Introduction to Deep Learning

DO VAN NGUYEN

Many Contents From:

Fei-Fei Li, Justin Johnson & Serena Yeung: Stanford Course on "CNN for Visual Regconition",
Nando de Freitas, Scott Reed & Oriol Vinyals: NIPS2017 Tutorial on"Deep Learning: Practice and Trends".

Parts of presentation

Neural Network

- Linear Classification and Perceptron
- Model optimization
- Multiple Perceptron: Neural network
- Training with gradient descent

Deep Learning: Architectures and Learning Methods

- Supervised Learning with Convolutional Neural Network
- Sequence Learning with Recurrent Neural Network
- Unsupervised learning - Generative Models
- Reinforcement learning

Neural Network

IMAGENET

www.image-net.org

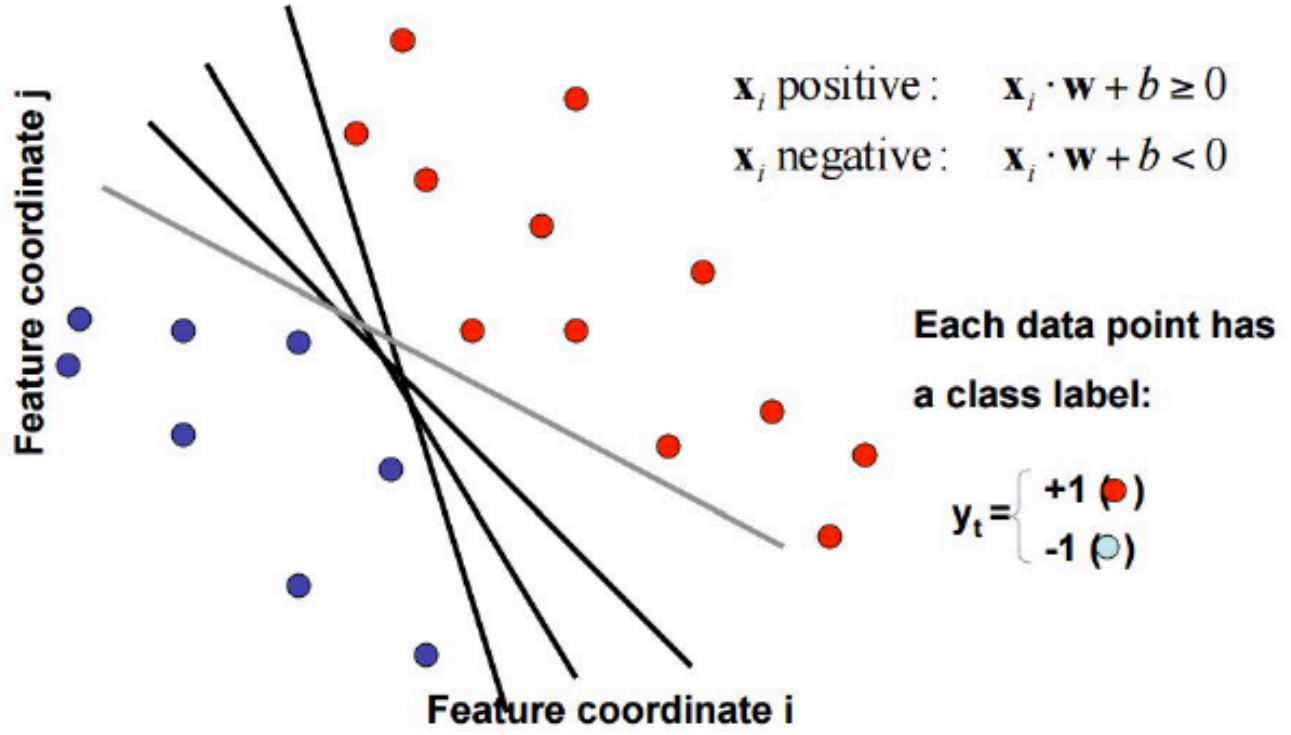
22K categories and 14M images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities

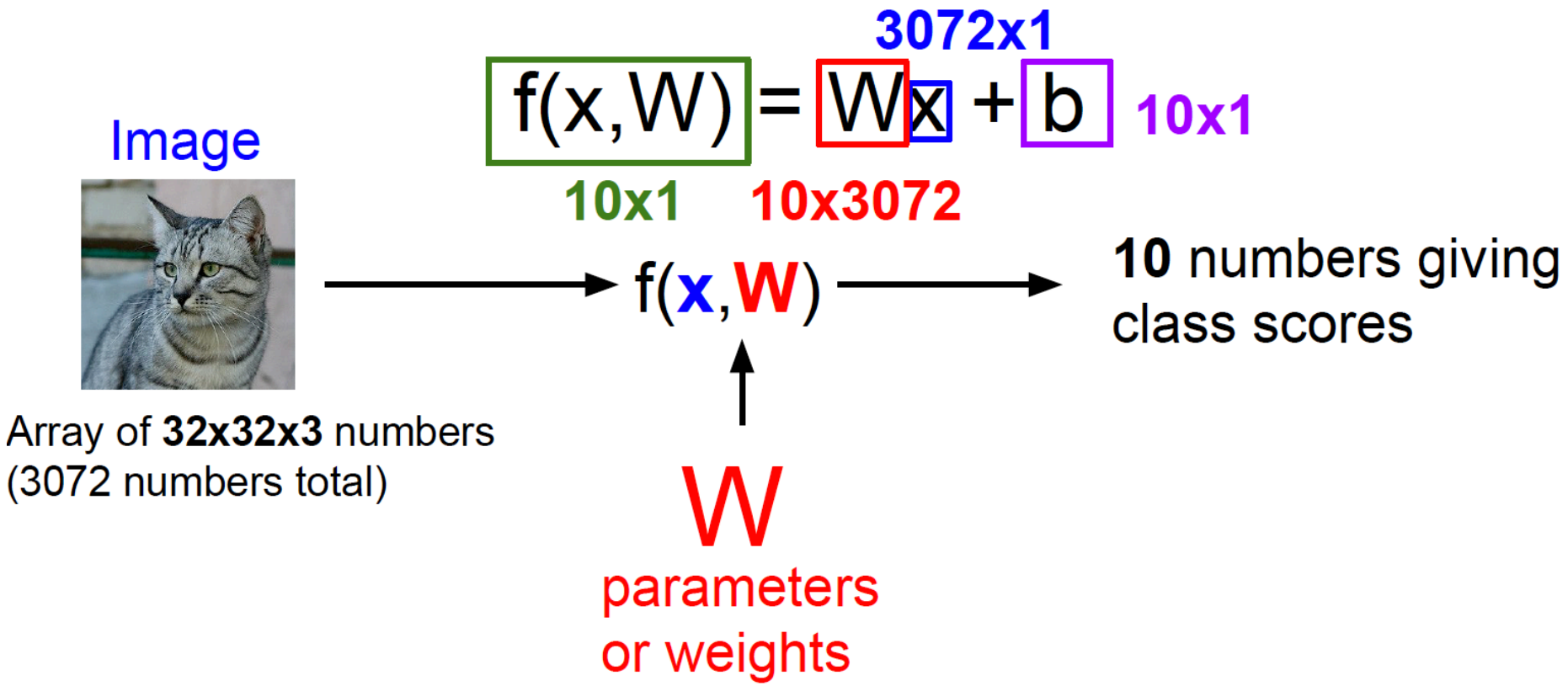
Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

Linear Classification

- Find linear expression (*hyperplane*) to separate positive and negative examples

