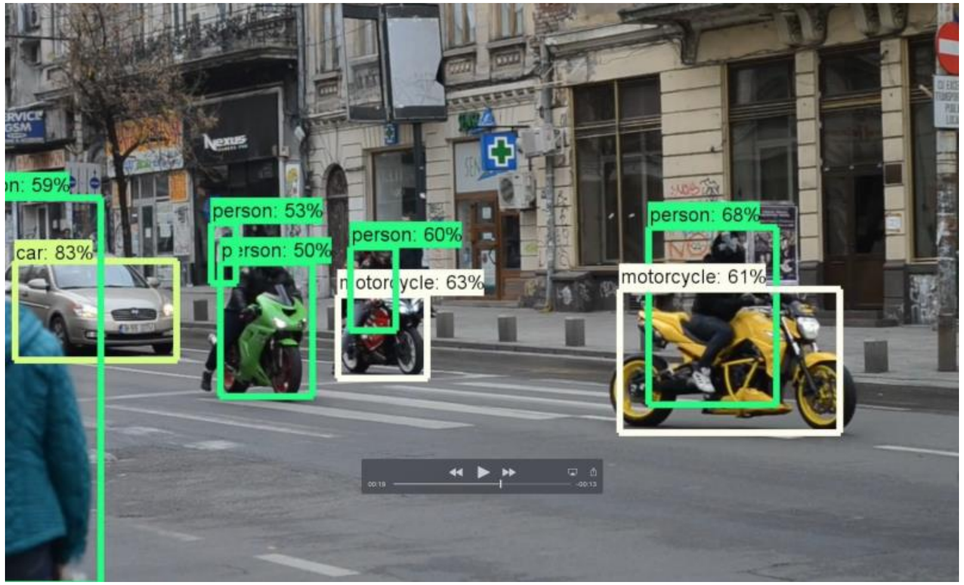
- One of the challenges of traditional machine learning approaches is: Feature Extraction
- For complex problems such as object recognition or handwriting recognition, this is a huge challenge



Deep Learning presents a good alternative

Limitations of Machine Learning in computer vision

- One of the challenges of traditional machine learning approaches is: Feature Extraction
- For complex problems such as object recognition or handwriting recognition, this is a huge challenge

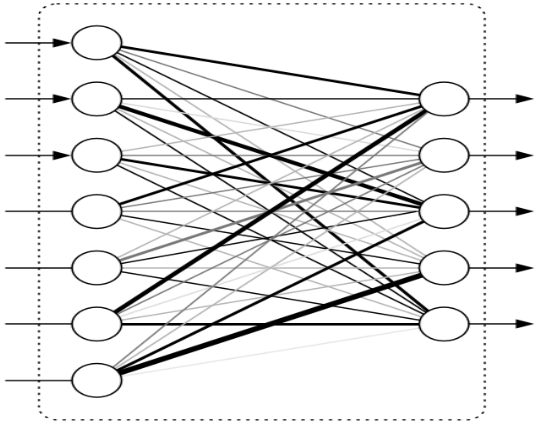


Deep Learning presents a good alternative

Limitations of Fully Connected Networks

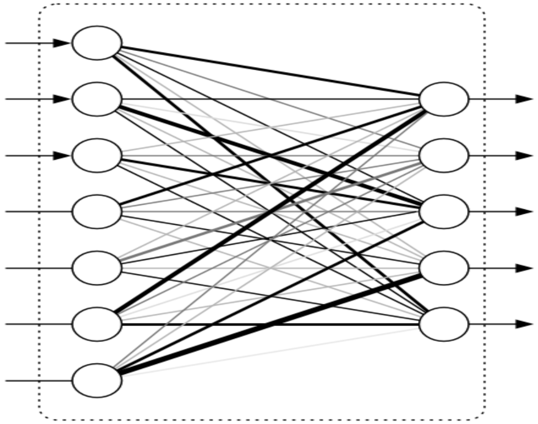
- Huge number of parameters, exponentially increasing with image size
- Higher number of parameters, leads to higher number of neurons, might lead to overfitting

Image with 20x20x3



Weights in the first hidden layer: 1200

Image with 200x200x3

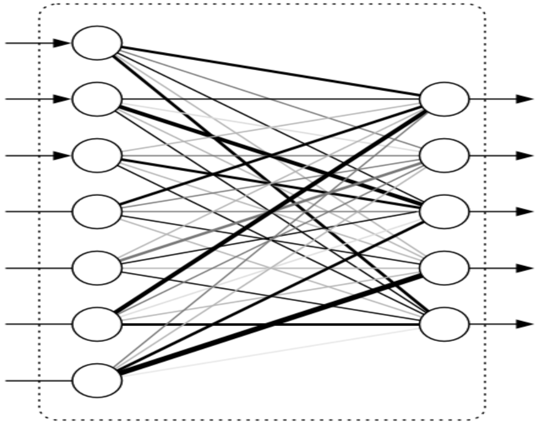


Weights in the first hidden layer: 120000

Limitations of Fully Connected Networks

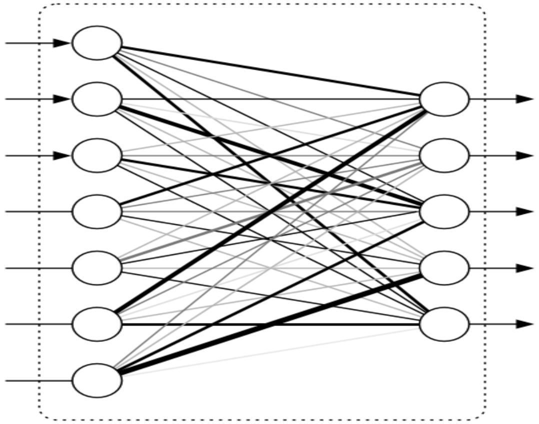
- Huge number of parameters, exponentially increasing with image size
- Higher number of parameters, leads to higher number of neurons, might lead to overfitting

Image with 20x20x3



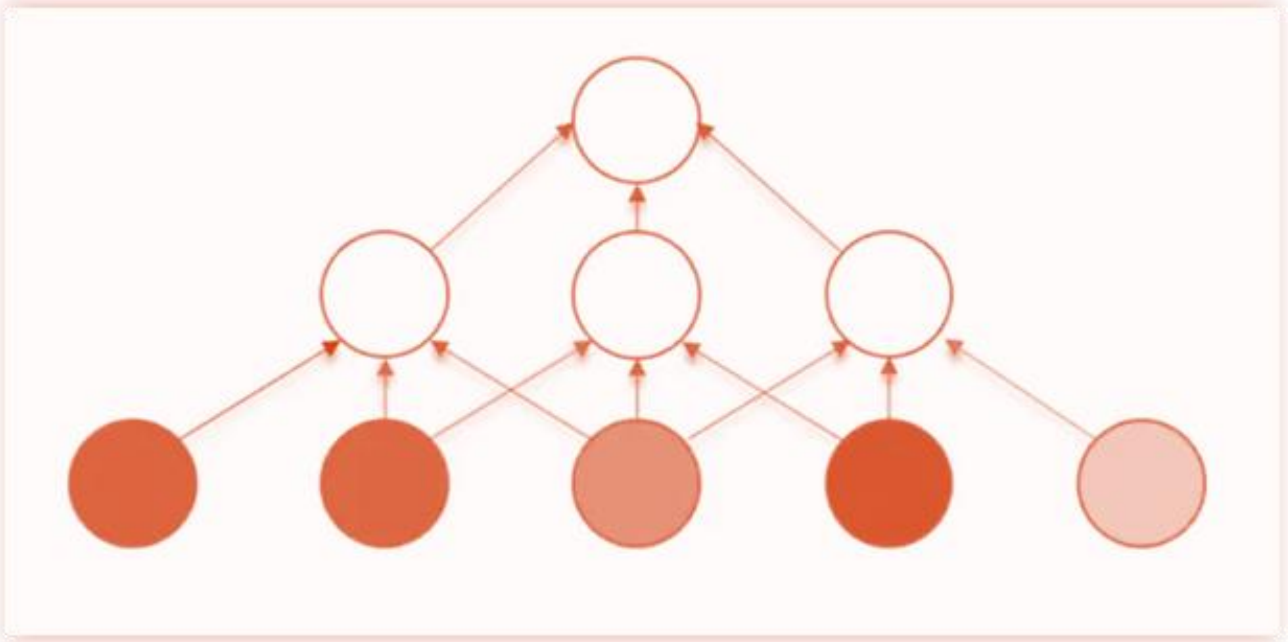
Weights in the first hidden layer: 1200

Image with 200x200x3



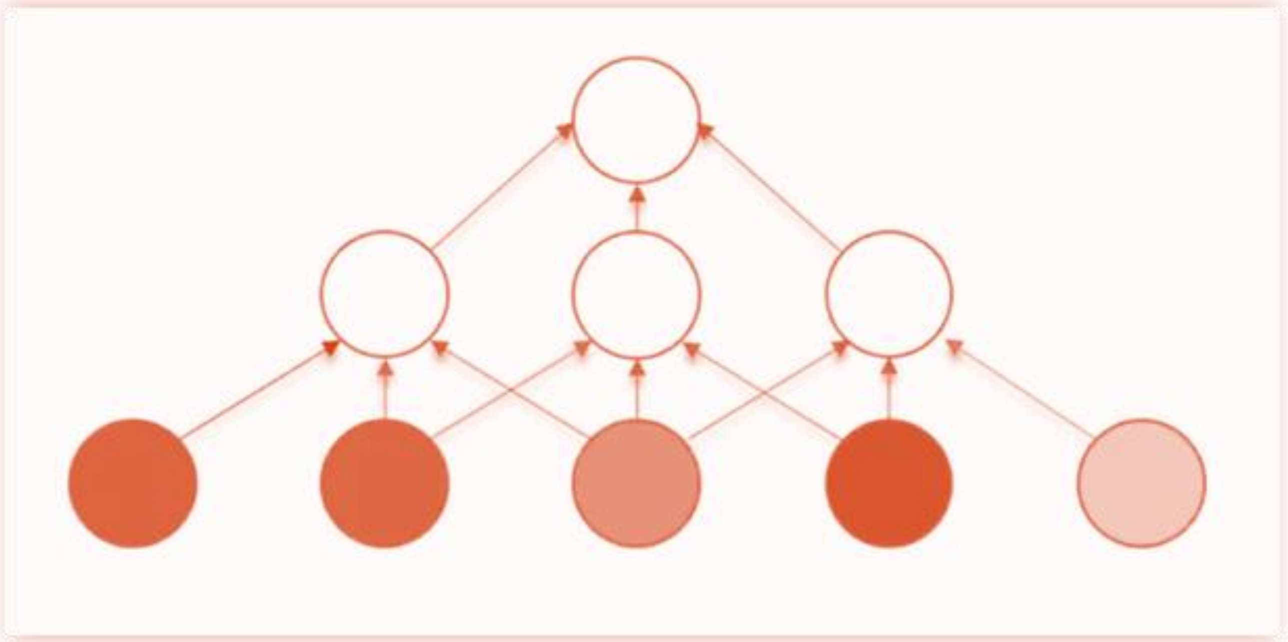
Weights in the first hidden layer: 120000

Convolutional Networks



In CNN, a neuron will only be connected to a small region of neuron before it instead of all neurons in fully connected network

Convolutional Networks



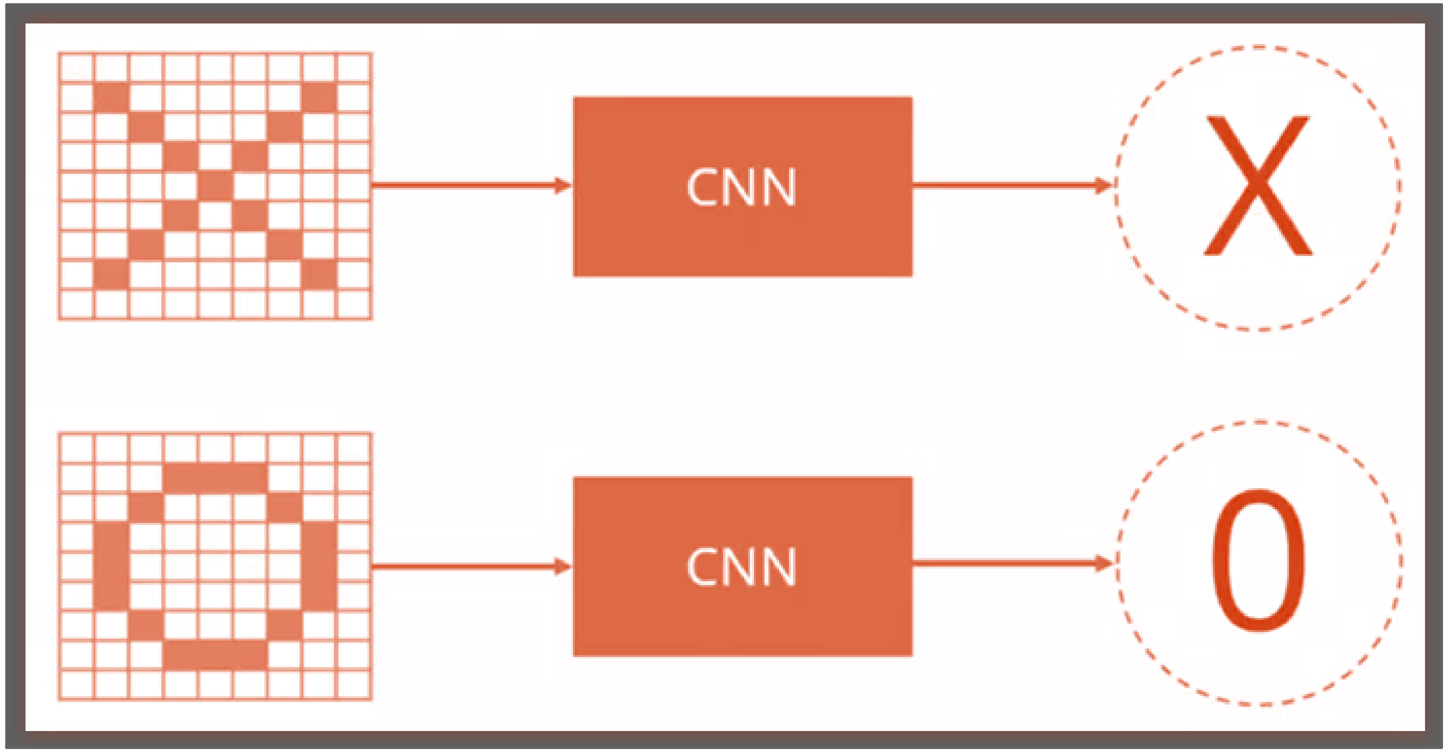
In CNN, a neuron will only be connected to a small region of neuron before it instead of all neurons in fully connected network

Convolutional Networks Architecture (1/2)

CNN has the following layers:

- Convolution Layer
- ReLu Layer
- Pooling Layer
- Fully Connected Layer

Let´s build a CNN for classifying X and O

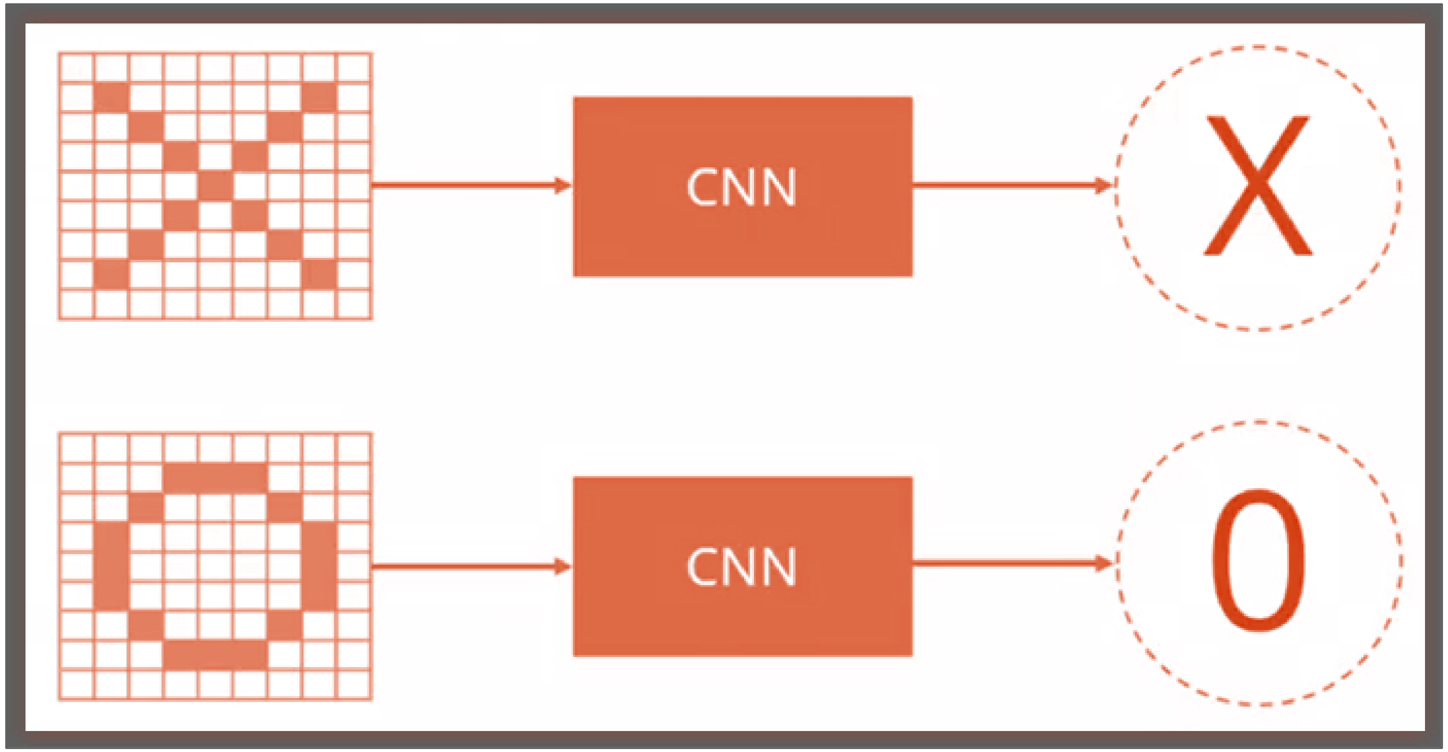


Convolutional Networks Architecture (1/2)

CNN has the following layers:

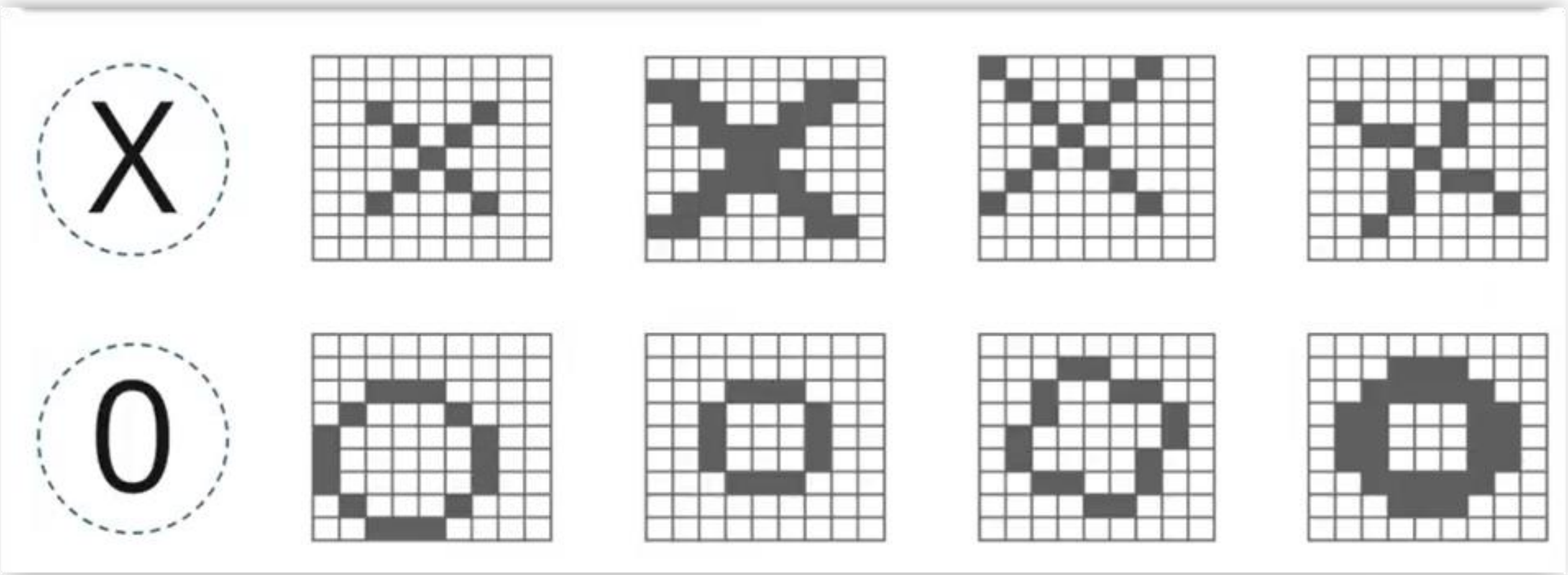
- Convolution Layer
- ReLu Layer
- Pooling Layer
- Fully Connected Layer

Let´s build a CNN for classifying X and O



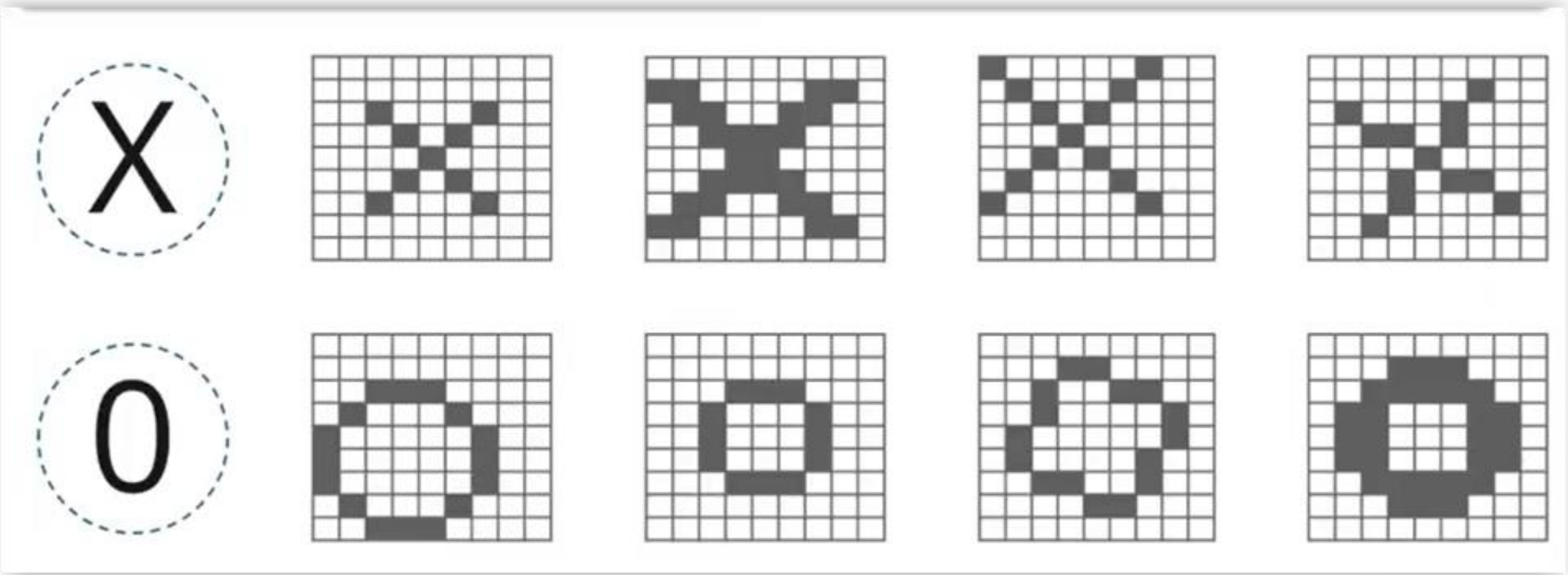
Convolutional Networks Architecture (2/2)

We should be able to classify even tricky cases



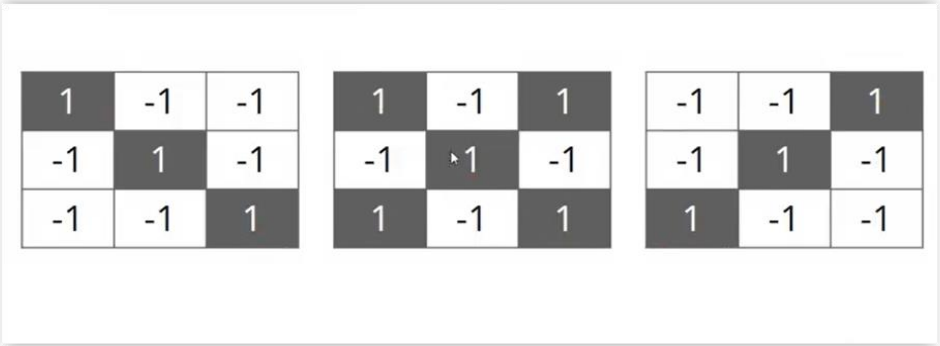
Convolutional Networks Architecture (2/2)

We should be able to classify even tricky cases

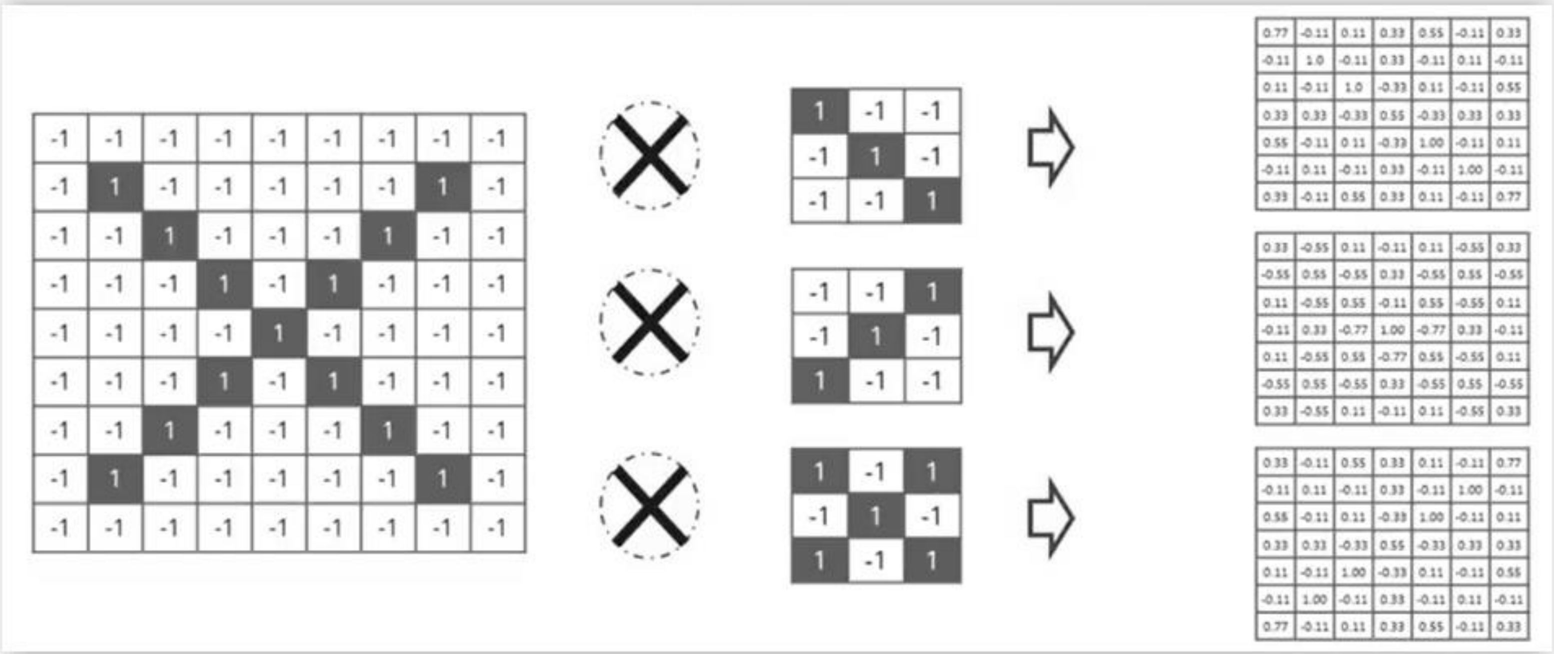


Convolutional Layer

- Performs convolution with predefined filters.
- In our example we chose the following 3 filters

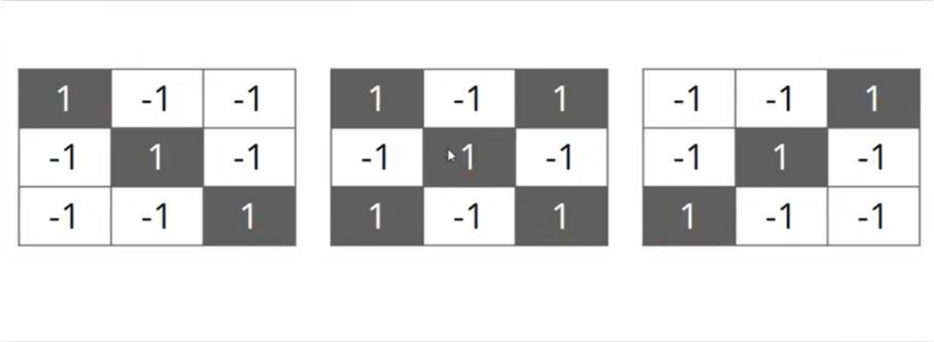


- We perform convolutions on each image region, and update the convolutional layer output

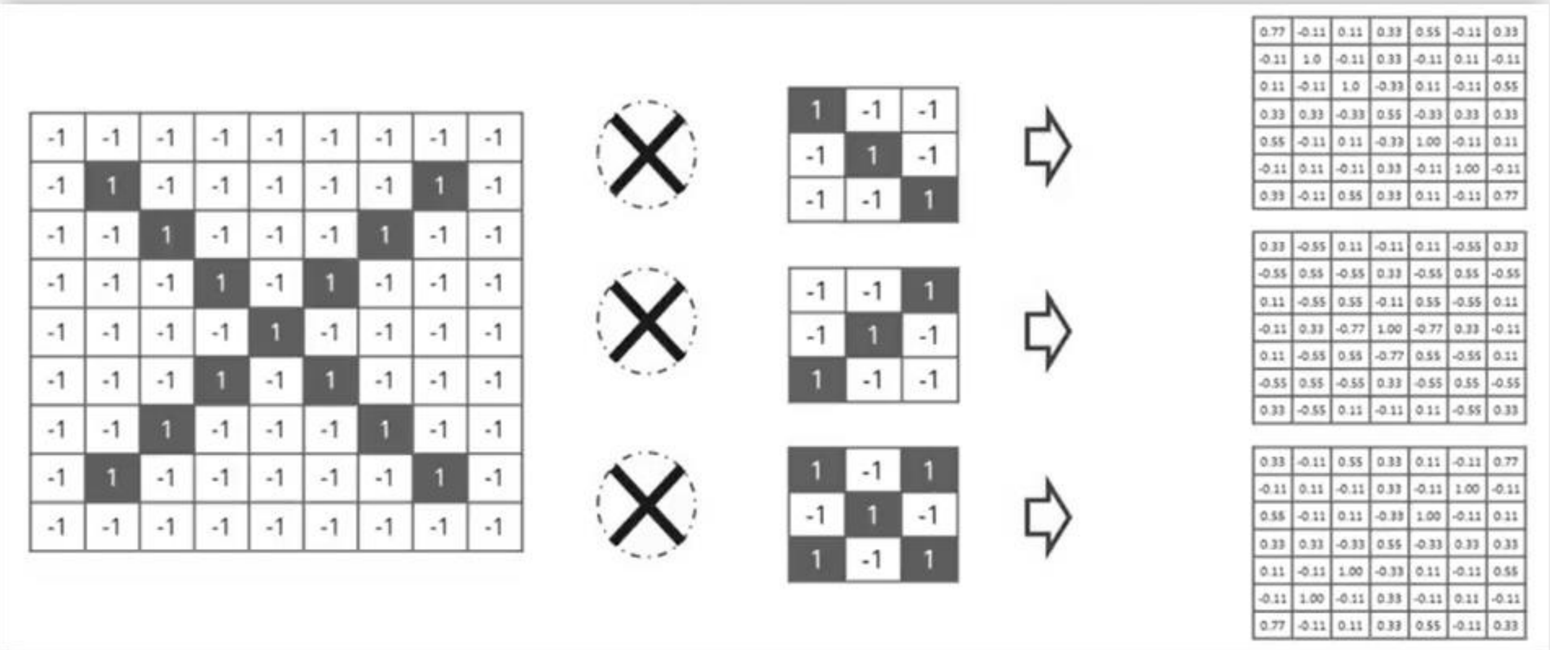


Convolutional Layer

- Performs convolution with predefined filters.
- In our example we chose the following 3 filters