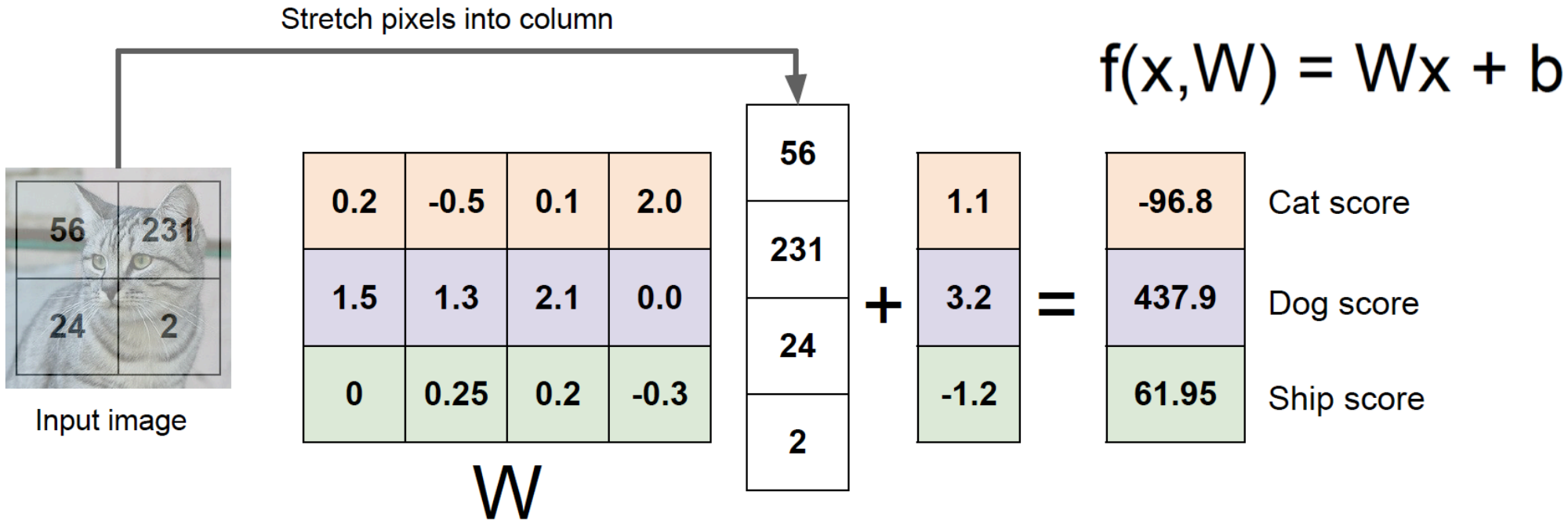


Linear Classification

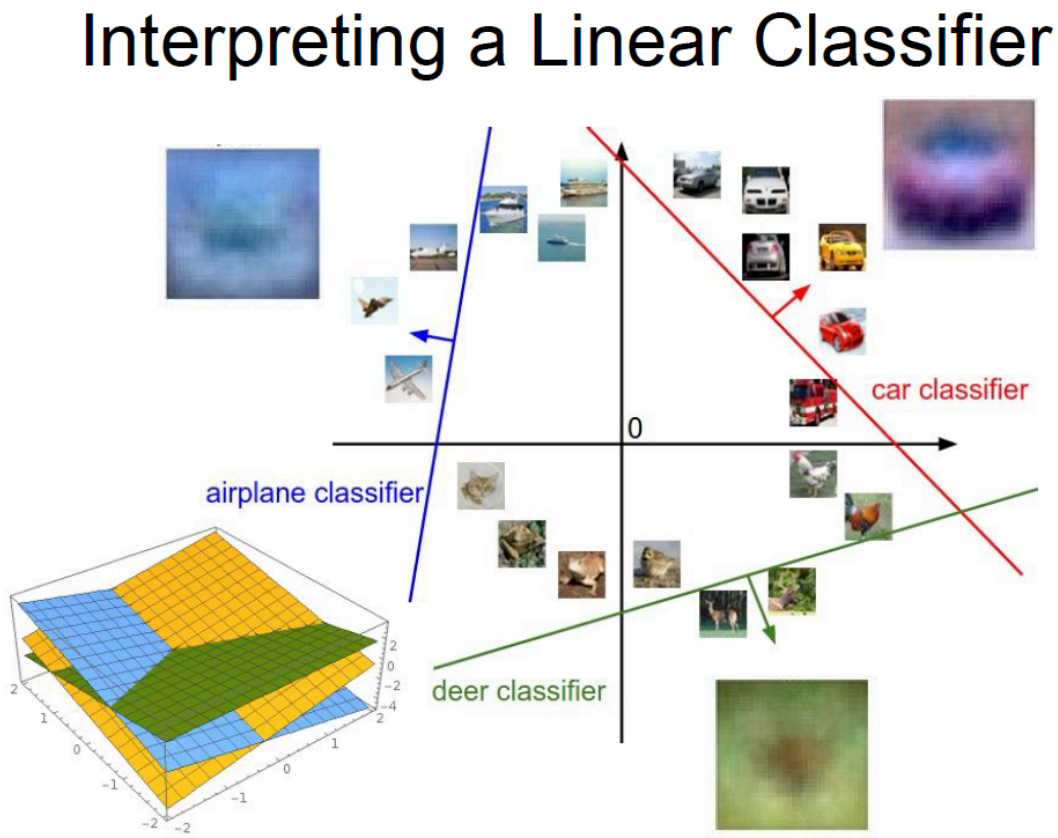


Linear Classification

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Linear Classification



Plot created using Wolfram Cloud

$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

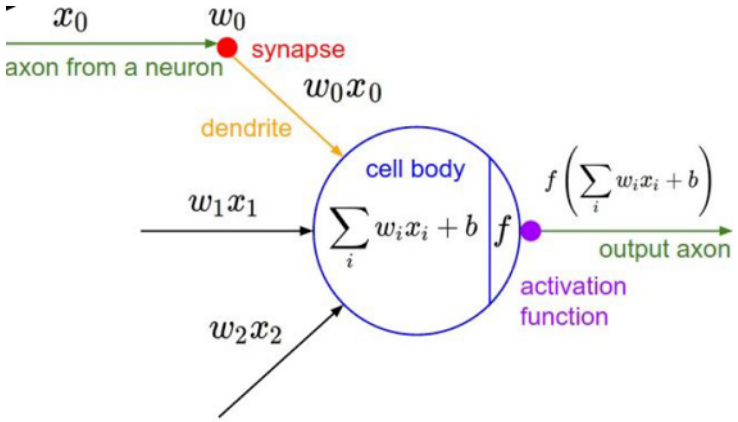
Neural Network

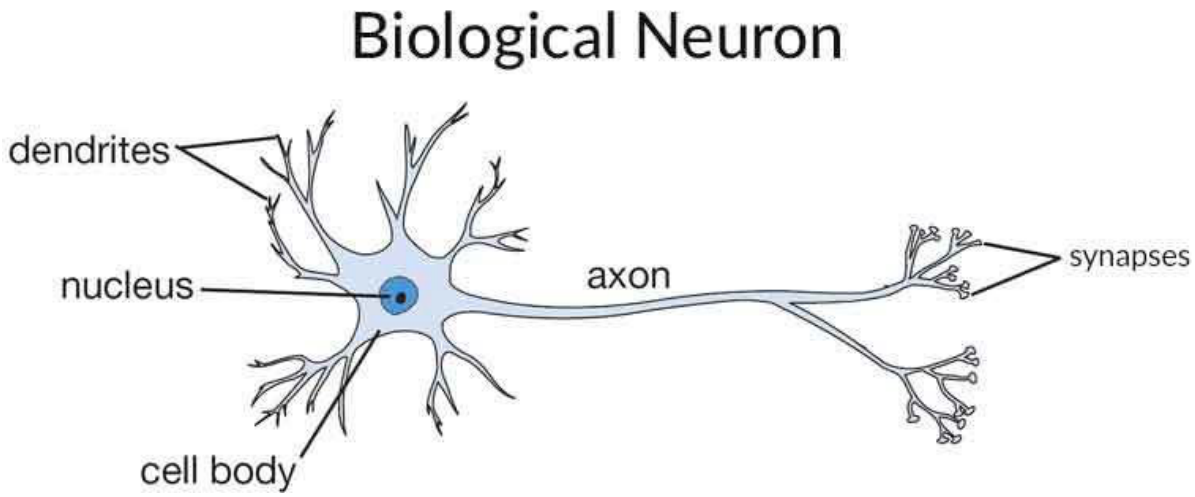
Linear classification

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Output = sign(Wx+b)

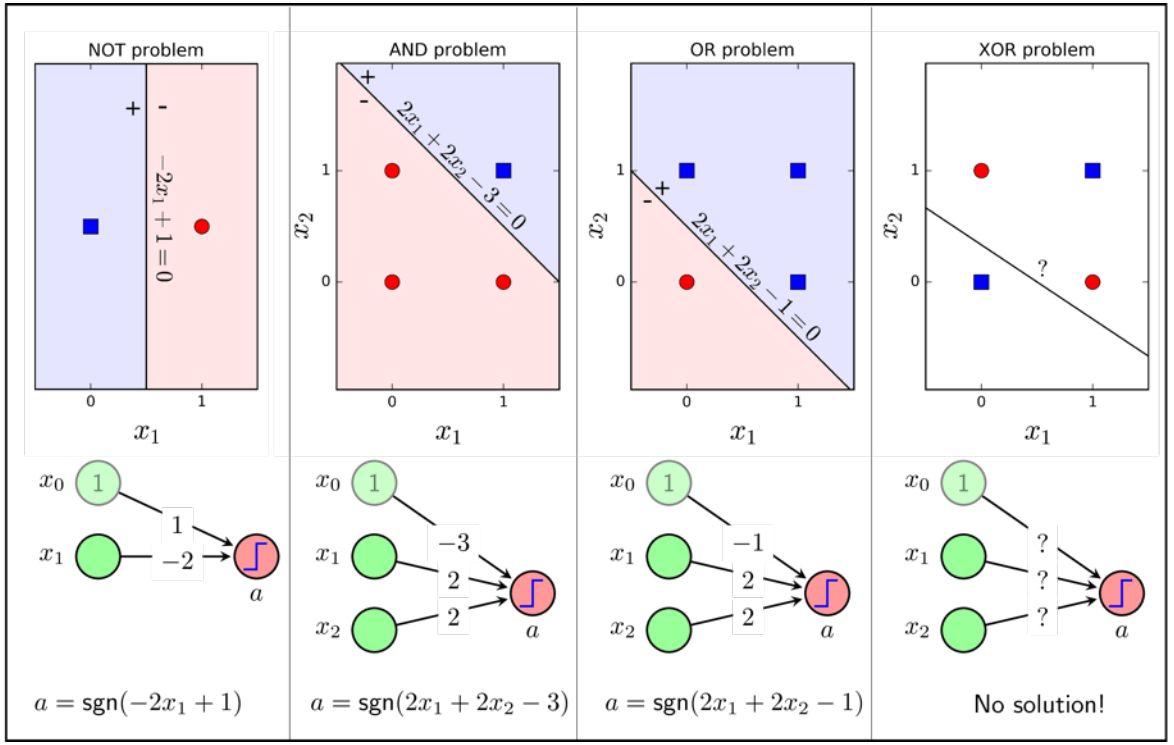
Perceptron





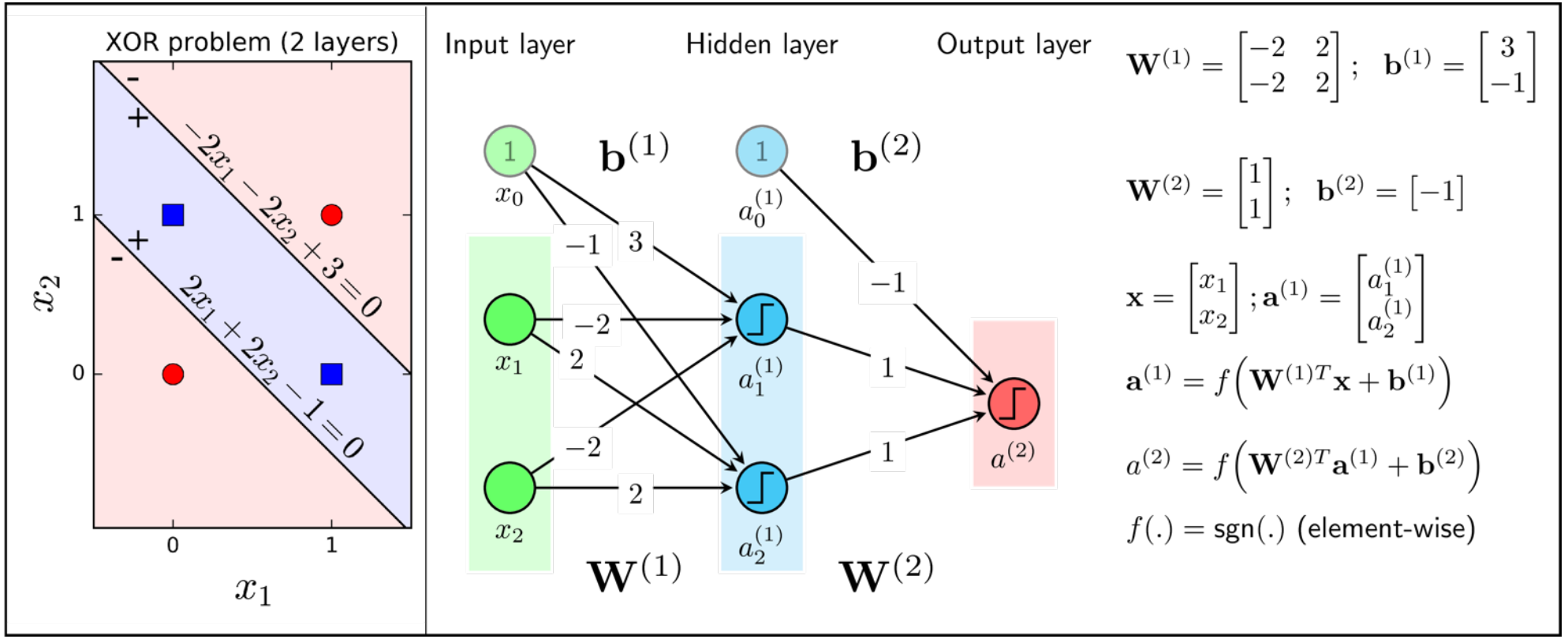
Neural Network

- Linear Classification problem
  - Some data points can be separated by lines
  - XOR problem: non-linearly separable



Neural Network

Multiple perceptron



$$\mathbf{W}^{(1)} = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix}; \quad \mathbf{b}^{(1)} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}; \quad \mathbf{b}^{(2)} = [-1]$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}; \mathbf{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \end{bmatrix}$$

$$\mathbf{a}^{(1)} = f\left(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