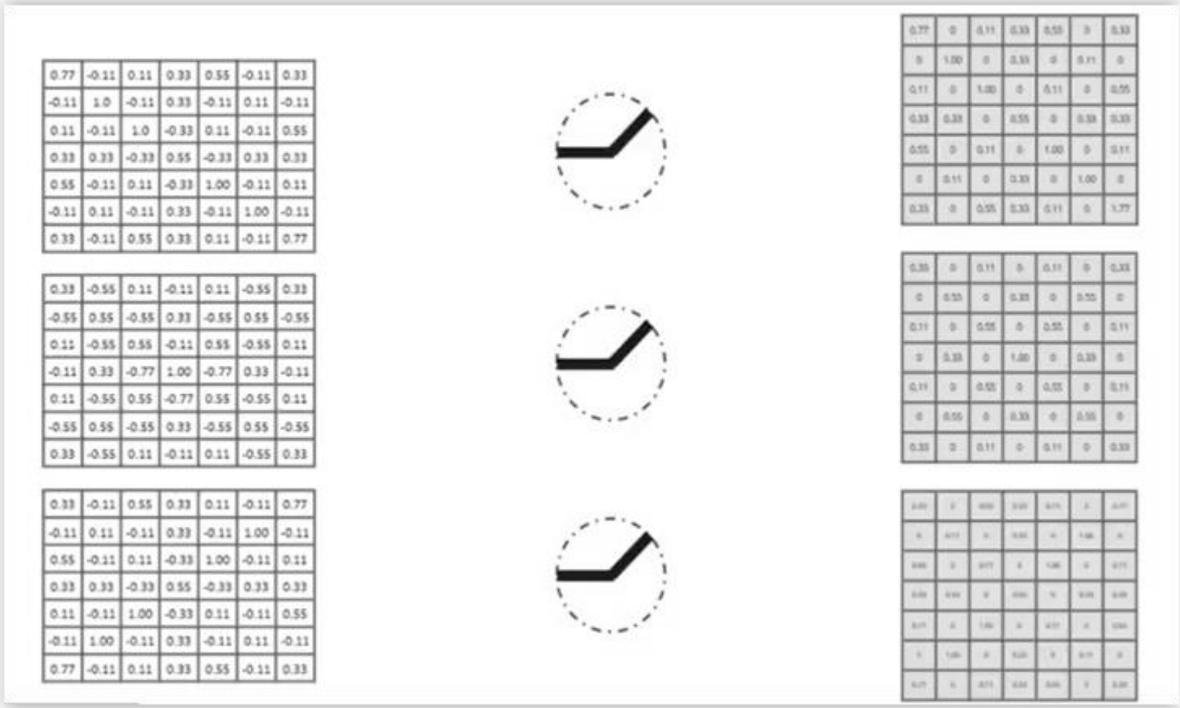


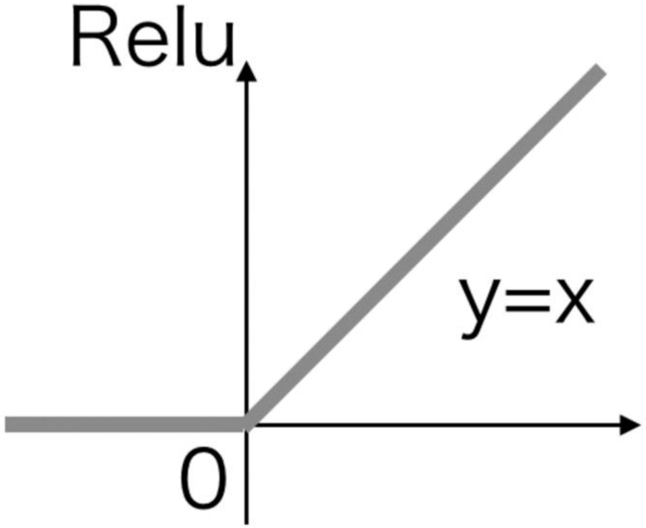
- We perform convolutions on each image region, and update the convolutional layer output



ReLu Layer

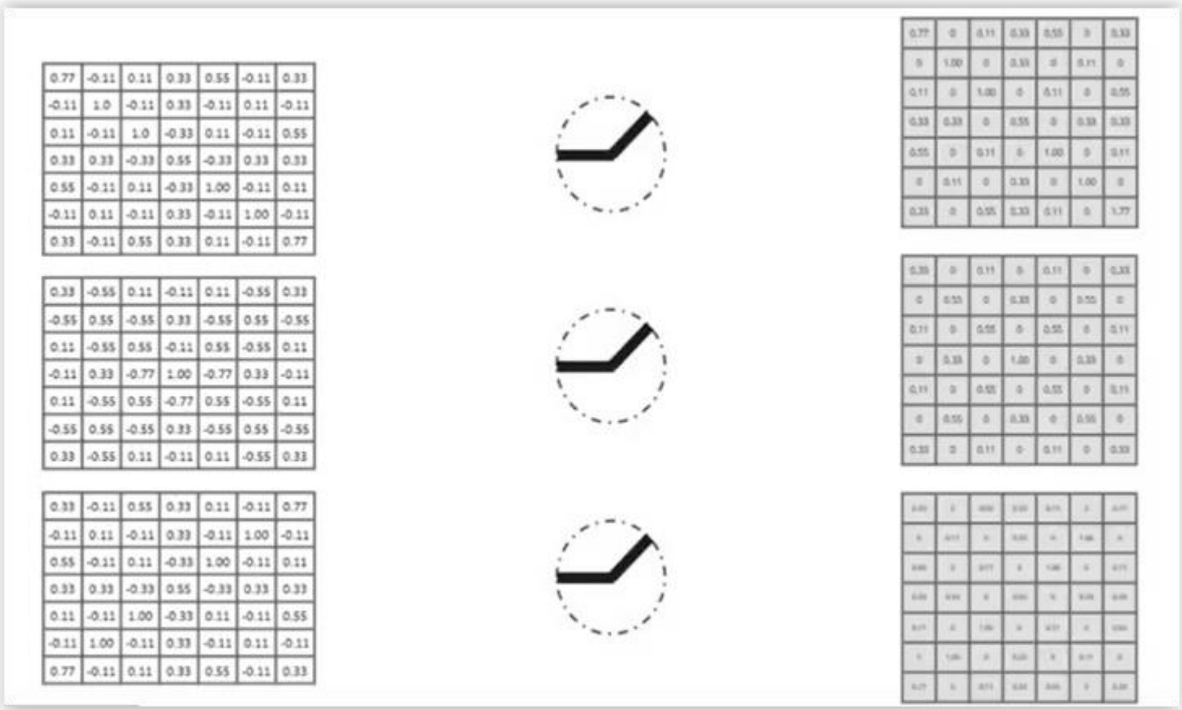
- Rectified Linear Unit (ReLu) is an activation function that fires a neuron if the input is above a certain quantity
- In this layer we remove negative image values and replace it by 0
- **This avoids values summing up to zero in the following layers**

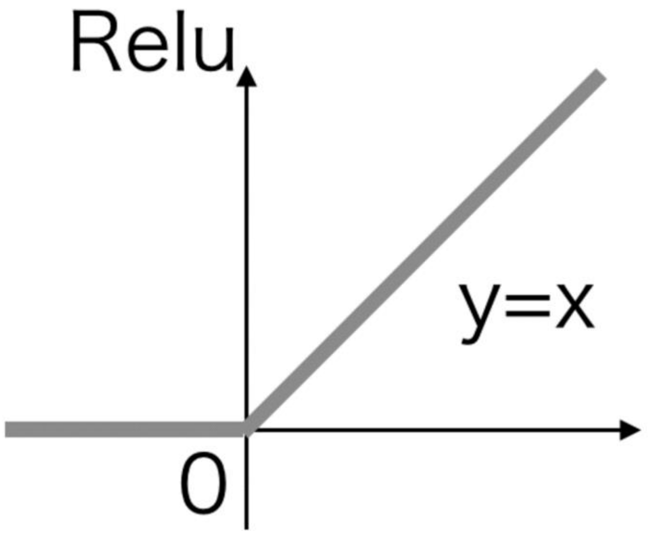




ReLu Layer

- **Rectified Linear Unit (ReLu)** is an activation function that fires a neuron if the input is above a certain quantity
- In this layer we remove negative image values and replace it by 0
- **This avoids values summing up to zero in the following layers**





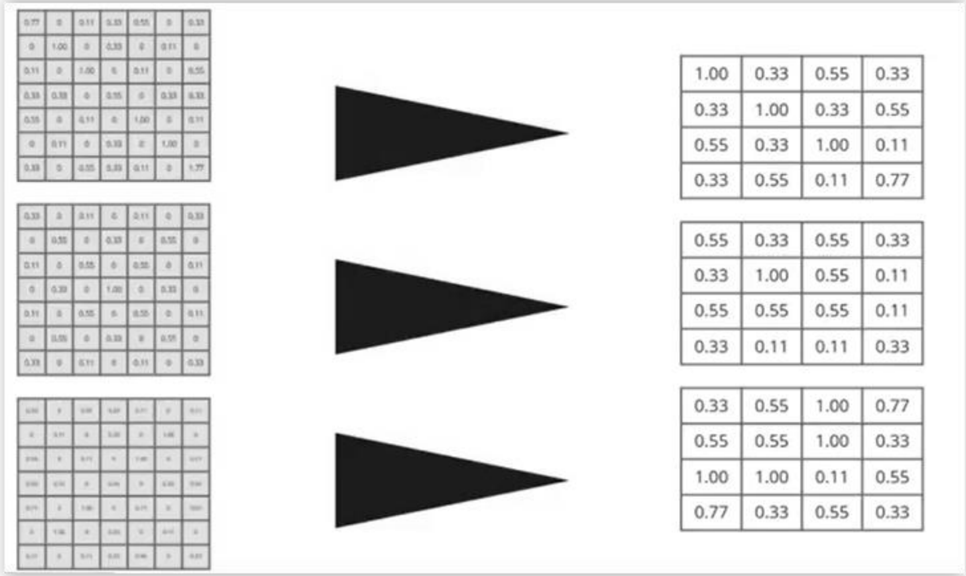
Pooling Layer

- We shrink image stack to smaller size

Steps

- Pick window size, usually 2 or 3
- Pick a stride, usually 2
- Walk your window across filtered images
- From each window, take the maximum value

Shrink the image



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

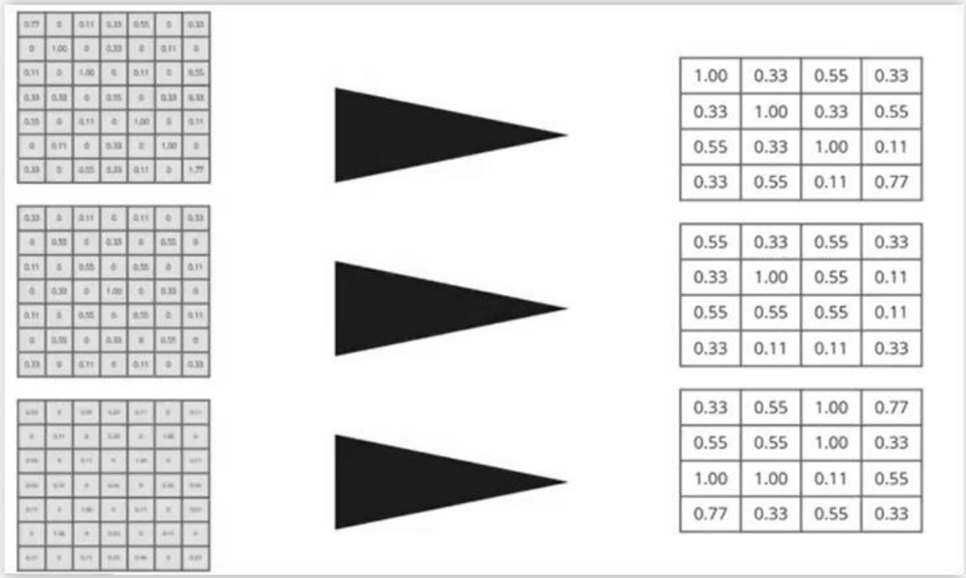
Pooling Layer

- We shrink image stack to smaller size

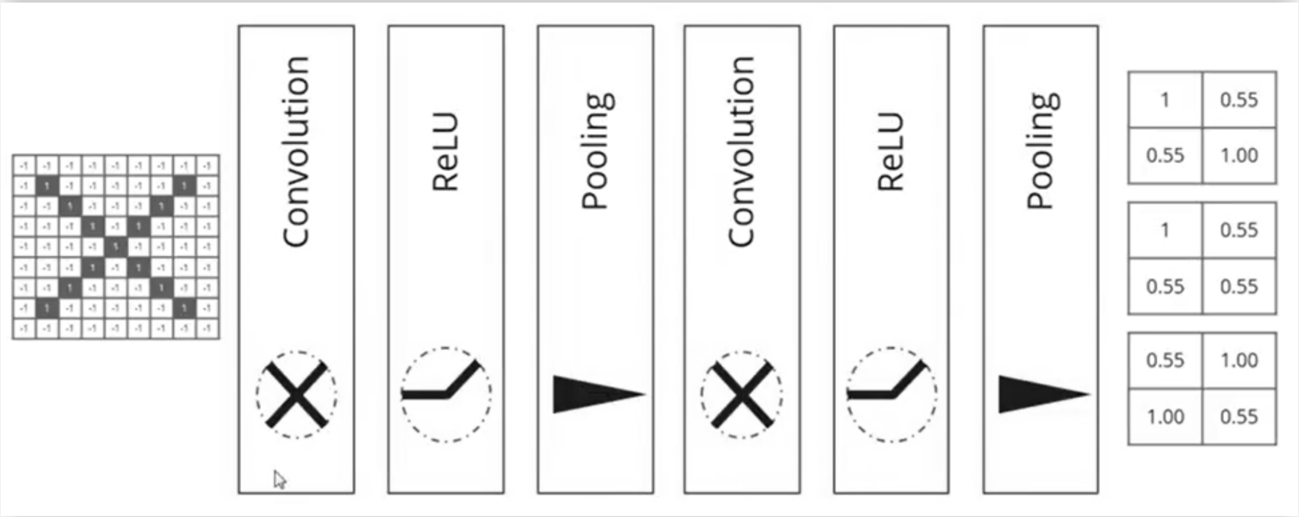
Steps

- Pick window size, usually 2 or 3
- Pick a stride, usually 2
- Walk your window across filtered images
- From each window, take the maximum value

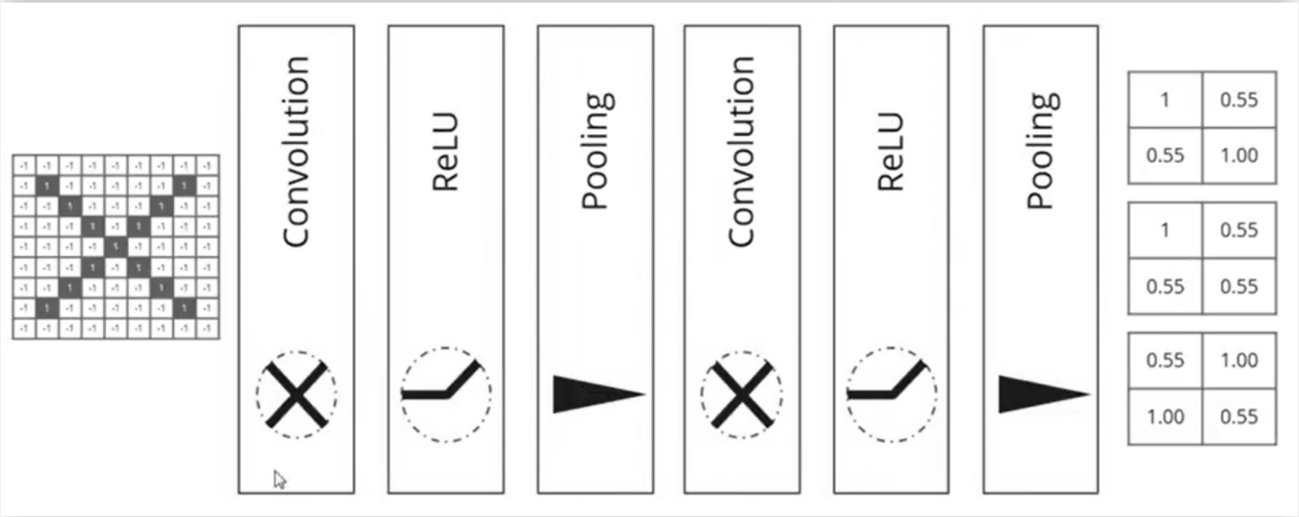
Shrink the image



Stacking up Layers

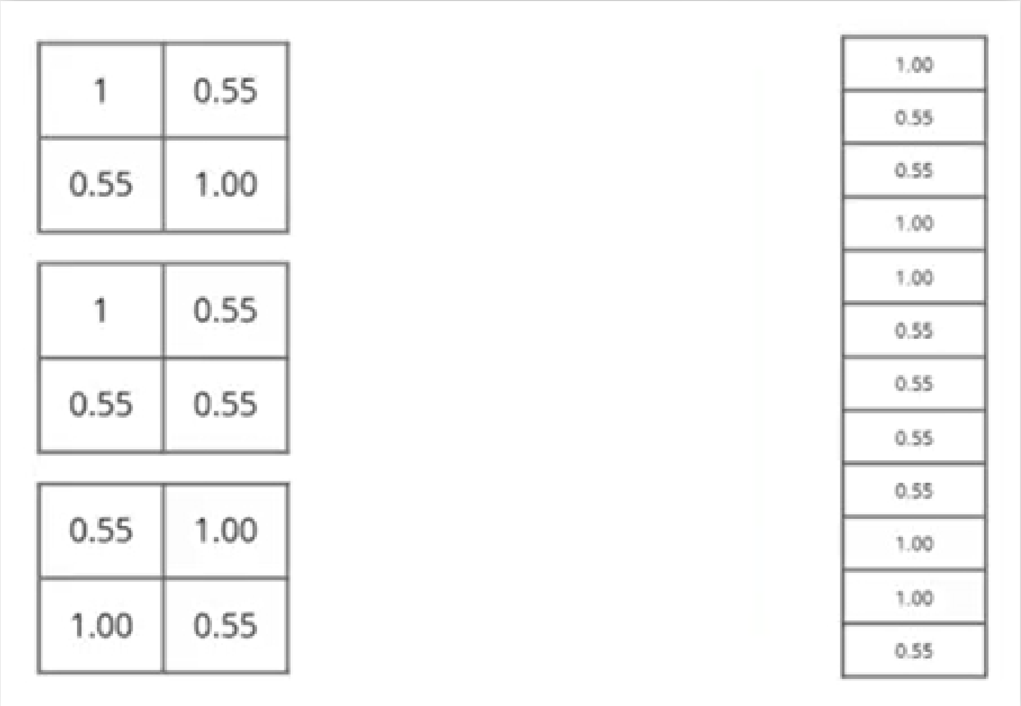


Stacking up Layers



Fully Connected Layer

- This is the final layer where classification happens
- We take our filtered and shrunk images and put them into a single list