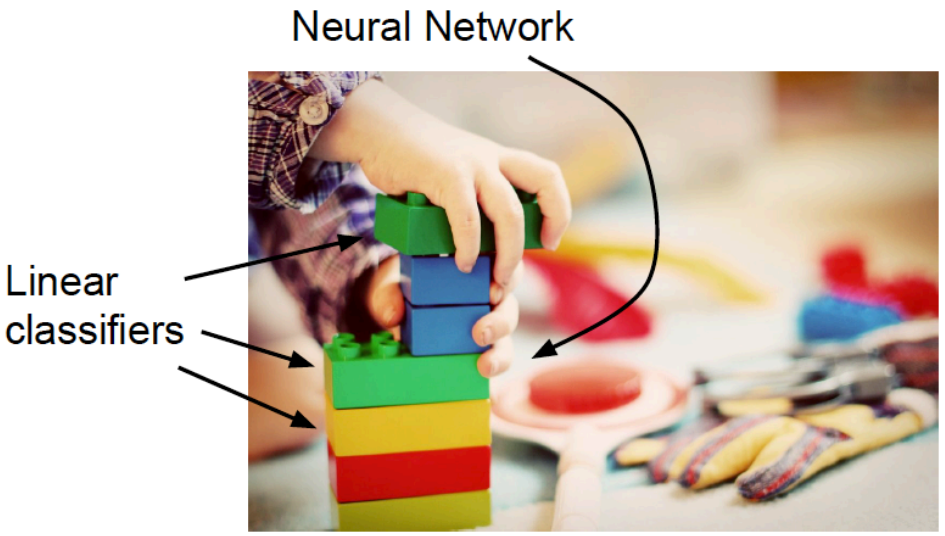
$$a^{(2)} = f\left(\mathbf{W}^{(2)T}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}\right)$$

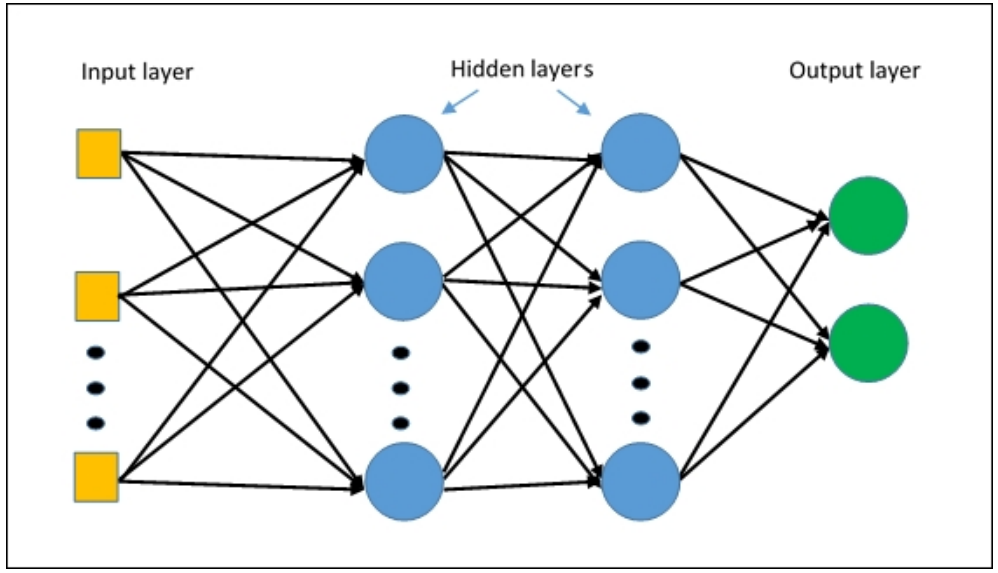
$f(.) = \text{sgn}(.)$ (element-wise)

Neural Network

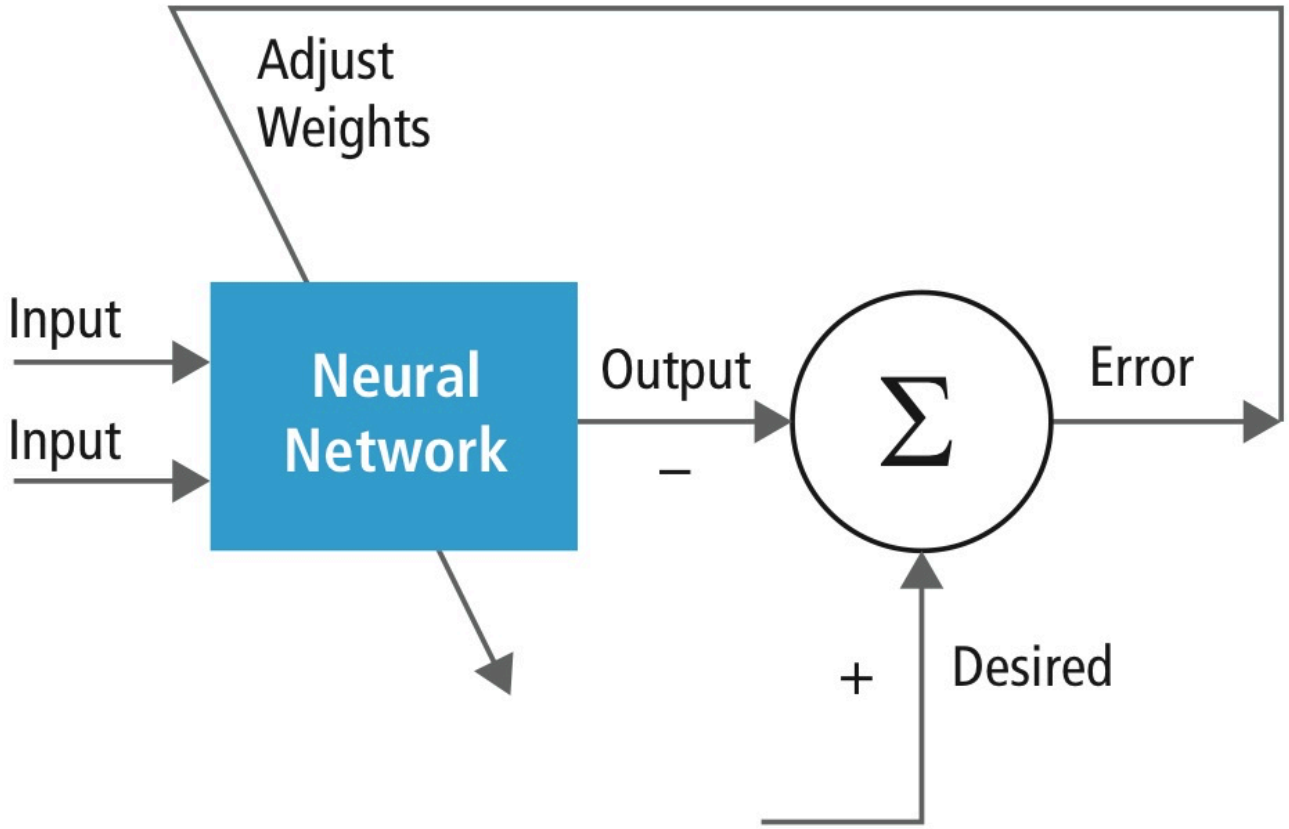
Multi-layers Perceptron

- Layers: Bricks with many types
- Network design: Construction and connection

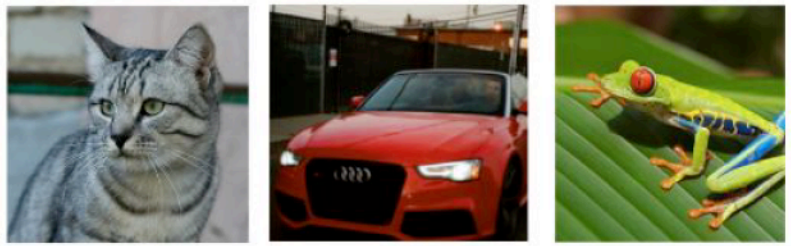




Training Pipeline



Training objective



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

Loss Function

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

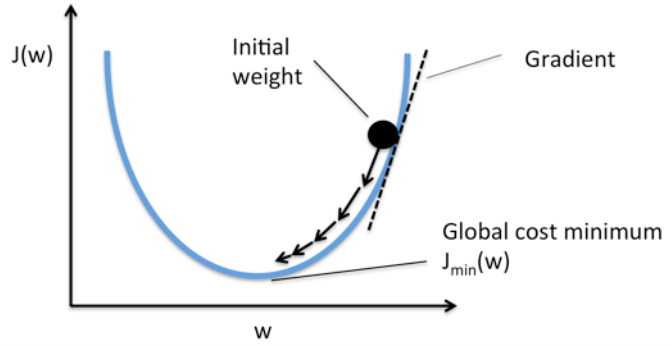
$$\{(x_i, y_i)\}_{i=1}^{N}$$

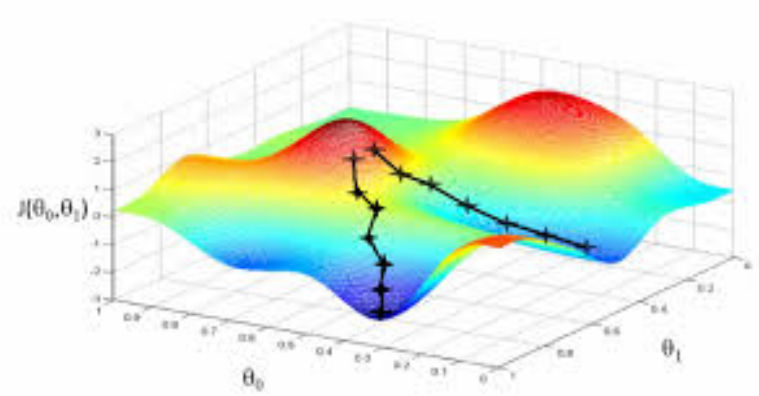
Where x_i is image and y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Gradient Descent





GRADIENT-DESCENT($training\_examples, \eta$)

Initialize each w_i to some small random value

Until the termination condition is met, Do

- Initialize each Δw_i to zero.
- For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
  - Input instance $\vec{x}$ to unit and compute output o
  - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