Fully Connected Layer

- This is the final layer where classification happens
- We take our filtered and shrunk images and put them into a single list

1	0.55
0.55	1.00

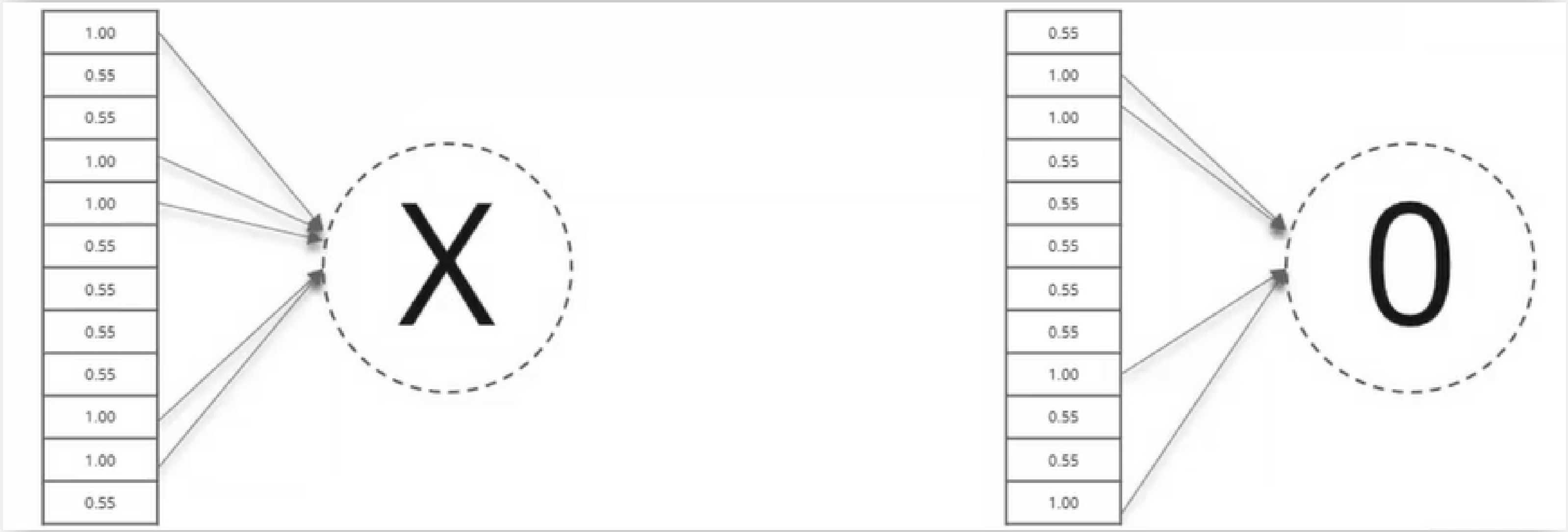
1	0.55
0.55	0.55

0.55	1.00
1.00	0.55

1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55

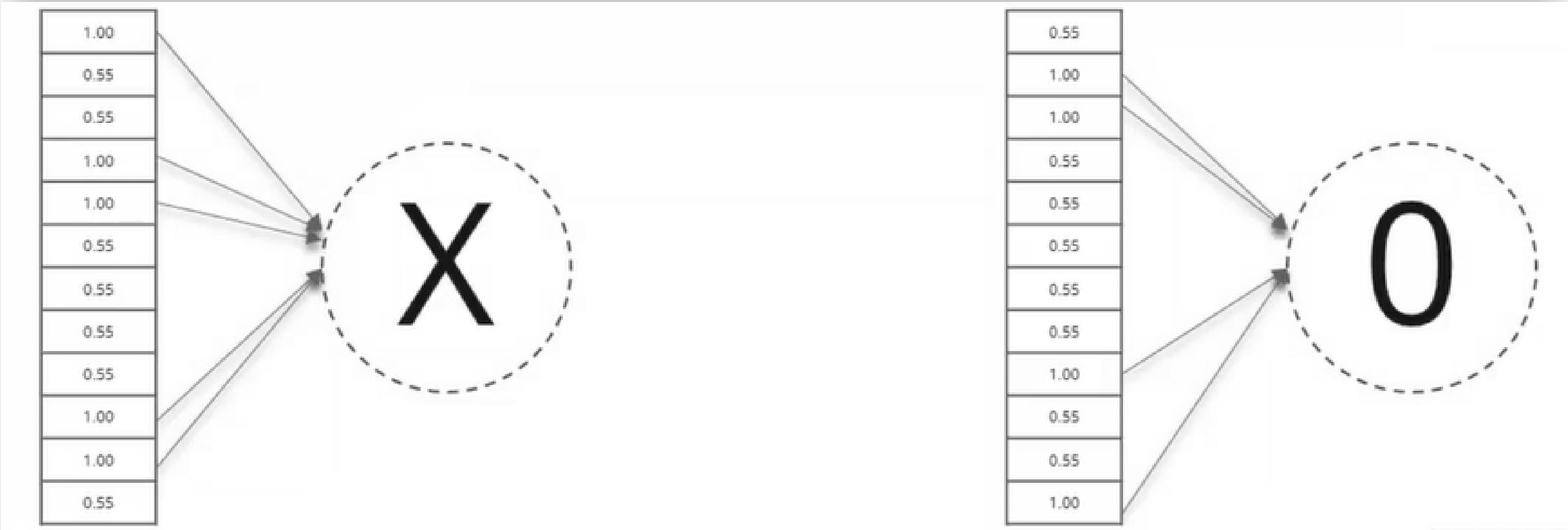
Output

When we feed "X" and "O" there will be some element in the vector that is high



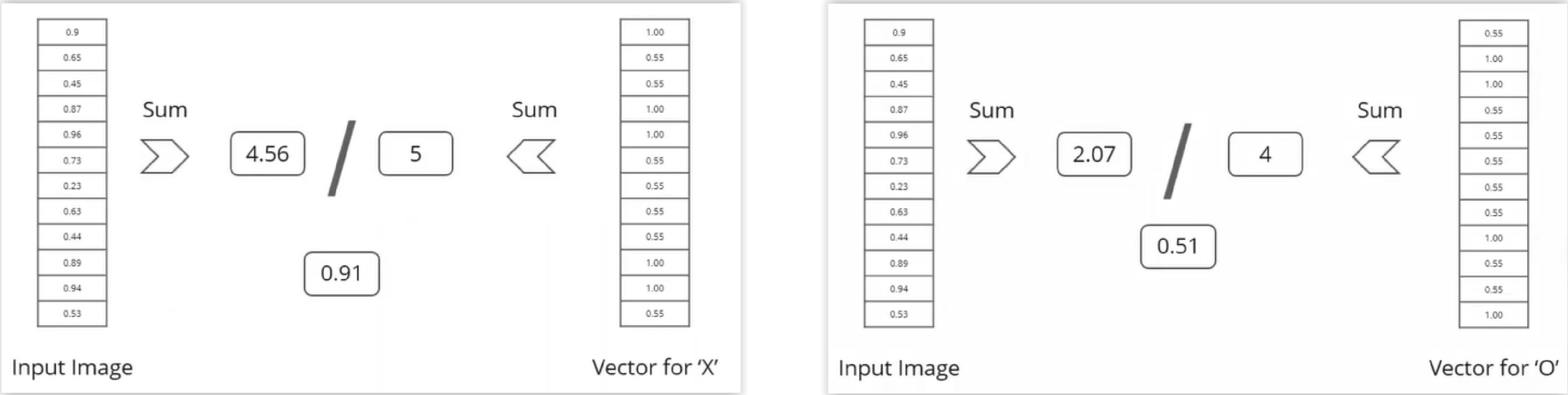
Output

When we feed "X" and "O" there will be some element in the vector that is high



Comparing the input vector with X and O

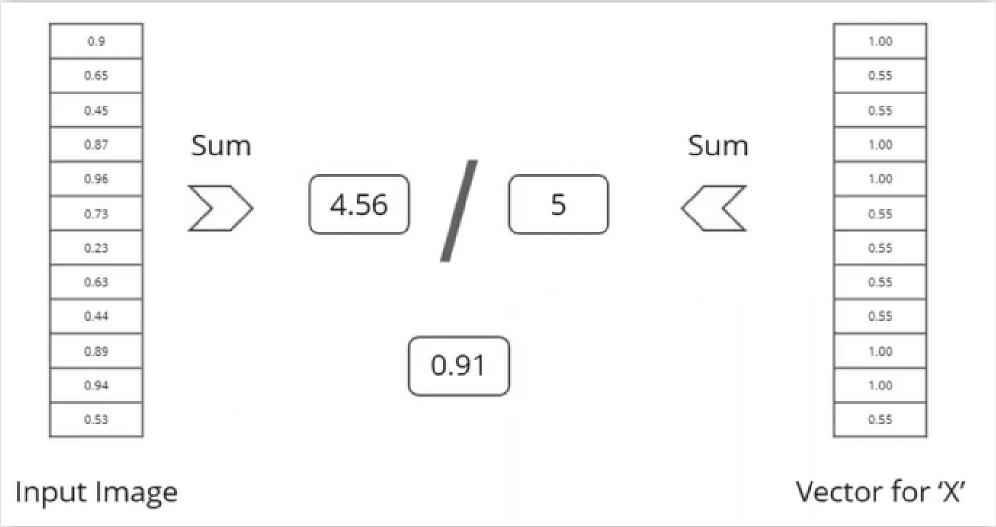
When we feed “X” and “O” there will be some element in the vector that is high