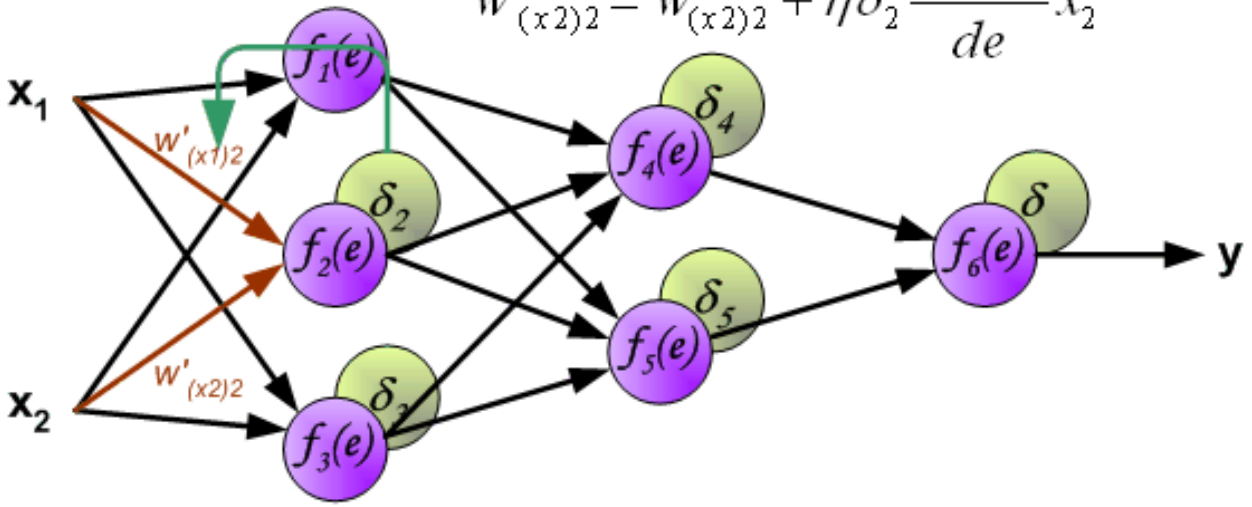
- For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

Back Propagation

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



Deep Learning

NETWORK STRUCTURES AND LEARNING METHODS

Convolutional Neural Network

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

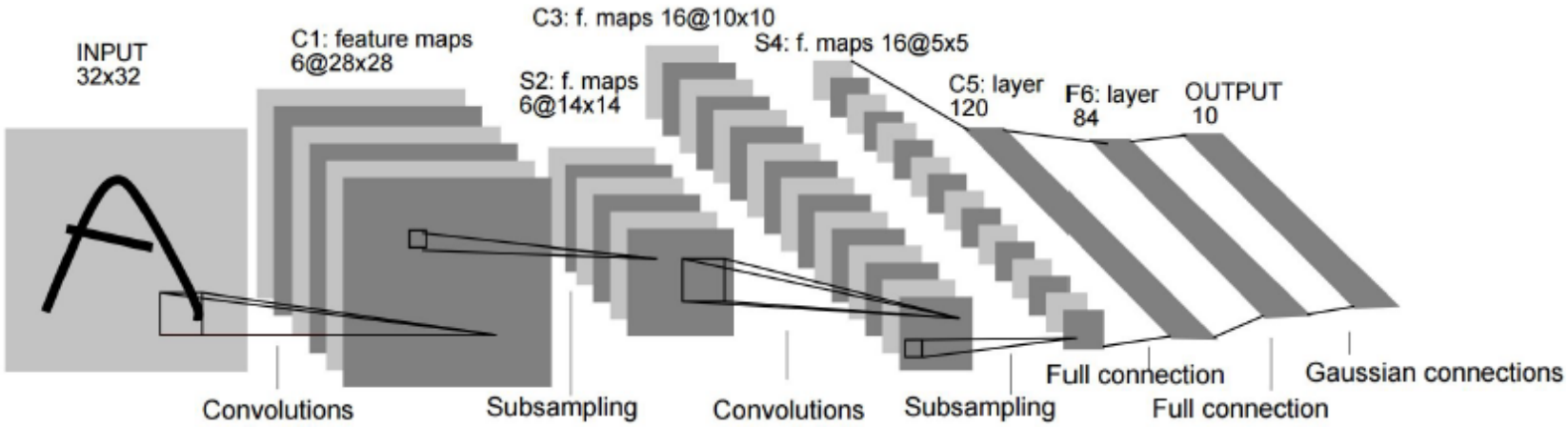
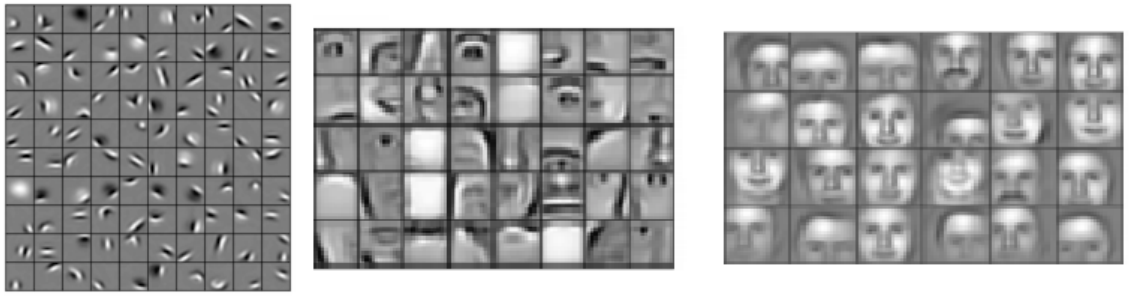


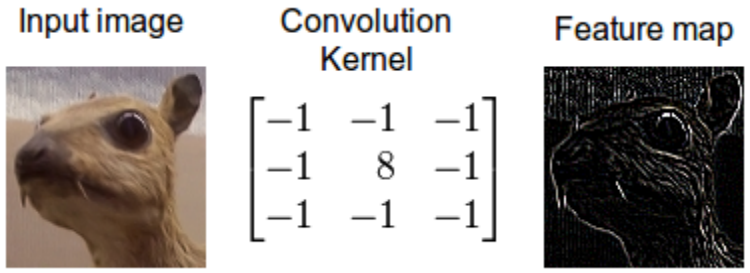
Image Classification Challenge

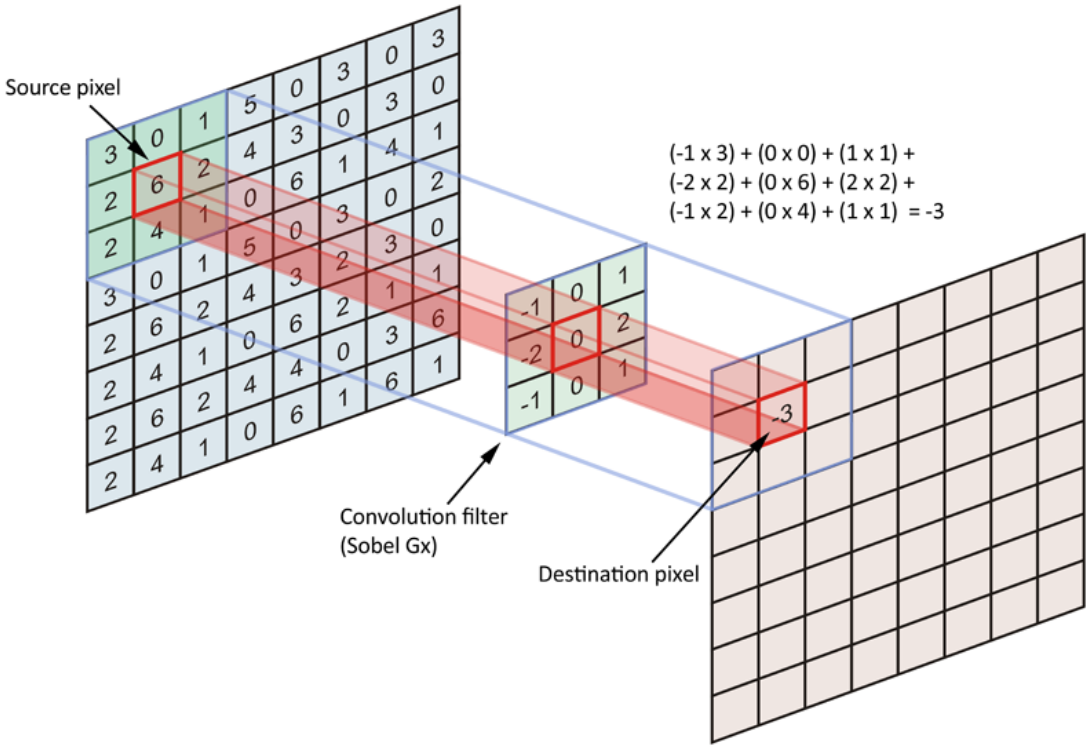
- Locality: objects tend to have a local spatial support
- Translation invariance: object appearance is independent of location
  - Can define these properties since an image lies on a grid/lattice
    - ConvNet machinery applicable to other data with such properties, e.g. audio/text
- Recognition: Supporting from hidden feature representation



Locality

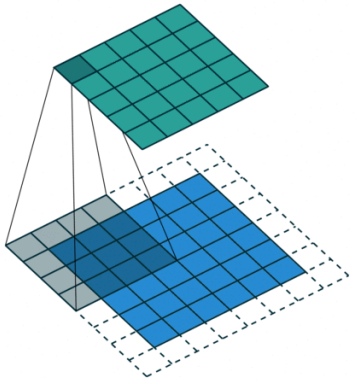
- Make fully-connected layer locally-connected
- Each unit/neuron is connected to a local rectangular area – receptive field
- Different units connected to different locations
  - output ("feature map") lies on a grid itself

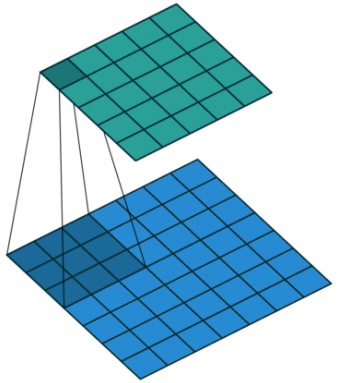


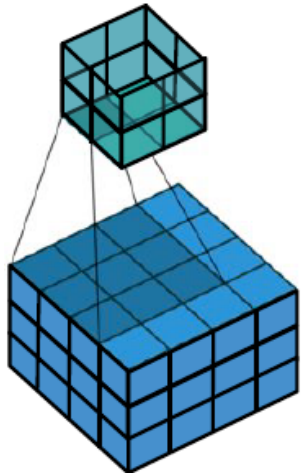


Translation Invariance

- Weight sharing
  - units connected to different locations have the same weights
  - equivalently, each unit is applied to all locations
- Convolutional layer – locally-connected layer with weight sharing (translation invariance)
- The weights are invariant, the output is equivariant
- Convolutional layer input and output can have multiple channels







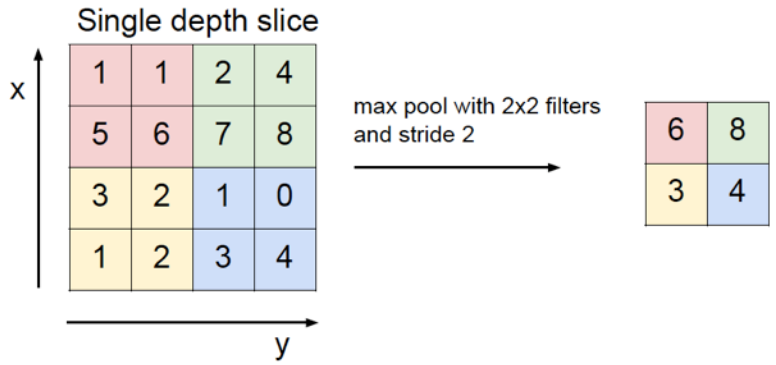
feature maps are 3-D tensors
height × width × channels

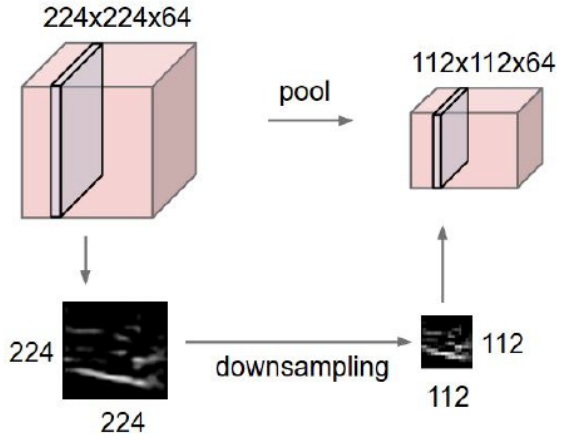
Recognition from hidden features

Pooling layer:

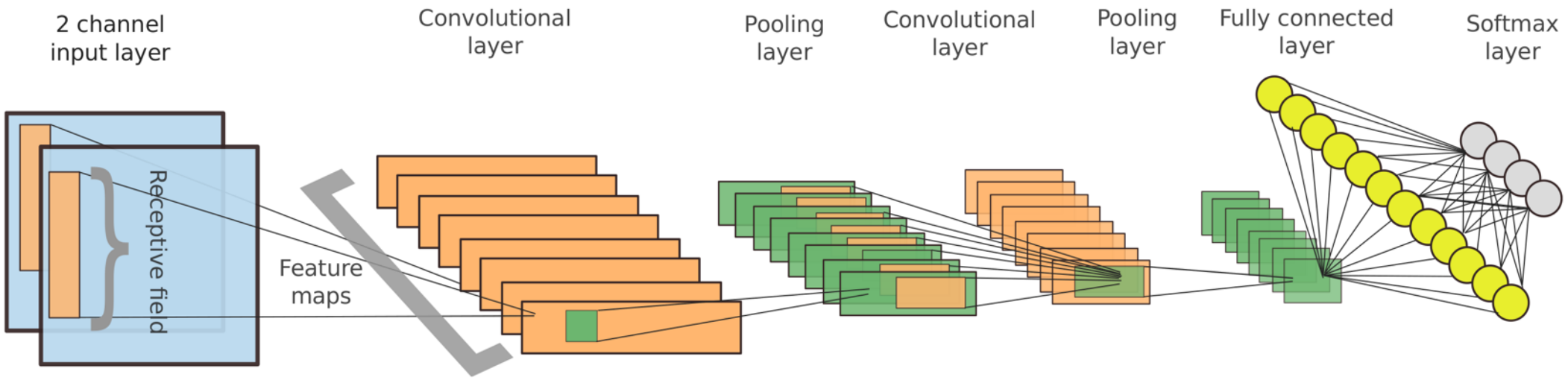
- makes the representations smaller and more manageable
- operates over each activation map independently

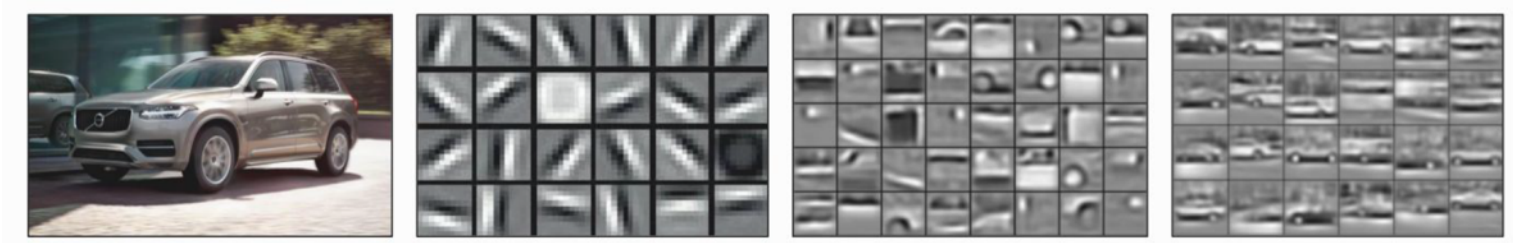
Max pooling:



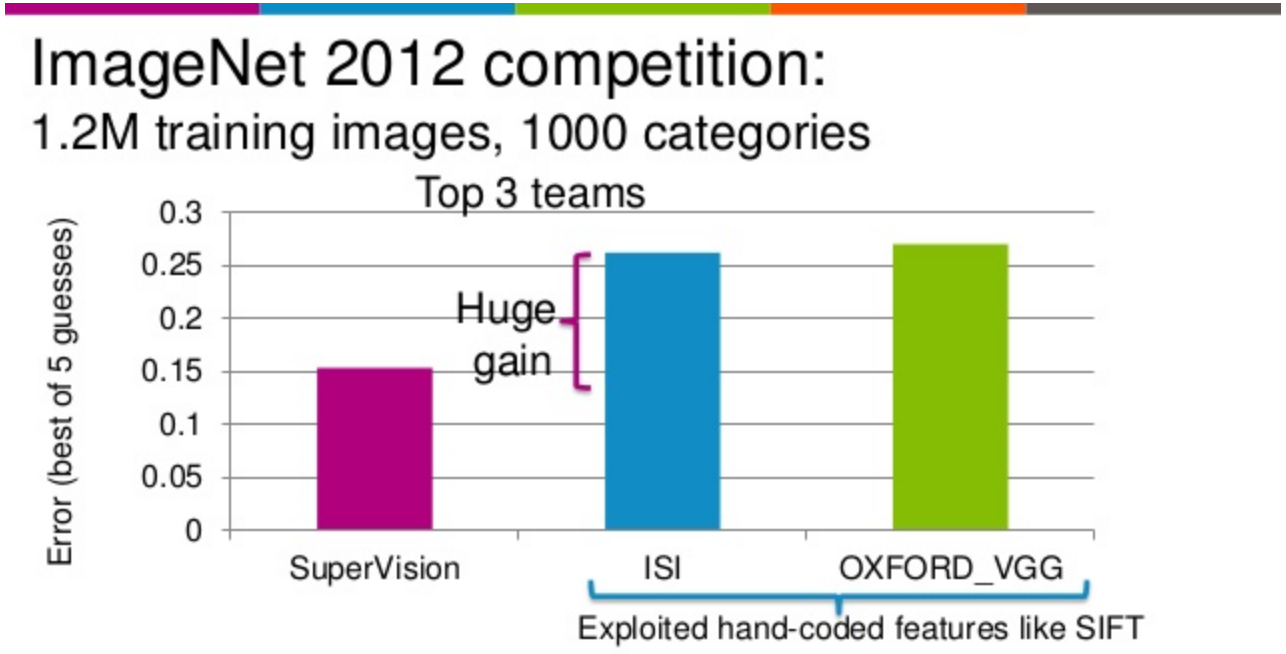


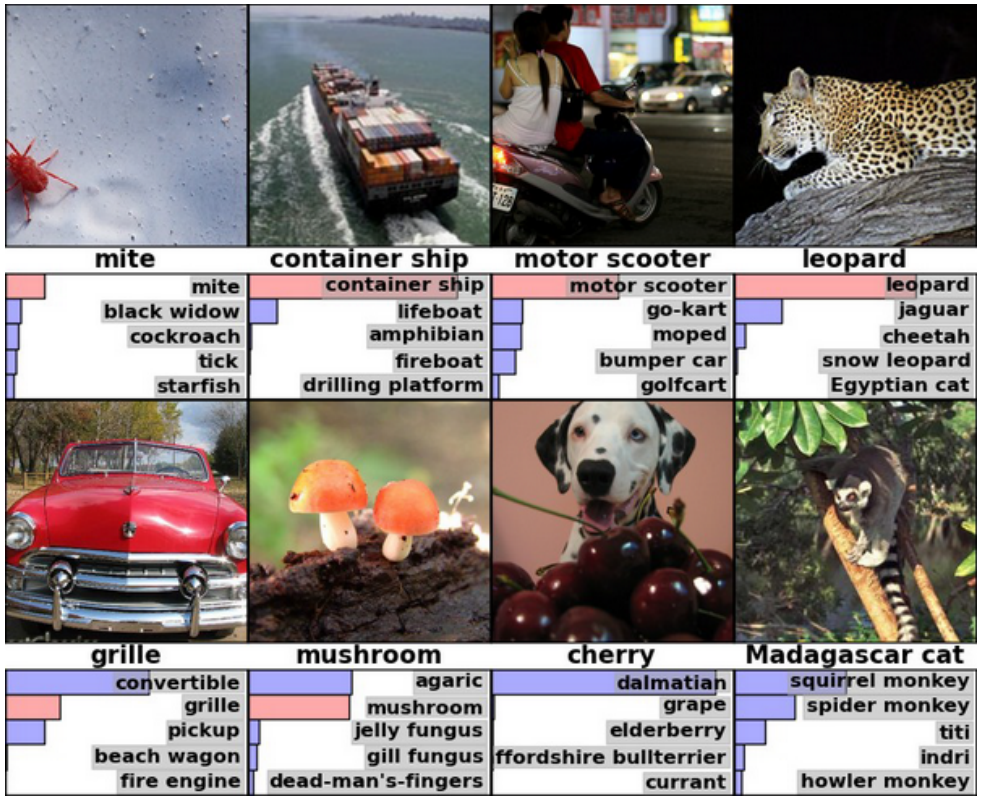
Typical structure for image classification





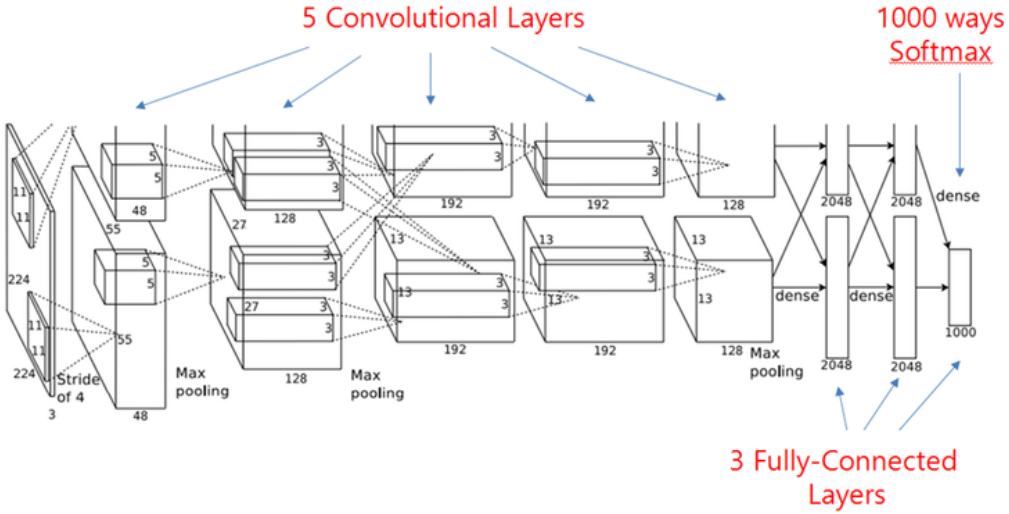
Object Recognition Show





Case Study: AlexNet

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)