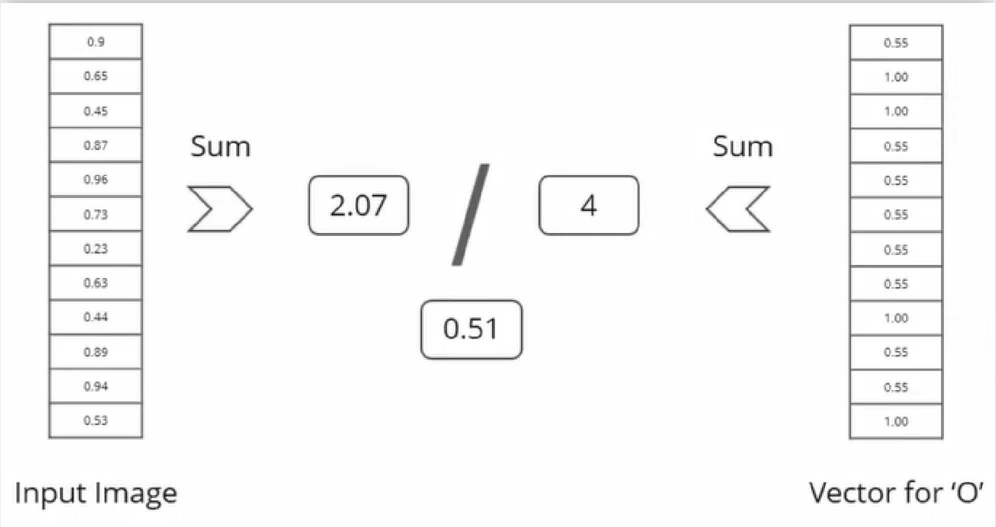


The input image is classified as X

Comparing the input vector with X and O

When we feed “X” and “O” there will be some element in the vector that is high



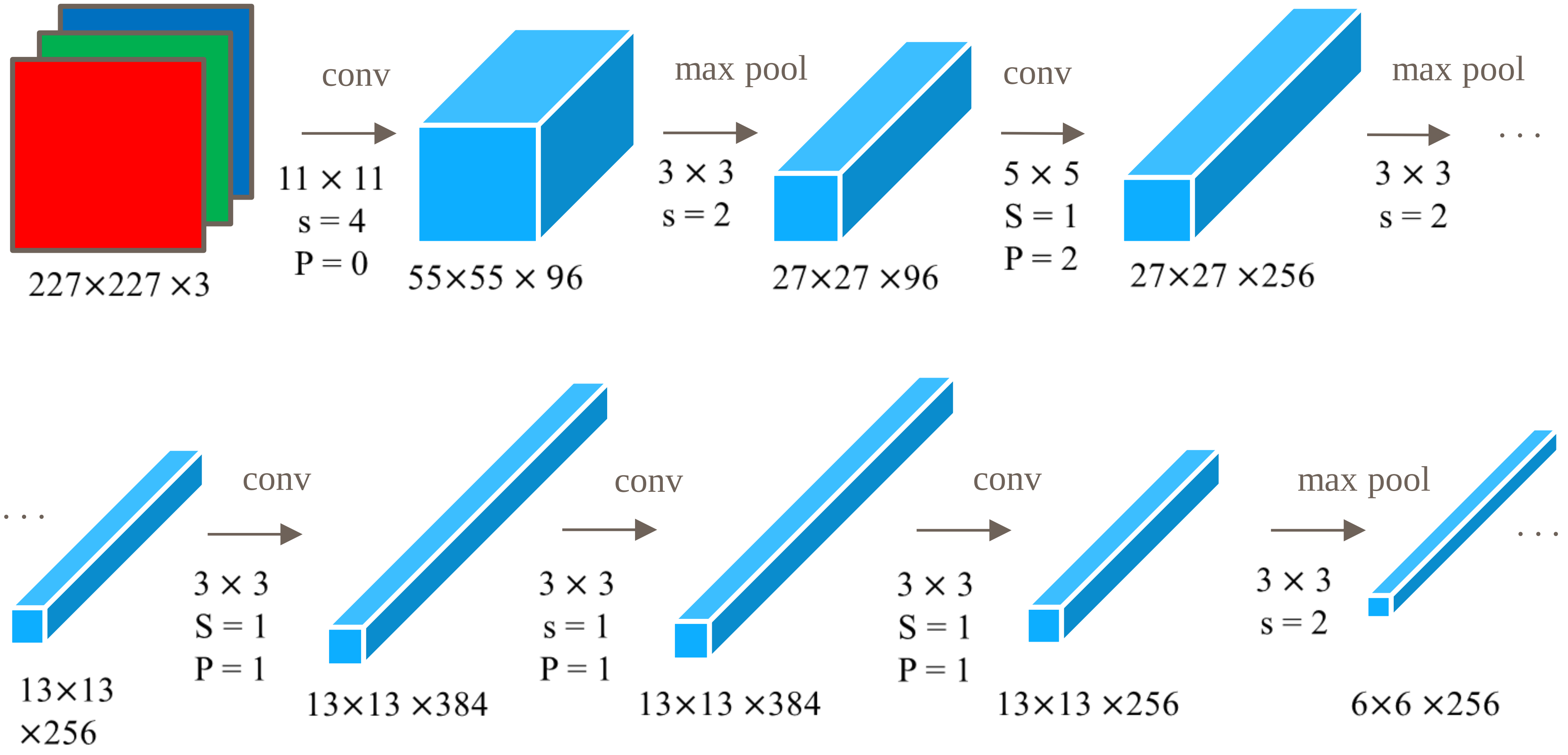


The input image is classified as X

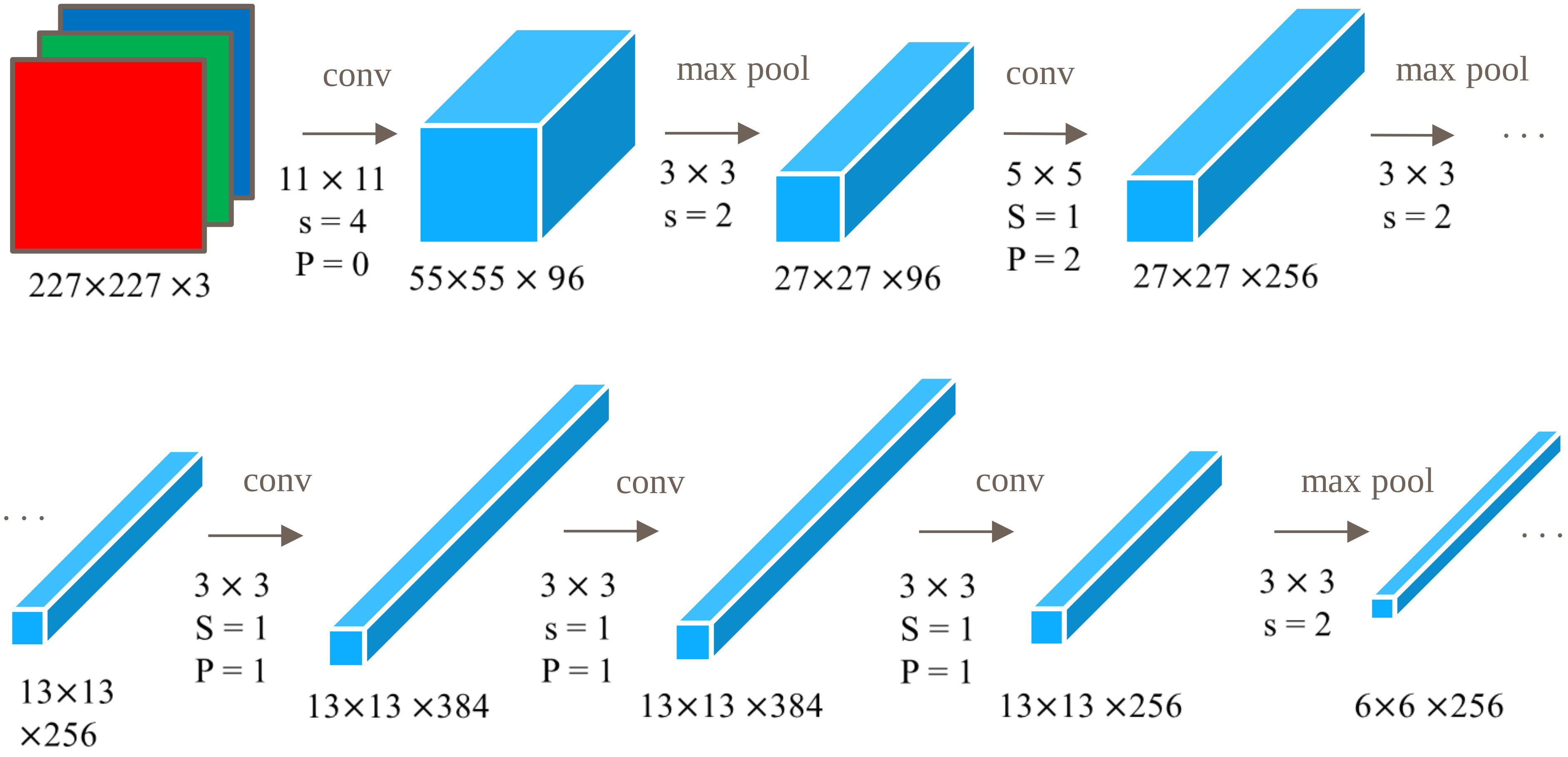
Convolutional NN Architectures

Convolutional NN Architectures

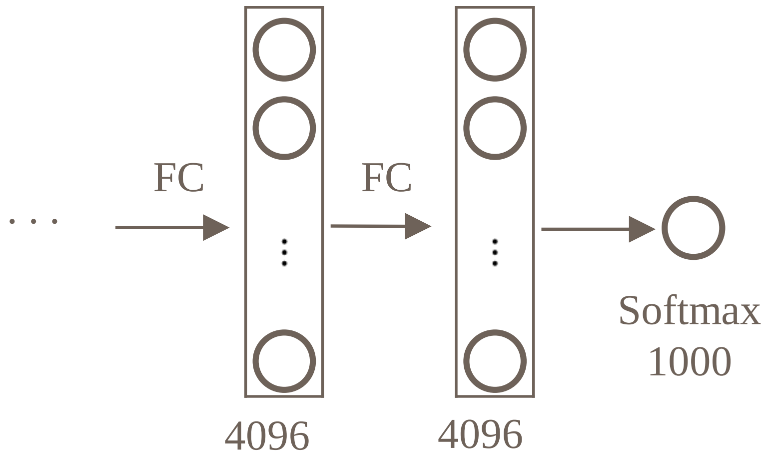
AlexNet


227×227 ×3
conv
11 × 11
s = 4
P = 0
55×55 × 96
max pool
3 × 3
s = 2
27×27 ×96
conv
5 × 5
S = 1
P = 2
27×27 ×256
max pool
3 × 3
s = 2
. . .
. . .
13×13
×256
conv
3 × 3
S = 1
P = 1
13×13 ×384
conv
3 × 3
s = 1
P = 1
13×13 ×384
conv
3 × 3
S = 1
P = 1
13×13 ×256
max pool
3 × 3
s = 2
6×6 ×256
. . .

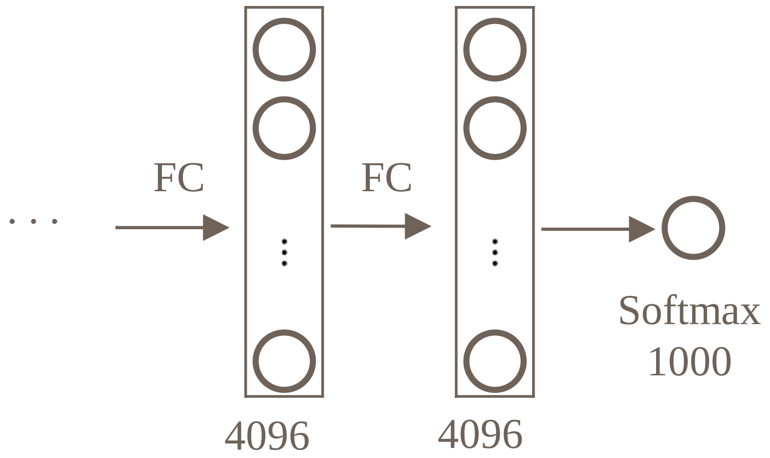
AlexNet


227×227 ×3
conv
11 × 11
s = 4
P = 0
55×55 × 96
max pool
3 × 3
s = 2
27×27 ×96
conv
5 × 5
S = 1
P = 2
27×27 ×256
max pool
3 × 3
s = 2
. . .
. . .
13×13
×256
conv
3 × 3
S = 1
P = 1
13×13 ×384
conv
3 × 3
s = 1
P = 1
13×13 ×384
conv
3 × 3
S = 1
P = 1
13×13 ×256
max pool
3 × 3
s = 2
6×6 ×256
. . .

AlexNet



AlexNet



VGGNet

Very Deep Convolutional Networks For Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman; 2015

The runner-up at the ILSVRC 2014 competition

Significantly deeper than AlexNet

140 million parameters

VGGNet

Very Deep Convolutional Networks For Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman; 2015

The runner-up at the ILSVRC 2014 competition

Significantly deeper than AlexNet

140 million parameters

Input
3x3 conv, 64
3x3 conv, 64
Pool 1/2
3x3 conv, 128
3x3 conv, 128
Pool 1/2
3x3 conv, 256
3x3 conv, 256
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
FC 4096
FC 4096
FC 1000
Softmax

VGGNet

REPSOL

Smaller filters

Only 3x3 CONV filters, stride 1, pad 1
and 2x2 MAX POOL , stride 2

Deeper network

AlexNet: 8 layers
VGGNet: 16 - 19 layers

Input
3x3 conv, 64
3x3 conv, 64
Pool 1/2
3x3 conv, 128
3x3 conv, 128
Pool 1/2
3x3 conv, 256
3x3 conv, 256
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
FC 4096
FC 4096
FC 1000
Softmax

VGGNet

Smaller filters

Only 3x3 CONV filters, stride 1, pad 1
and 2x2 MAX POOL , stride 2

Deeper network

AlexNet: 8 layers
VGGNet: 16 - 19 layers

GoogleNet

Going Deeper with Convolutions - Christian Szegedy et al.; 2015
ILSVRC 2014 competition winner
Also significantly deeper than AlexNet
x12 less parameters than AlexNet
Focused on computational efficiency

GoogleNet

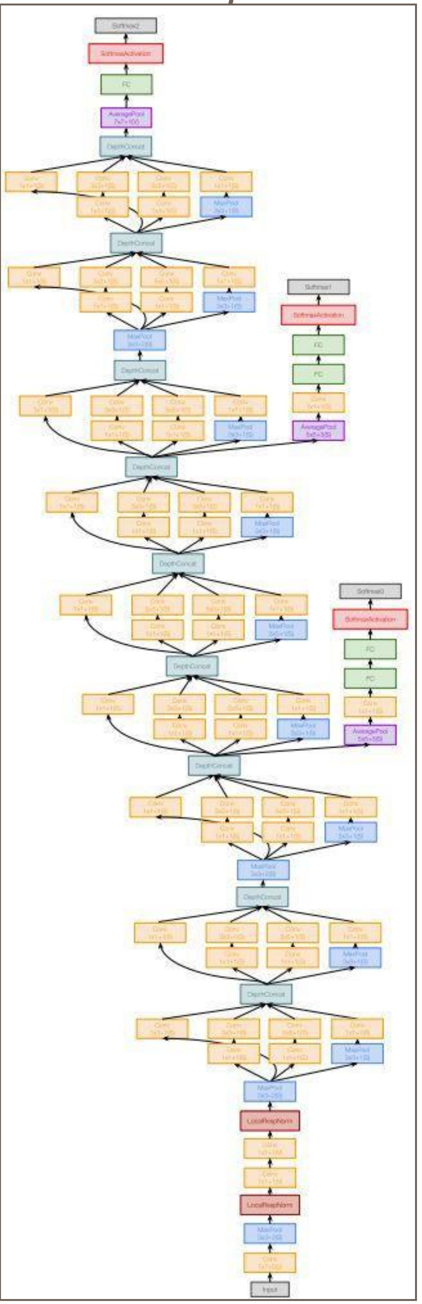
Going Deeper with Convolutions - Christian Szegedy et al.; 2015

ILSVRC 2014 competition winner

Also significantly deeper than AlexNet

x12 less parameters than AlexNet

Focused on computational efficiency



GoogleNet

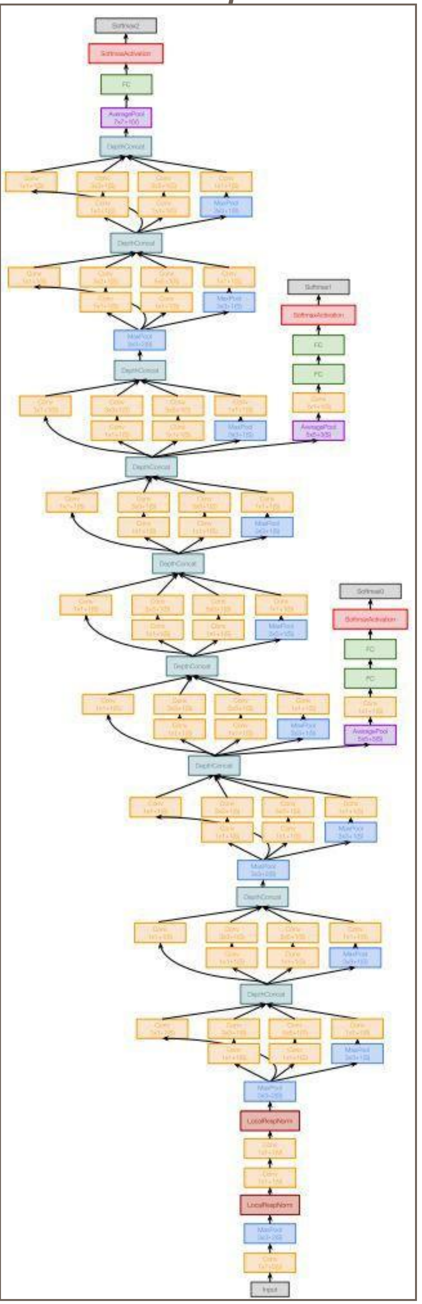
22 layers

Efficient **“Inception” module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure

No FC layers

Only 5 million parameters!

ILSVRC’14 classification winner (6.7% top 5 error)



GoogleNet

REPSOL

22 layers

Efficient **"Inception" module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure

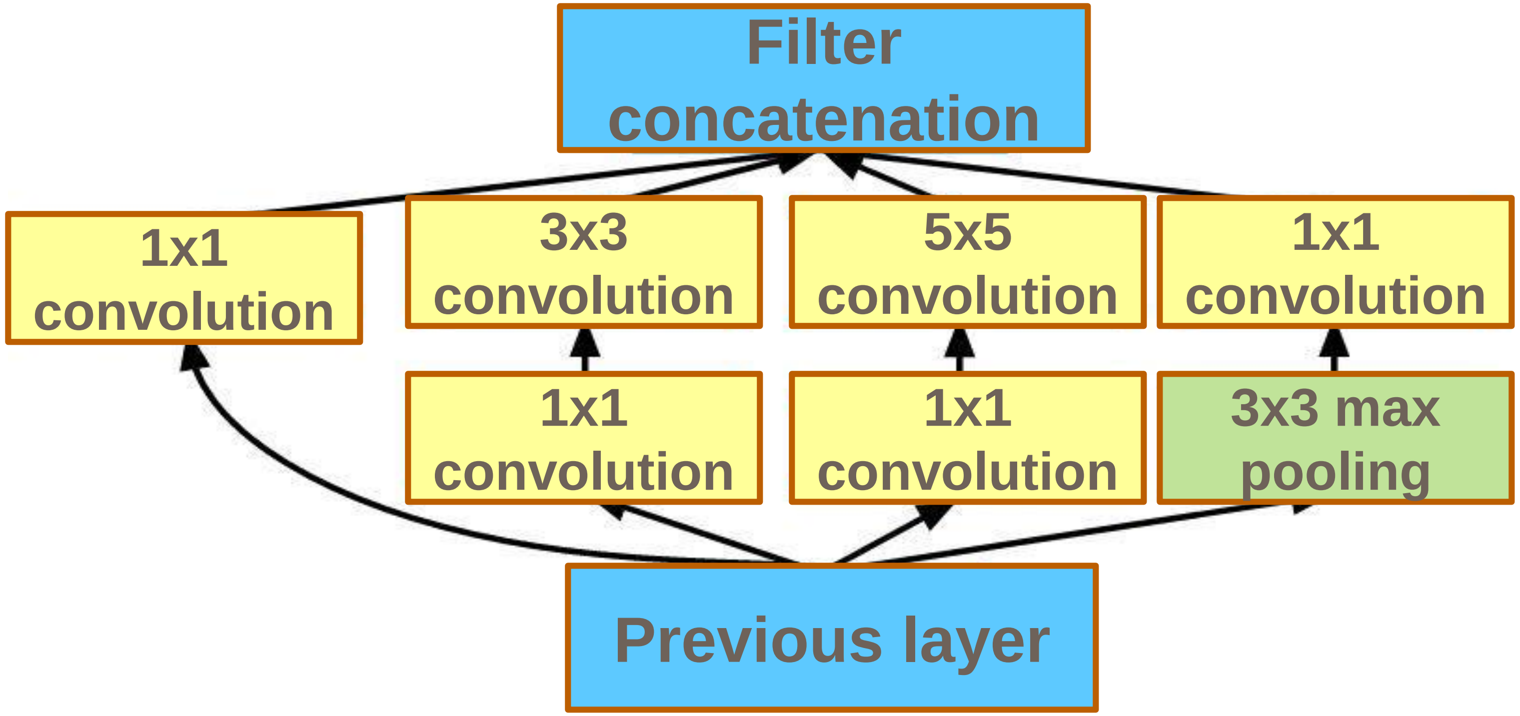
No FC layers

Only 5 million parameters!

ILSVRC'14 classification winner (6.7% top 5 error)

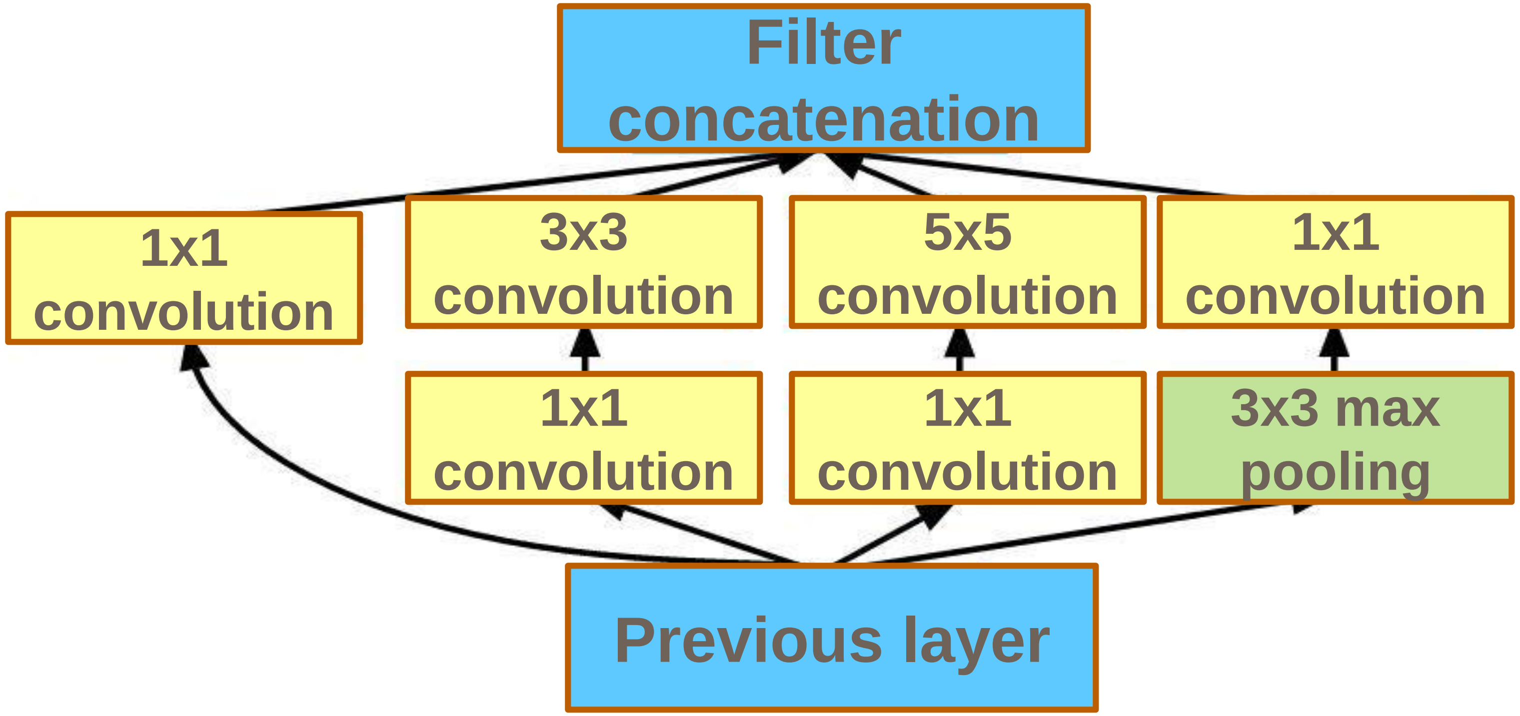
GoogleNet

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other



GoogleNet

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other



ResNet

Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015

Extremely deep network – 152 layers

Deeper neural networks are more difficult to train.