[Krizhevsky et al. 2012]

Case Study: VGGNet

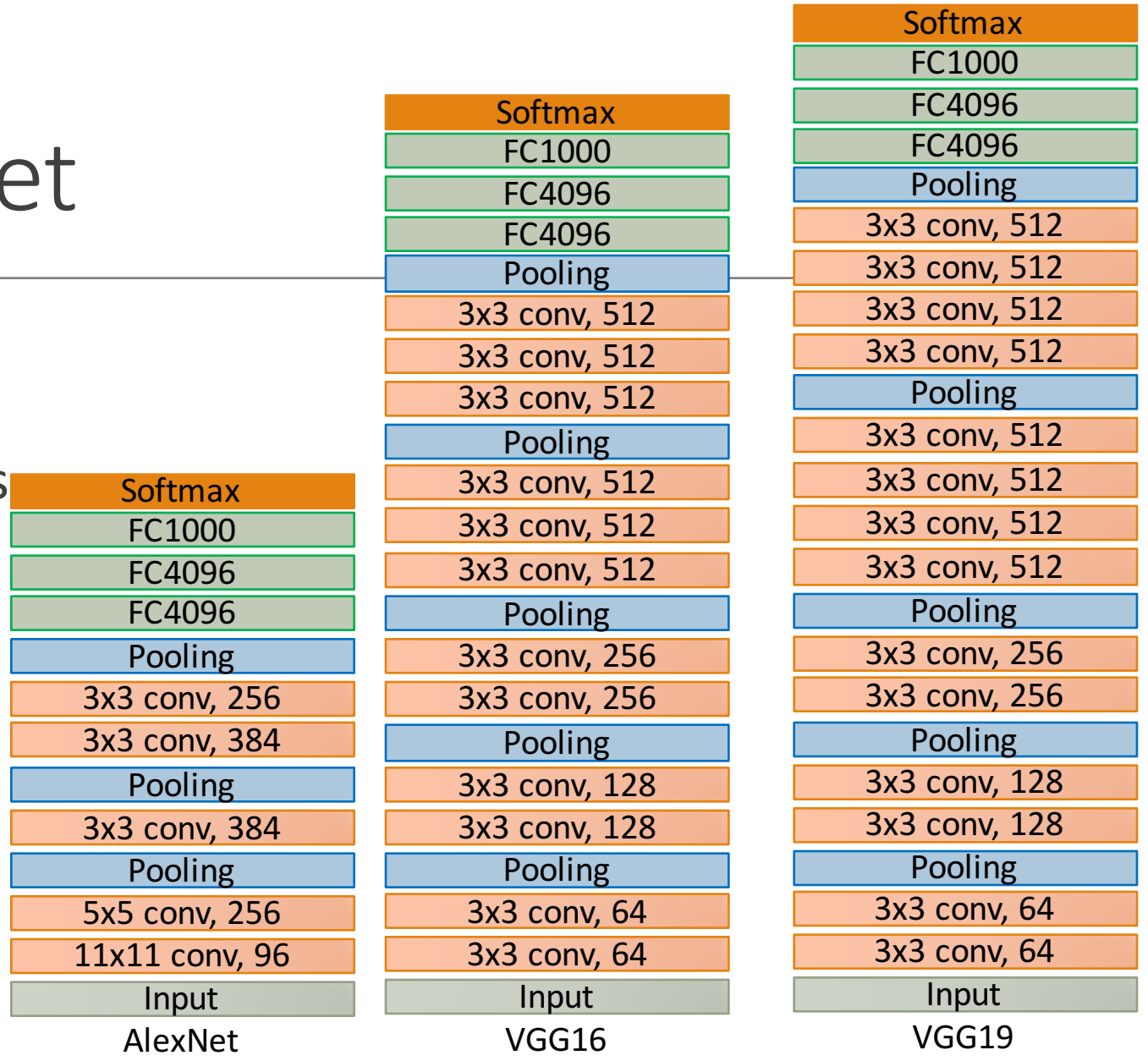
Small filters, Deeper networks
[Simonyan and Zisserman, 2014]

8 layers (AlexNet) -> 16 - 19 layers (VGG16Net)

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

Deeper, more non-linearities

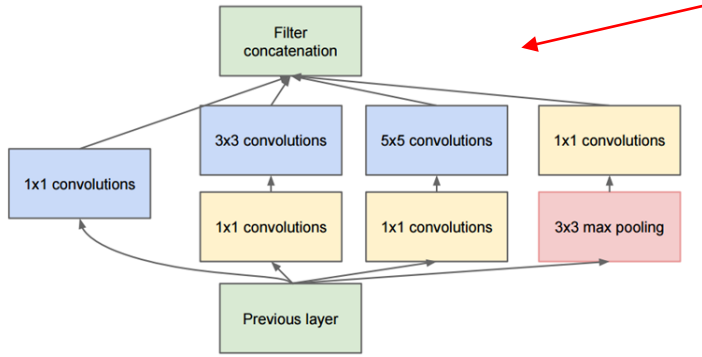
And fewer parameters



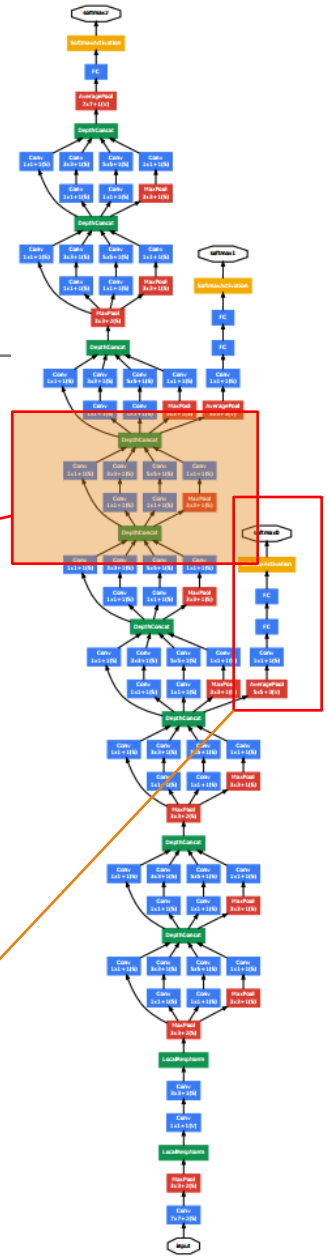
Case Study: GoogleNet *[Szegedy et al., 2014]*

Inception Module: Network in Network

- Apply parallel filter operations on the input from previous layer:
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise
- Auxiliary Classifier: Auxiliary classification outputs to inject additional gradient at lower layers



Inception

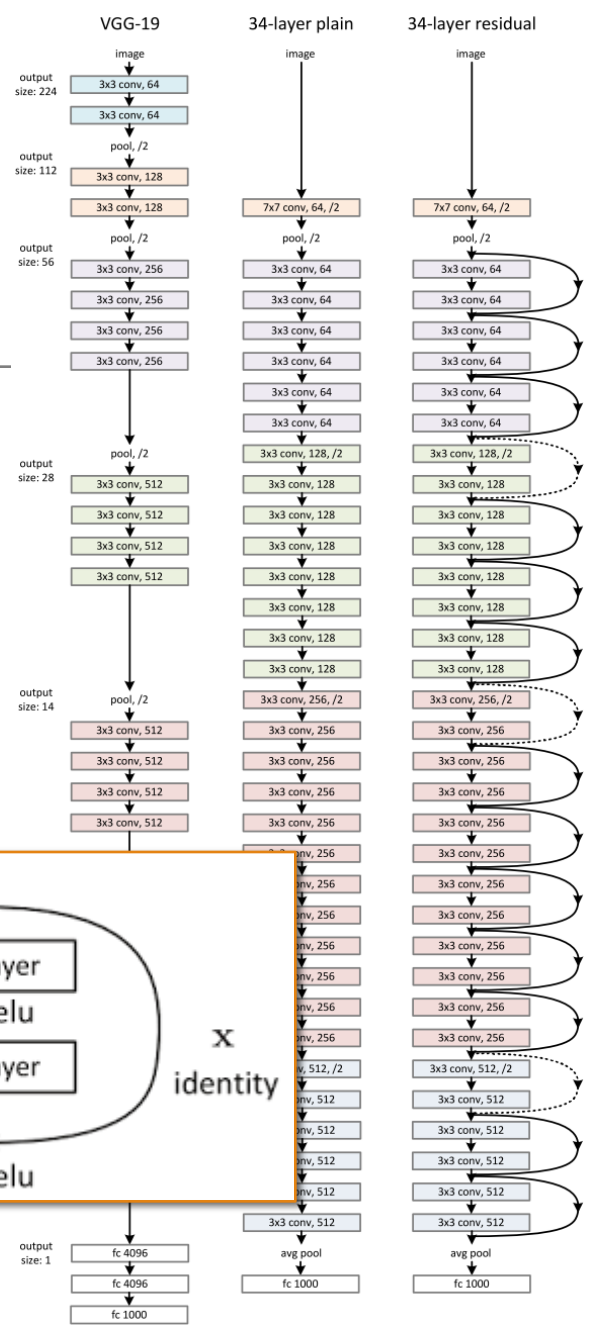


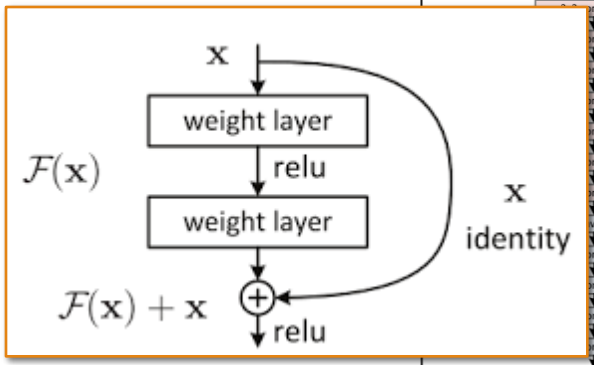
Auxiliary Classifier

Case Study: ResNet

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions inILSVRC'15 and COCO'15!
- Optimization problem: The deeper model should be able to perform at least as well as the shallower model.
  - Many approaches to deal



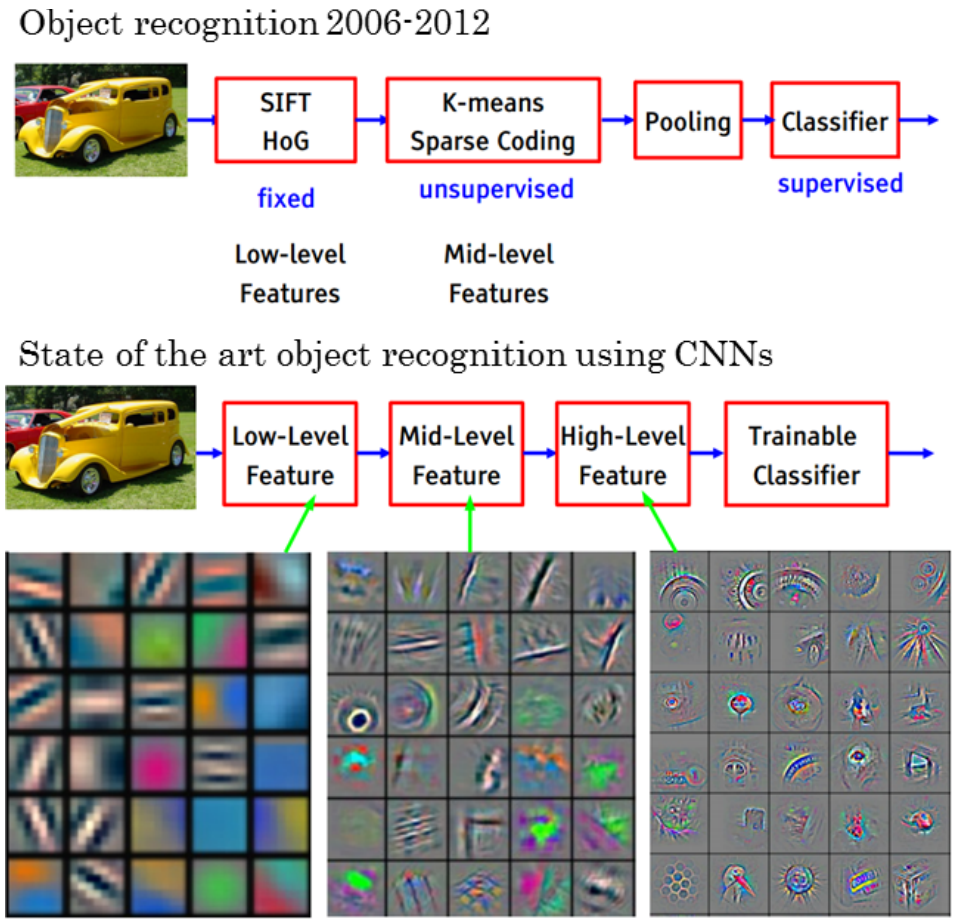


Why Deep Learning

2010-11: hand-crafted computer vision pipelines

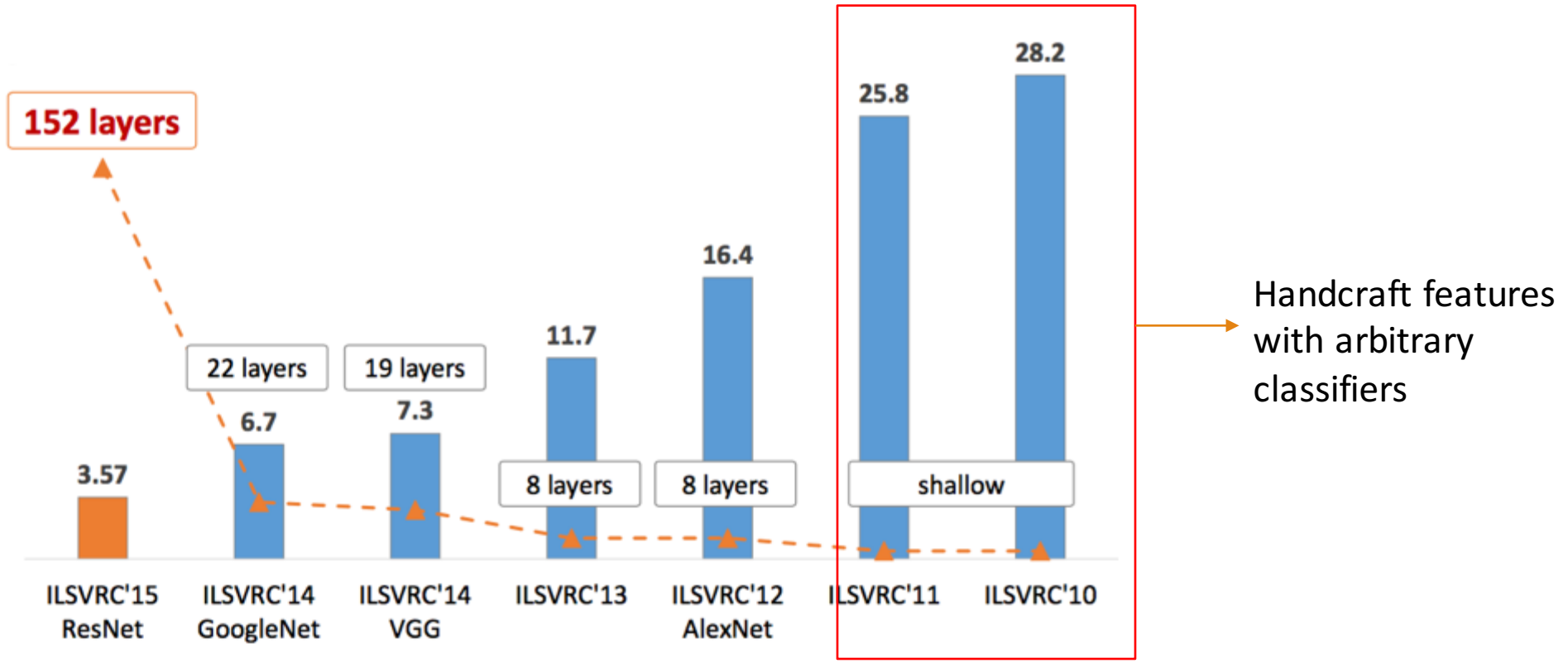
2012-2016: ConvNets, now tend to feature learning

- 2012: AlexNet : major deep learning success
- 2013: ZFNet: improvements over AlexNet
- 2014:
  - VGGNet: deeper, simpler
  - InceptionNet: deeper, faster
- 2015: ResNet: even deeper
- 2016: ensembled networks
- 2017: Squeeze and Excitation Network