Deep networks suffer from vanishing and exploding gradients.

Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

ResNet

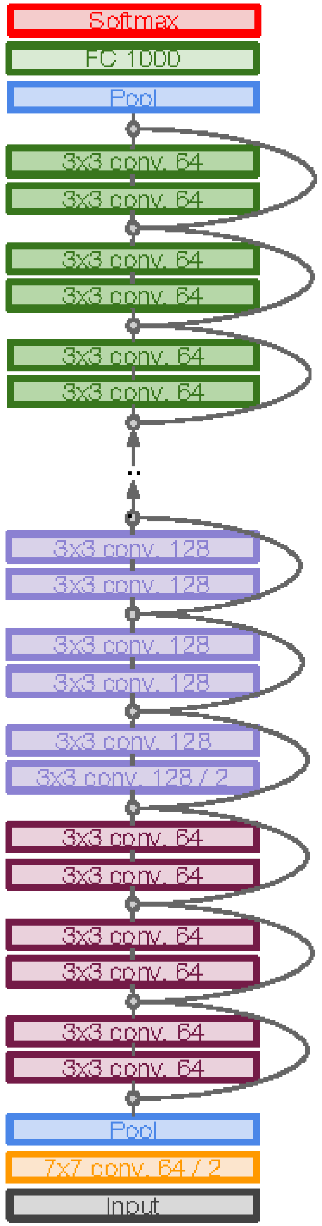
Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015

Extremely deep network – 152 layers

Deeper neural networks are more difficult to train.

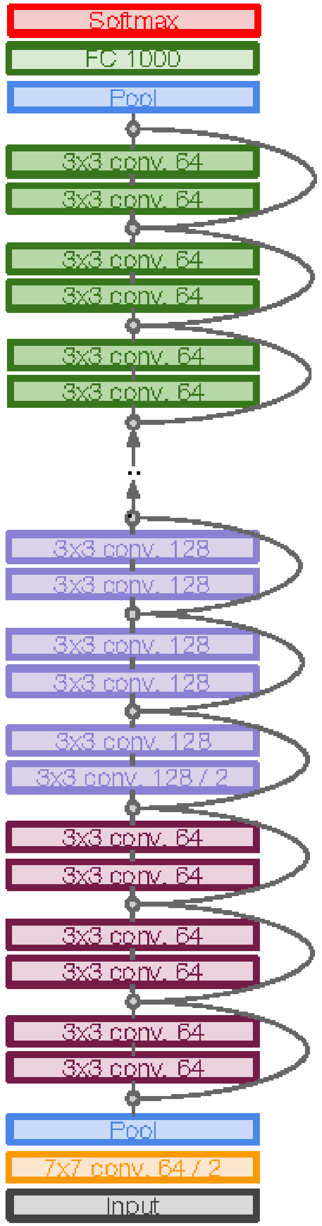
Deep networks suffer from vanishing and exploding gradients.

Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.



ResNet

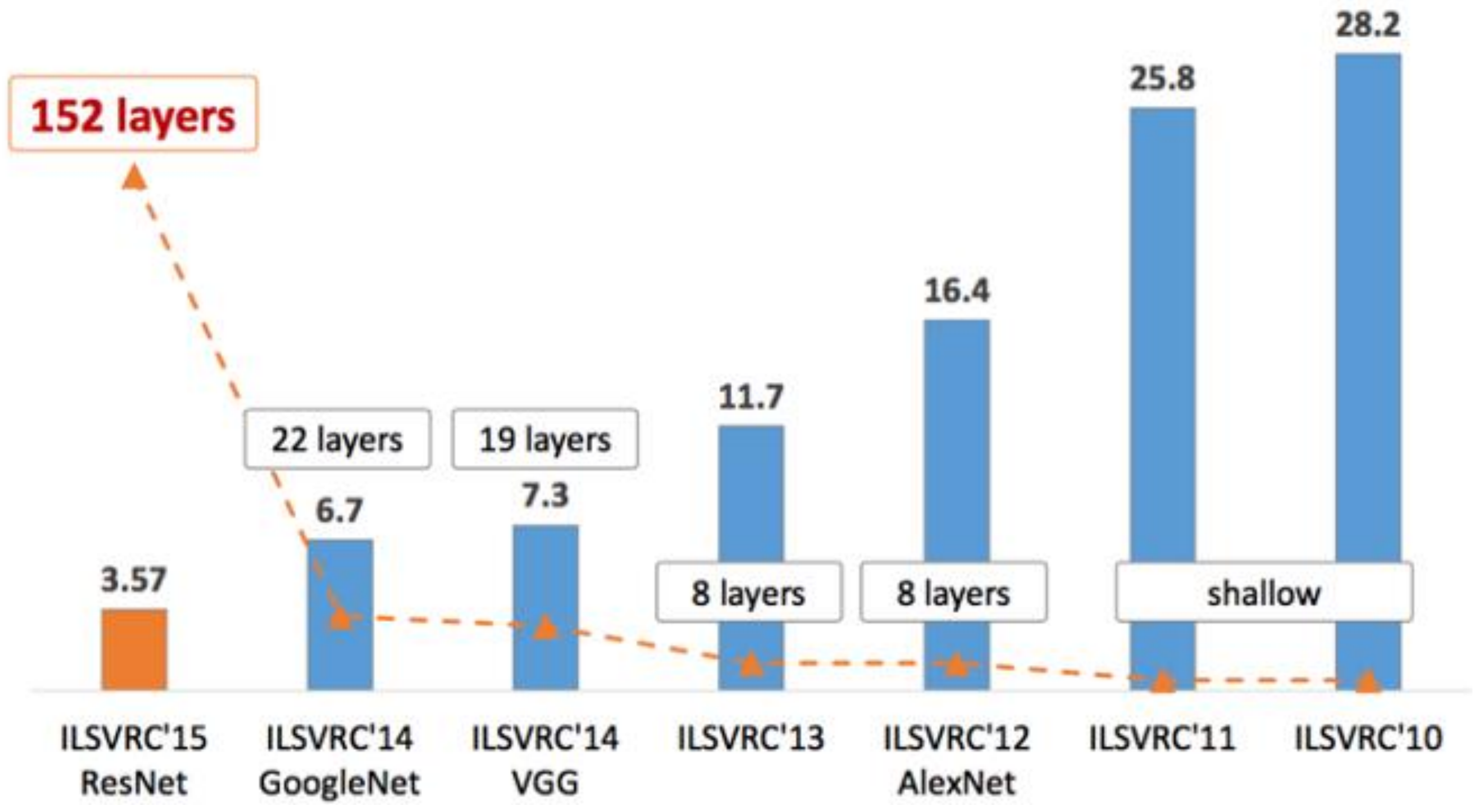
- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
  Swept all classification and detection competitions in ILSVRC'15



ResNet

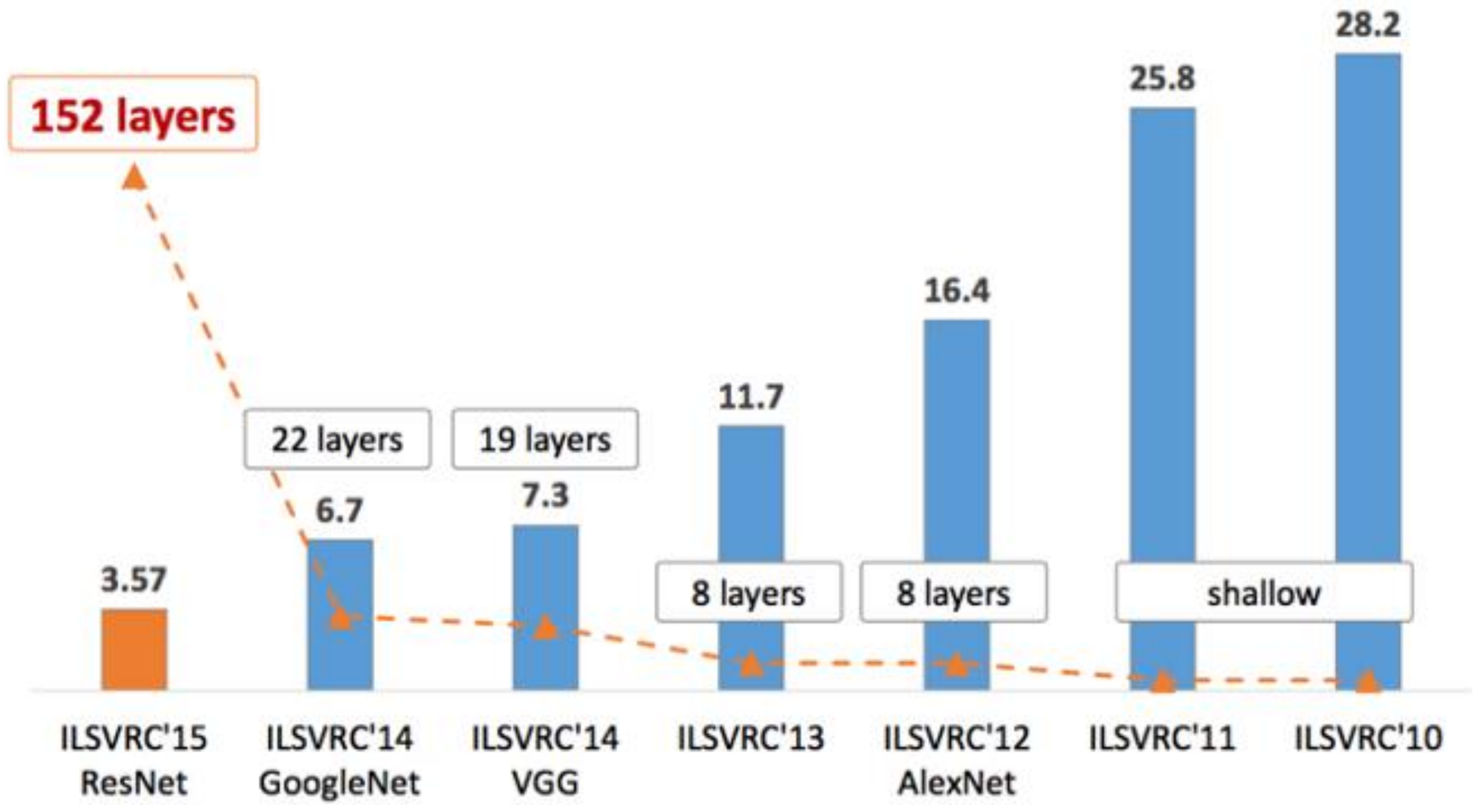
- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
  Swept all classification and detection competitions in ILSVRC'15

Comparison of Convolutional nets



Error on Top 5 score, using imageNet data

Comparison of Convolutional nets



Error on Top 5 score, using imageNet data

Remember:

- A **Convolutional NN** consists of an **input** and an **output** layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of **convolutional** layers, **pooling** layers, fully **connected** layers and **normalization** layers
- **Convolutional neural networks** are the **go to** NN for **computer vision** applications

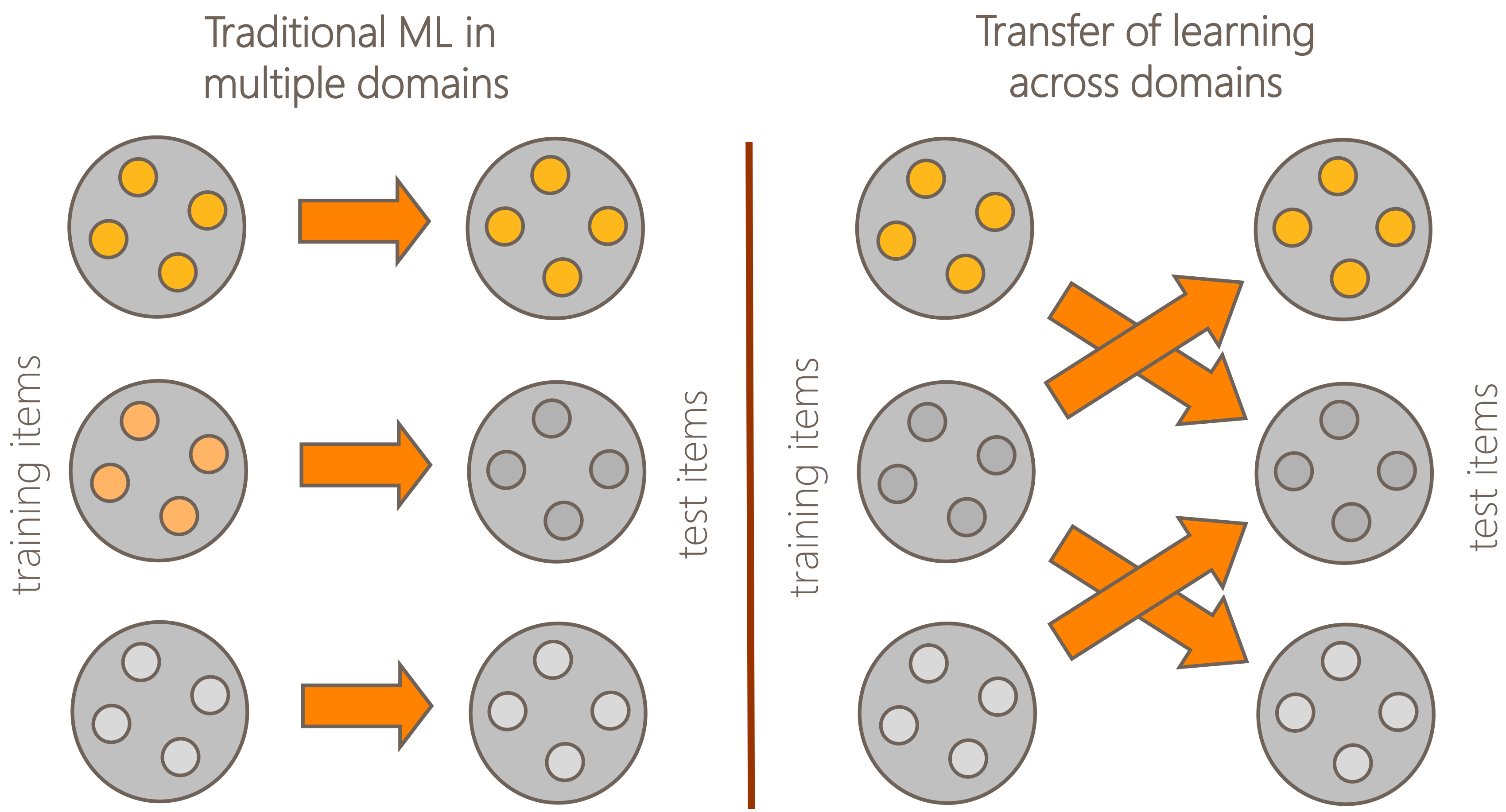
Remember:

- A **Convolutional NN** consists of an **input** and an **output** layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of **convolutional** layers, **pooling** layers, fully **connected** layers and **normalization** layers
- **Convolutional neural networks** are the **go to** NN for **computer vision** applications

Transfer learning

Transfer learning

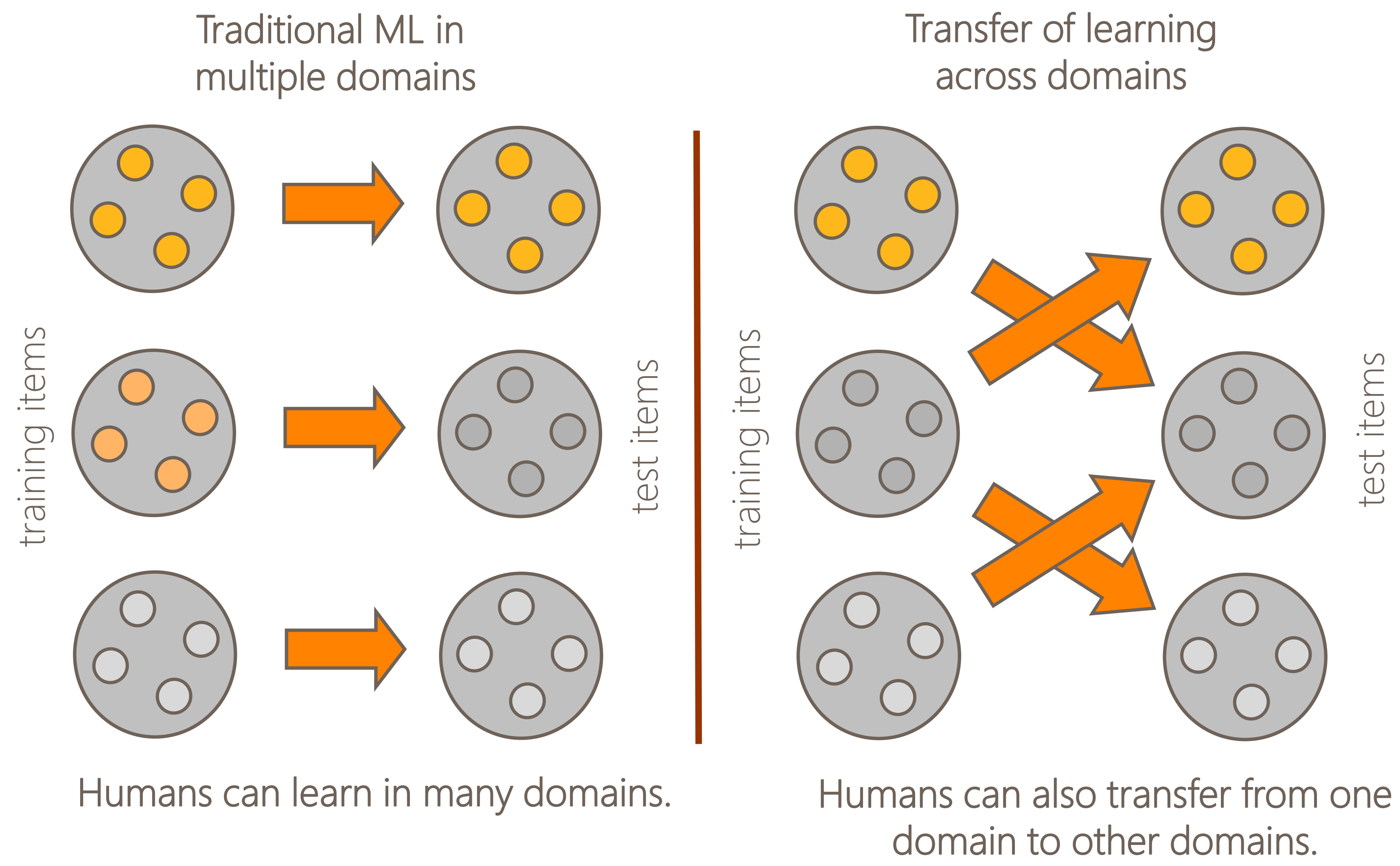
Traditional Machine Learning vs Transfer Learning



Humans can learn in many domains.

Humans can also transfer from one domain to other domains.

Traditional Machine Learning vs Transfer Learning



Motivation for Transfer Learning

- In some domains, labeled data are in short supply.
- In some domains, the calibration effort is very expensive.
- In some domains, the learning process is time consuming.

- How to extract knowledge learnt from related domains to help learning in a target domain with a few labeled data points?
- How to extract knowledge learnt from related domains to speed up learning in a target domain?

Transfer Learning is the **fastest and easiest way to build a deep learning model** without worrying about how much data you have

Motivation for Transfer Learning

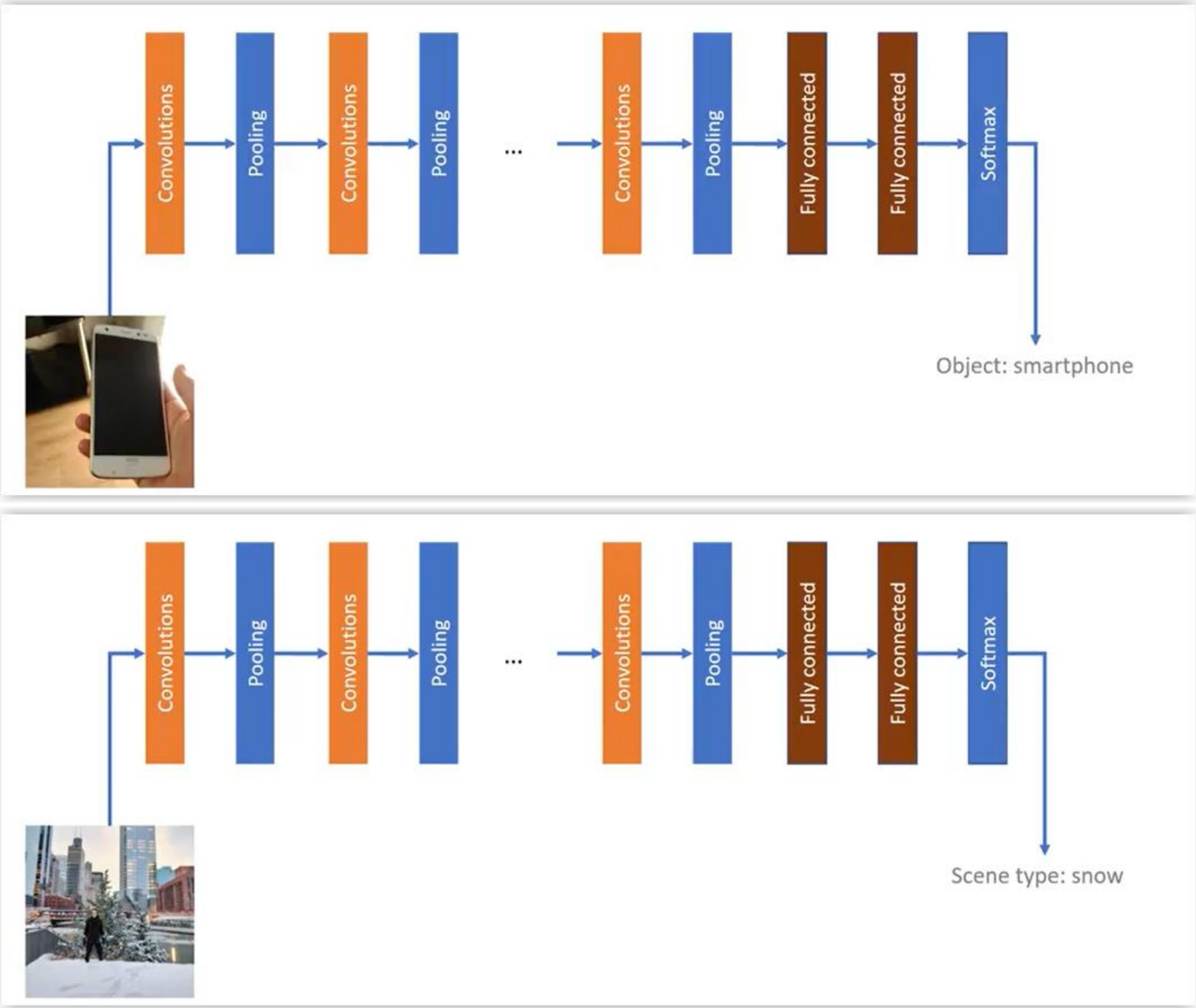
- In some domains, labeled data are in short supply.
- In some domains, the calibration effort is very expensive.
- In some domains, the learning process is time consuming.

- How to extract knowledge learnt from related domains to help learning in a target domain with a few labeled data points?
- How to extract knowledge learnt from related domains to speed up learning in a target domain?

Transfer Learning is the fastest and easiest way to build a deep learning model without worrying about how much data you have

Transfer Learning with Pretrained Models (1/3)

We can use a model trained for classifying images, to classify scenes

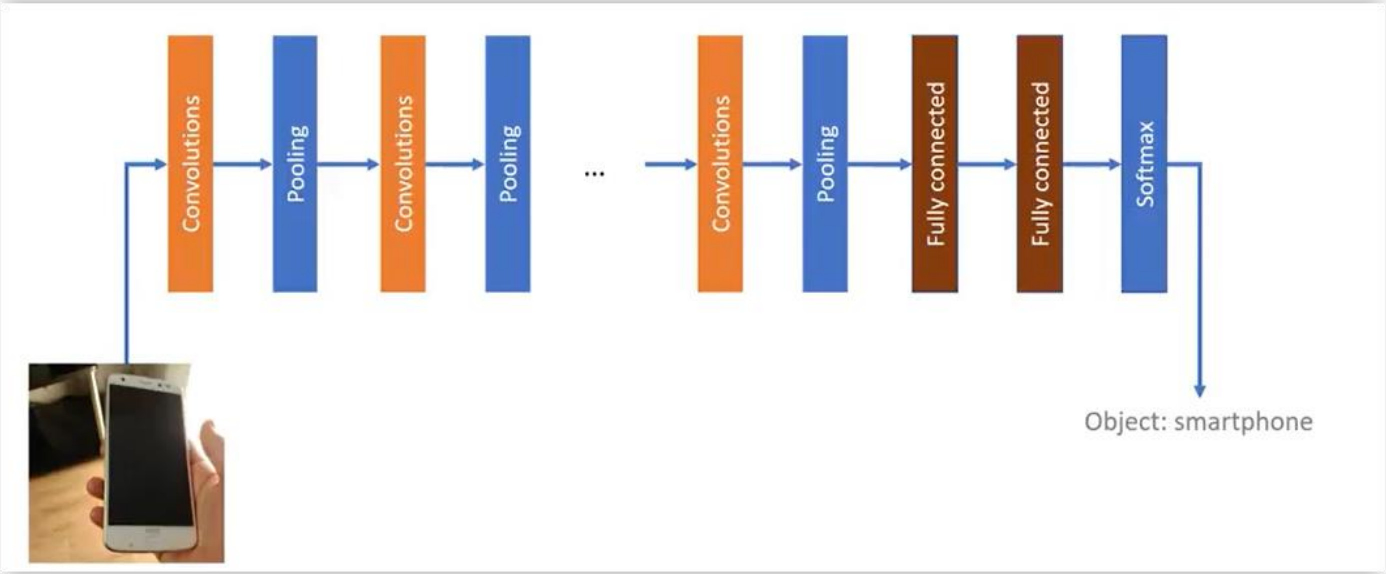


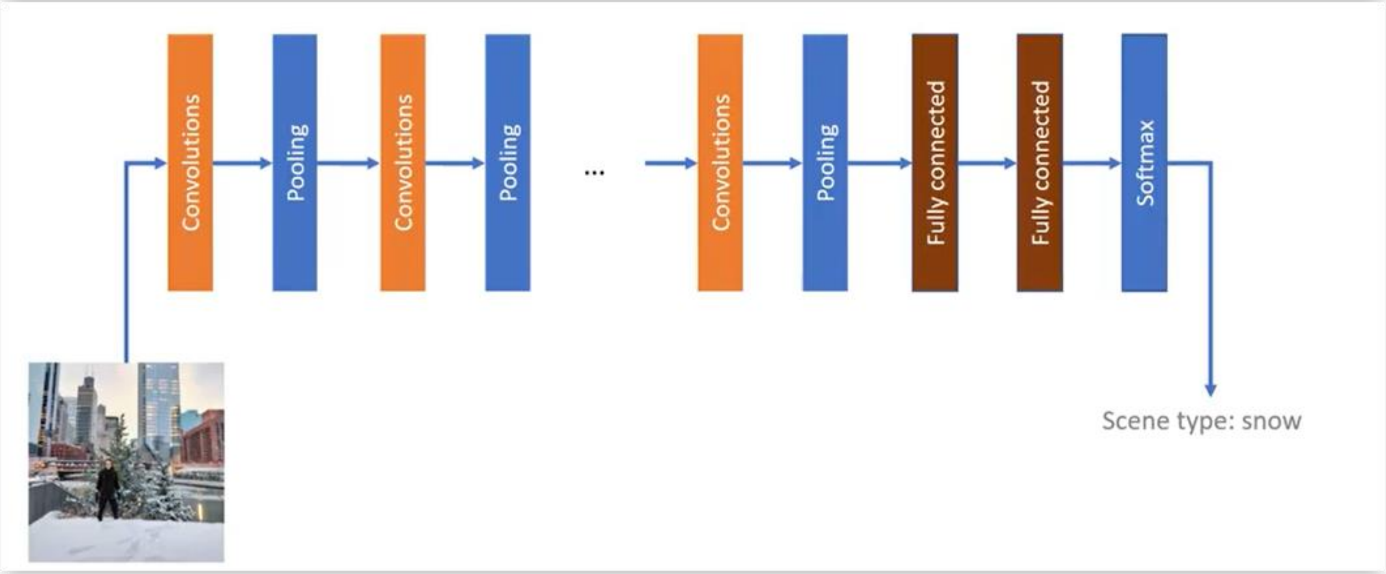
TL is particularly relevant in computer vision tasks

Transfer Learning with Pretrained Models (1/3)

REPSOL

We can use a model trained for classifying images, to classify scenes

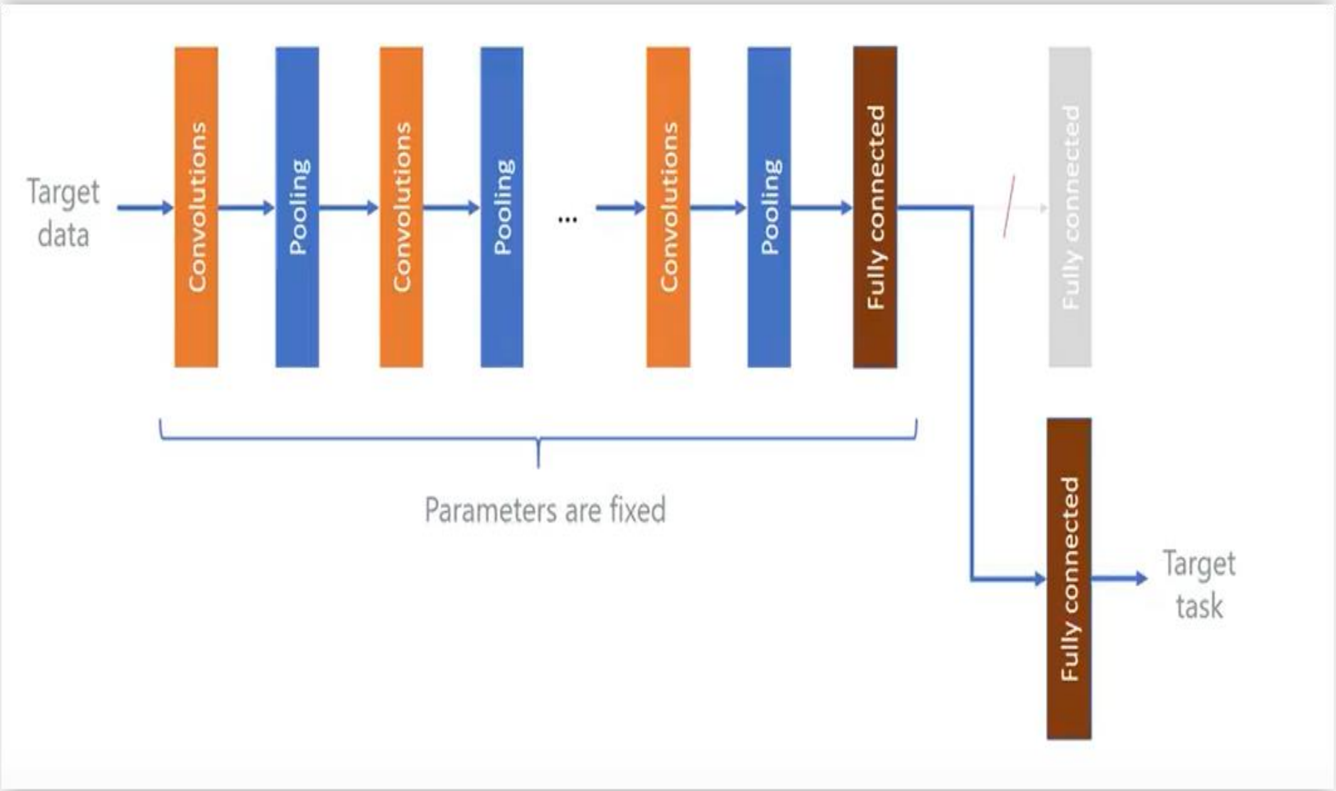




TL is particularly relevant in computer vision tasks

Transfer Learning with Pretrained Models (2/3)

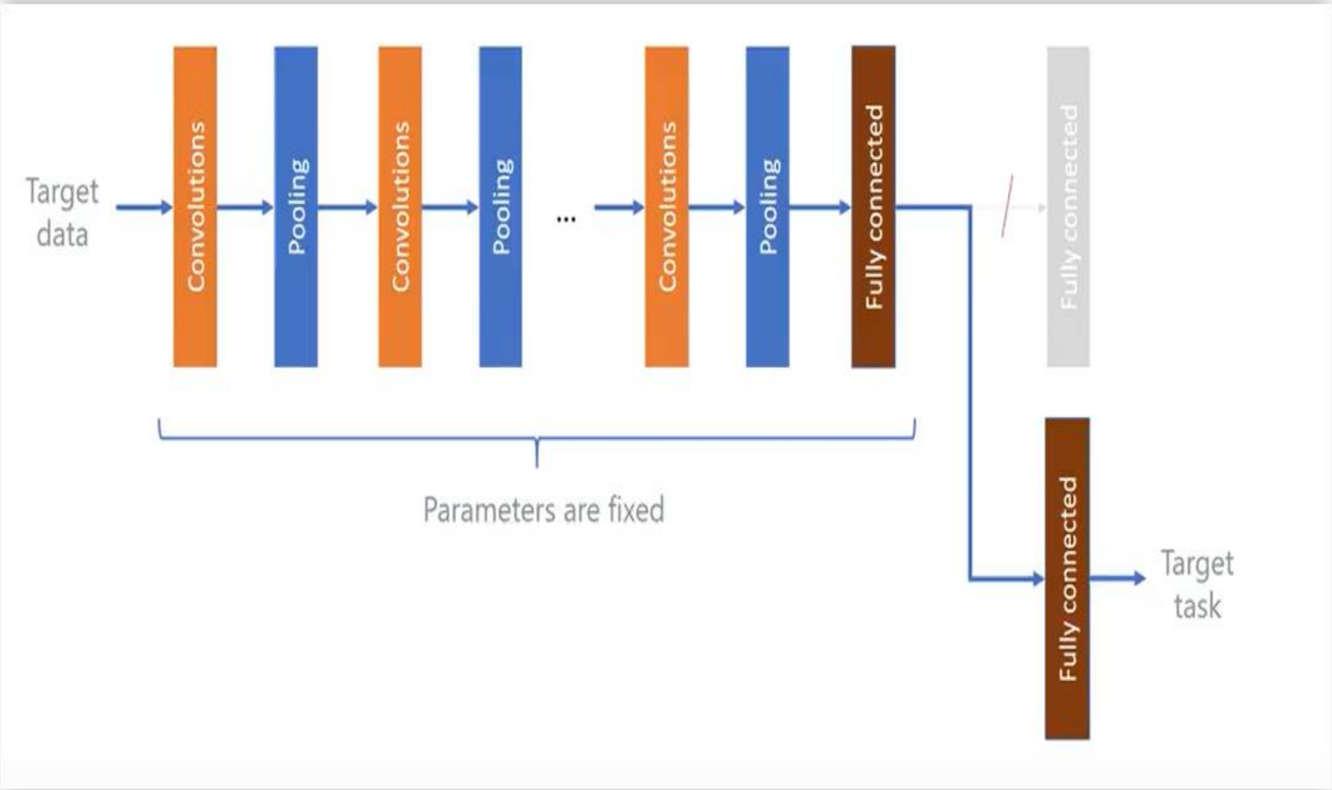
Remove top layer of already built model



Train parameters only on the top layer for the new task

Transfer Learning with Pretrained Models (2/3)

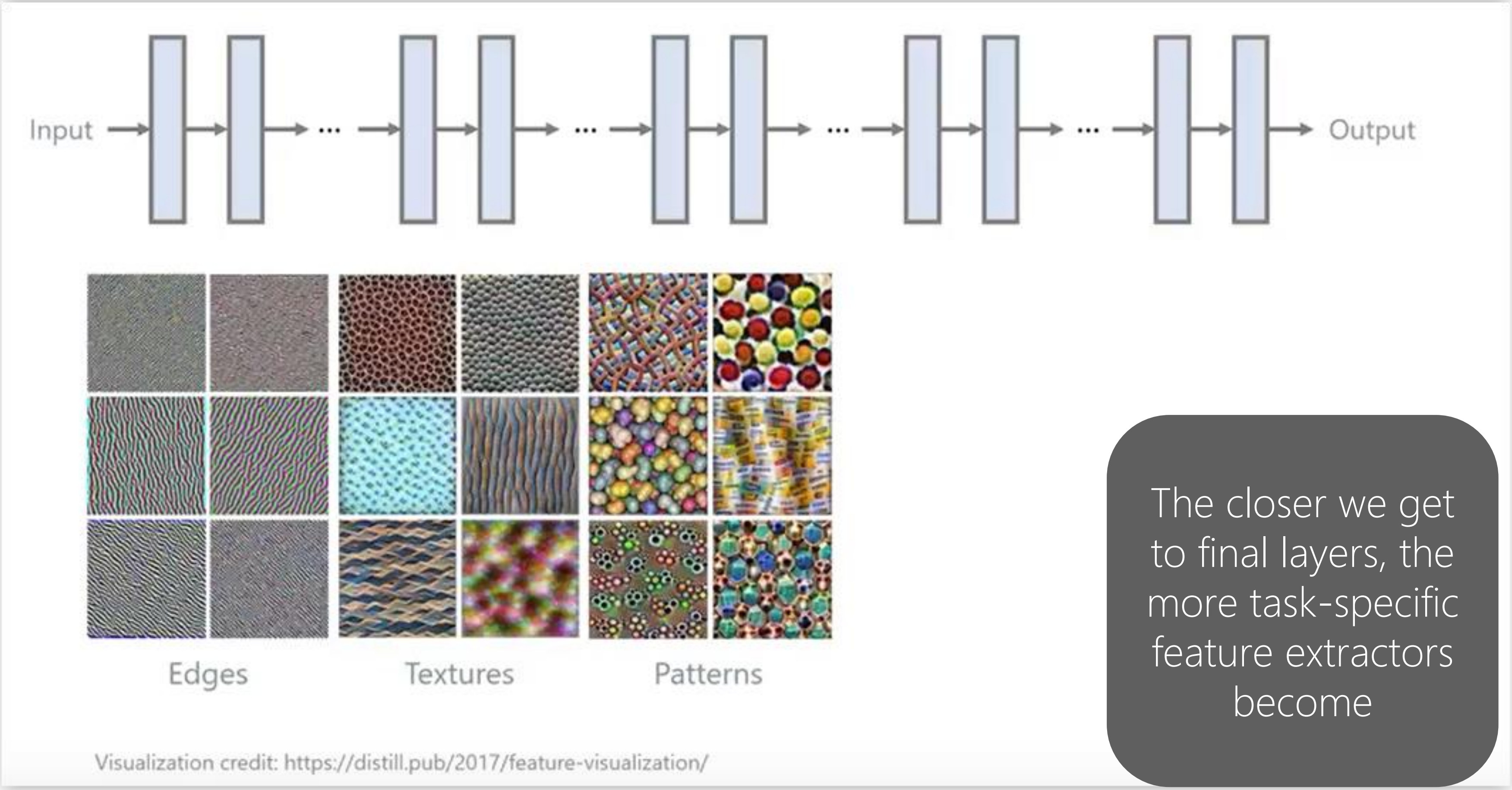
Remove top layer of already built model



Train parameters only on the top layer for the new task

Transfer Learning with Pretrained Models (3/3)

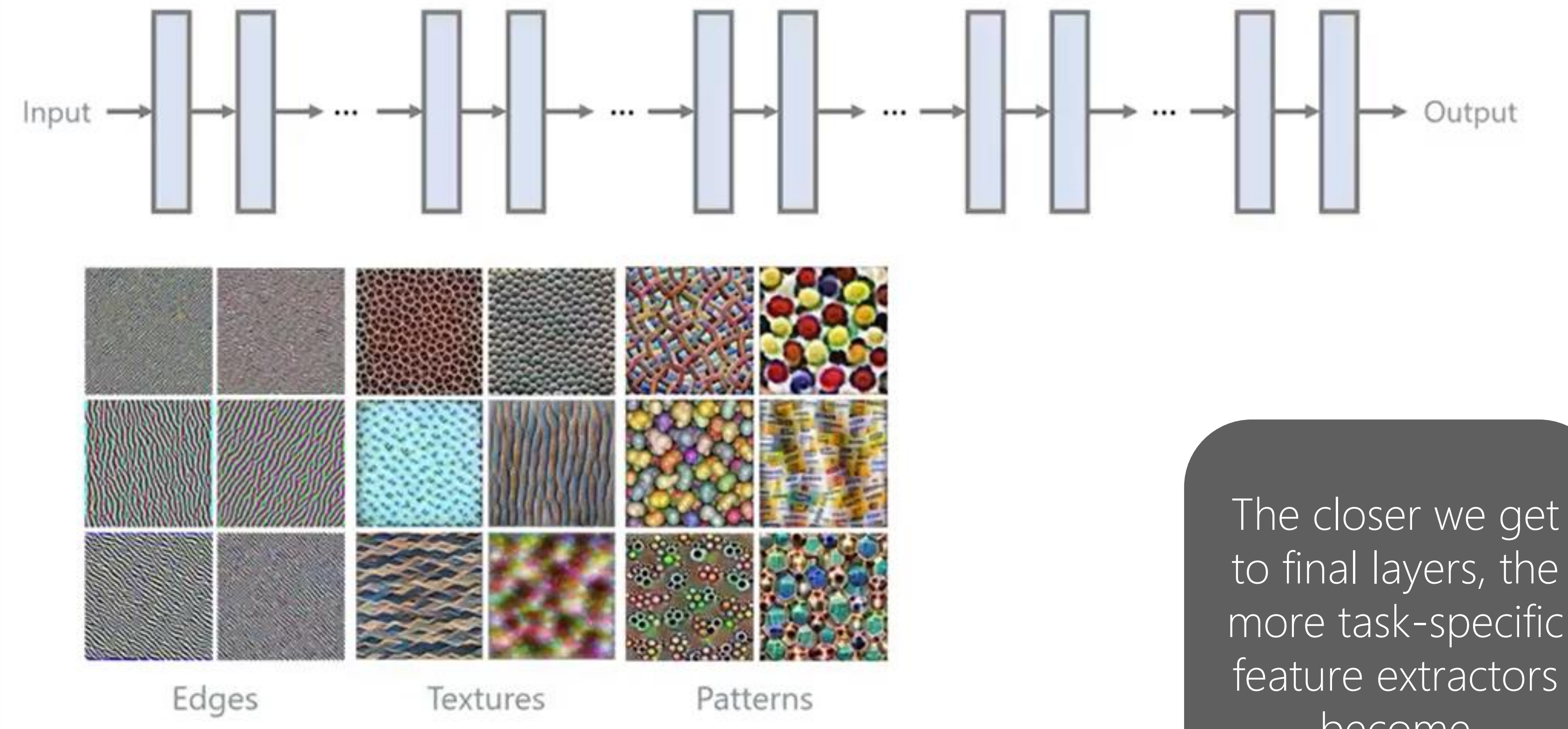
Works because early layer capture similar features. Generic feature extractors that can be used in different settings



The closer we get to final layers, the more task-specific feature extractors become

Transfer Learning with Pretrained Models (3/3)

Works because early layer capture similar features. Generic feature extractors that can be used in different settings

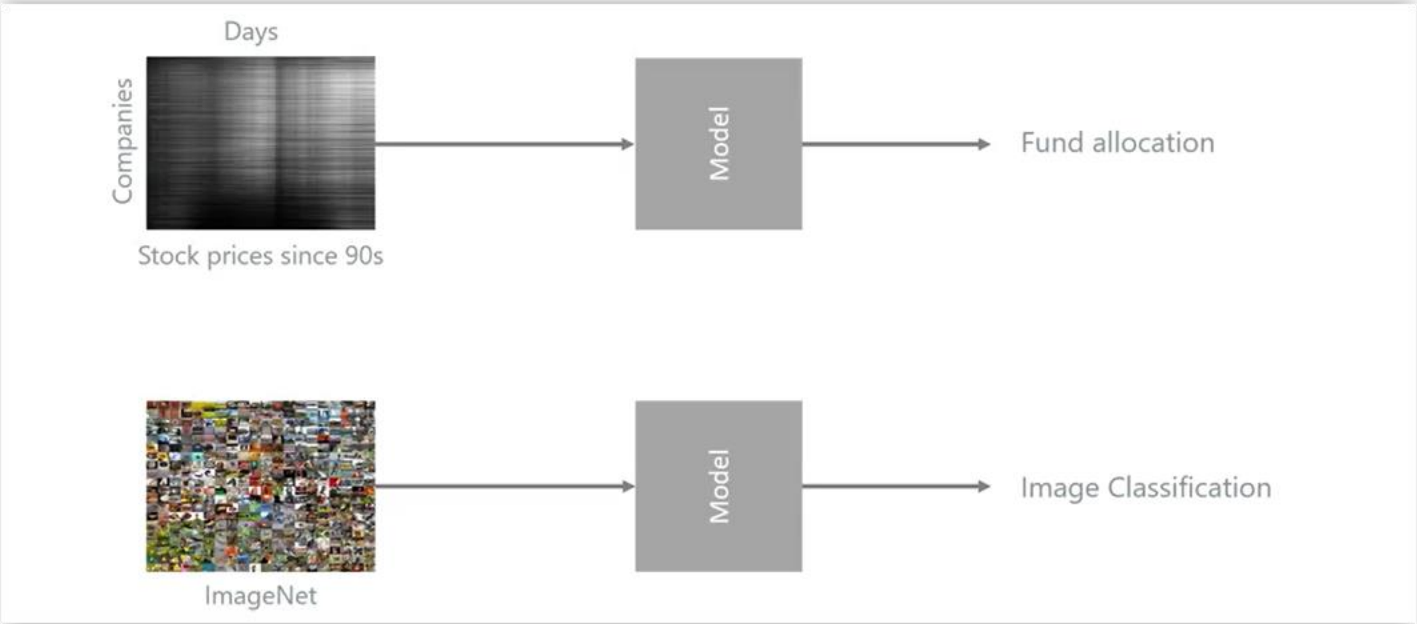


Visualization credit: https://distill.pub/2017/feature-visualization/

The closer we get to final layers, the more task-specific feature extractors become

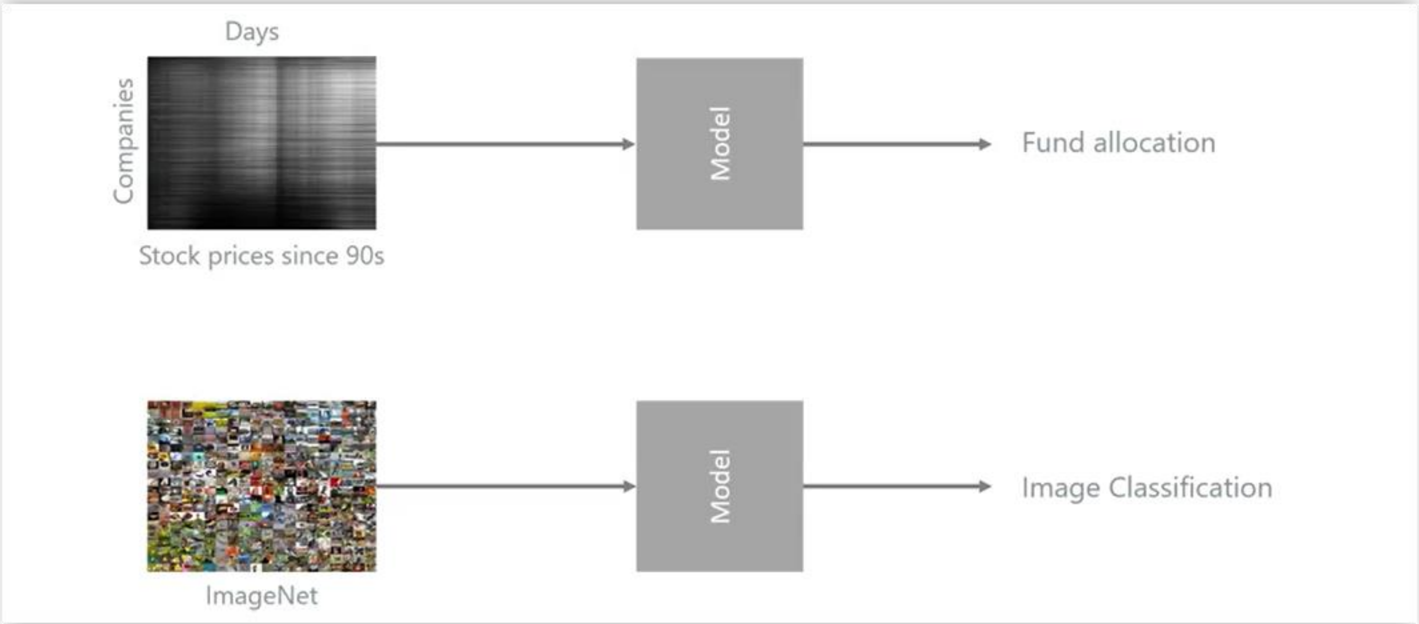
Limitations of Transfer Learning

- Data and tasks are vastly different
- Architectures of initial and target tasks vastly different



Limitations of Transfer Learning

- Data and tasks are vastly different
- Architectures of initial and target tasks vastly different



Remember:

- Transfer Learning is the fastest and easiest way to build a deep learning model without worrying about how much data you have

Remember:

- Transfer Learning is the fastest and easiest way to build a deep learning model without worrying about how much data you have