Convolutional Network in Competitions

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

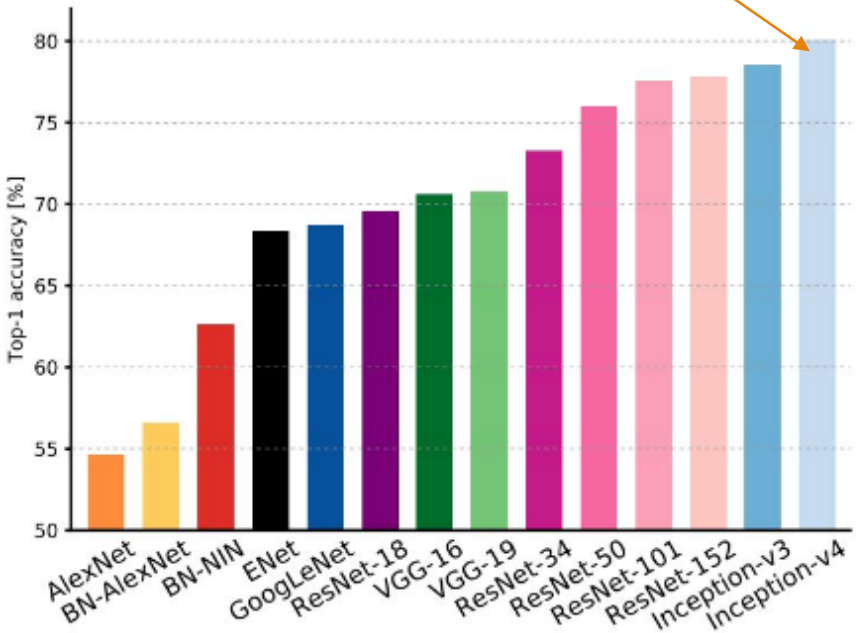


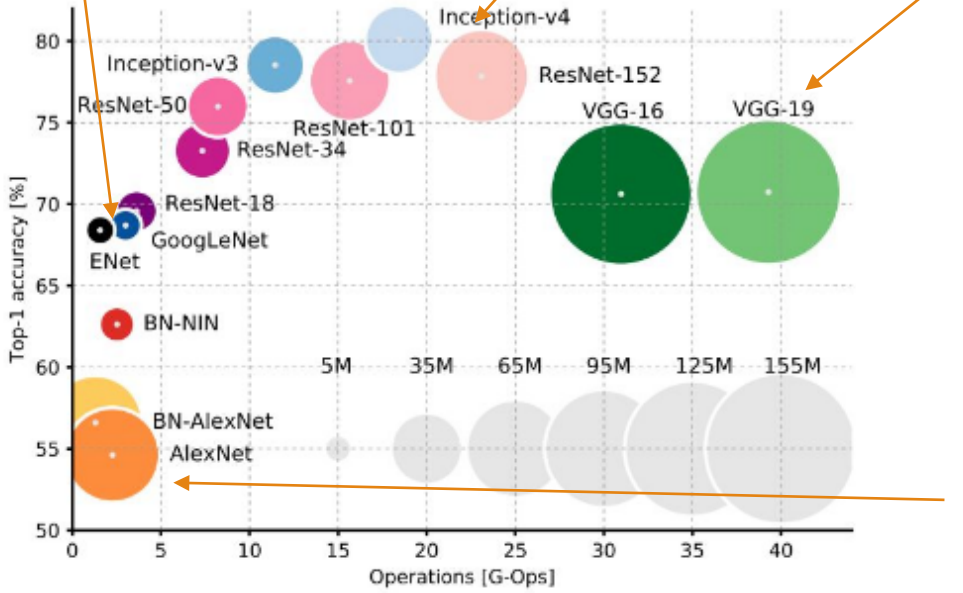
IncepV4:ResNet+Inception

GoogleNet: effective

ResNet: Moderate efficiency depending on model, highest accuracy

VGG: Highest memory, most operations

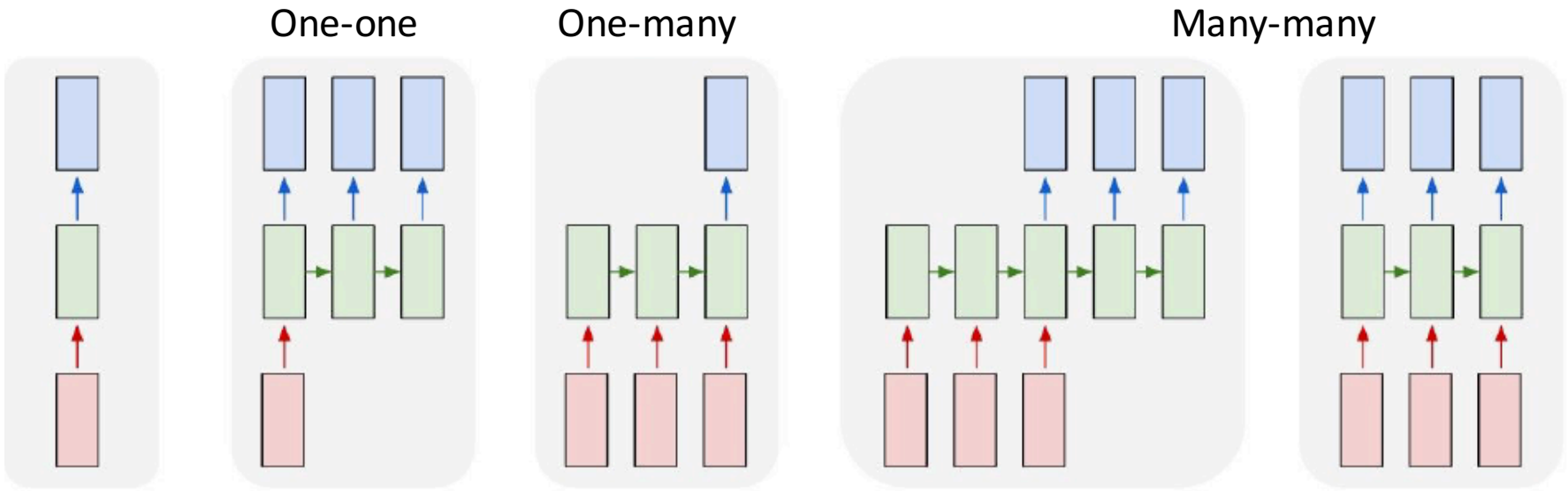




AlexNet: Smaller compute, still memory heavy, lower accuracy

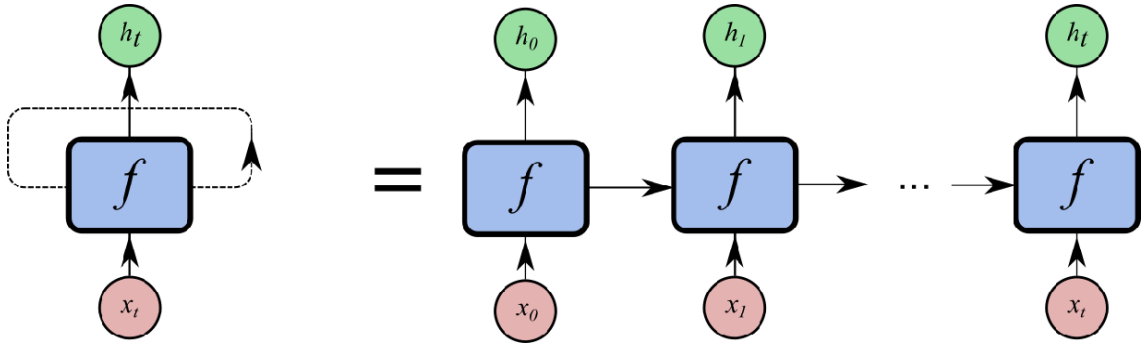
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Sequence Learning



Recurrent Neural Network

- Usually want to predict a vector at some time steps
- Process a sequence of vector x by applying a recurrence formula at every step



$$h_t = f_W(h_{t-1}, x_t)$$

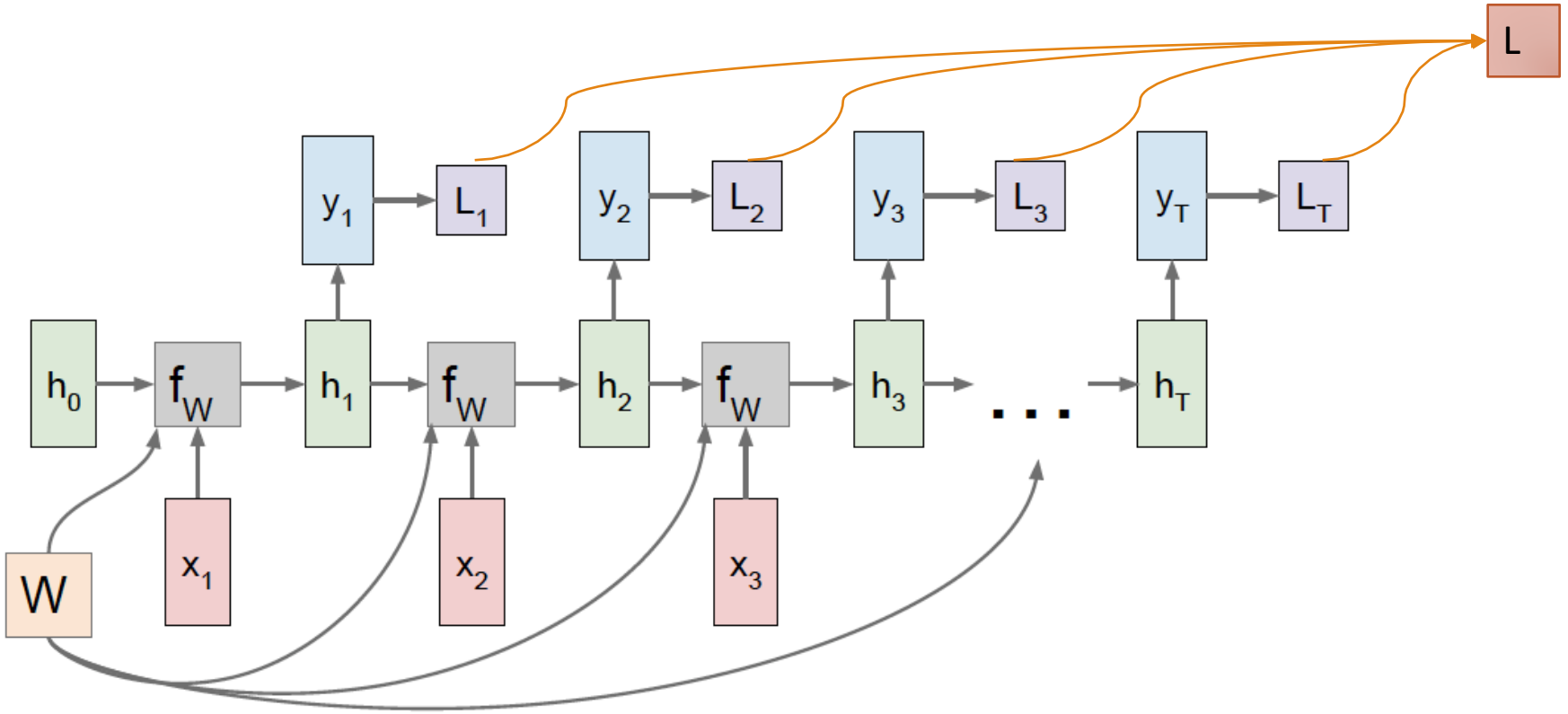
new state

some function with parameters W

old state

input vector at some time step

RNN: Computational Graph

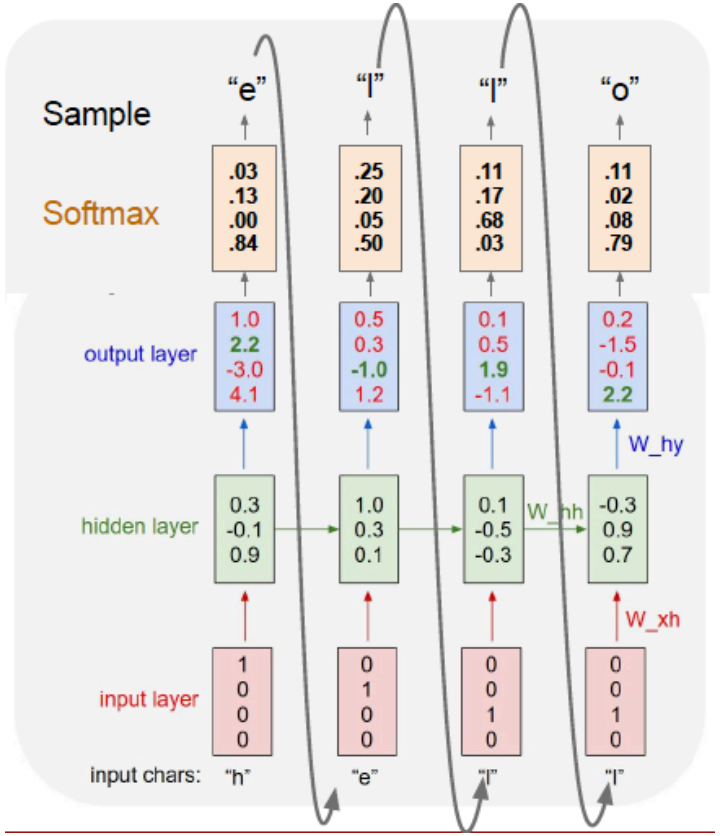


RNN: Character-level

Character level Language Sampling

Vocabulary: [h,e,l,o]

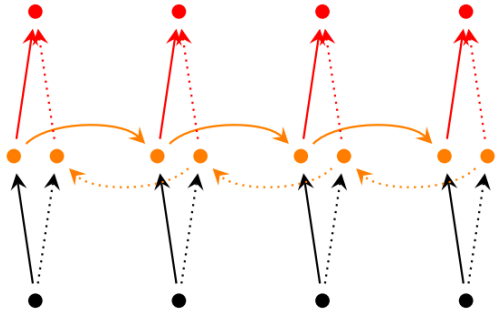
At least-time sample characters one at a time feed back to the model

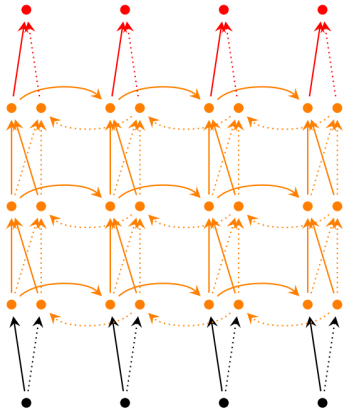


More RNN Structures

Bidirectional RNNs:

- Output at time t depend on.
  - the previous elements in the sequence.
  - future elements.

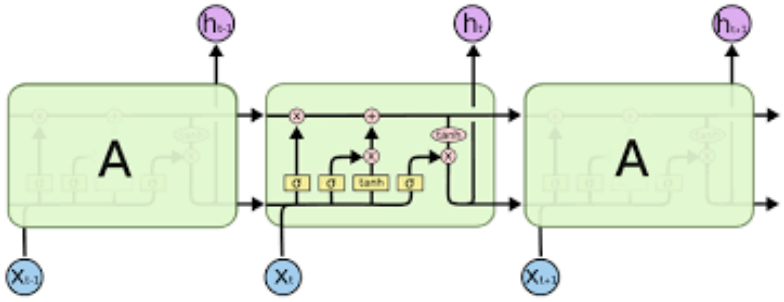




Deep (Bidirectional) RNNs:

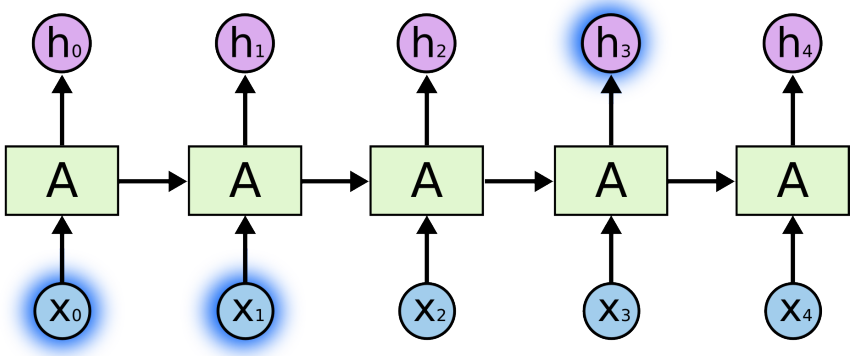
- Similar to Bidirectional RNNs.
- Have multiple layers per time step.

Long short-term memory (next slides)

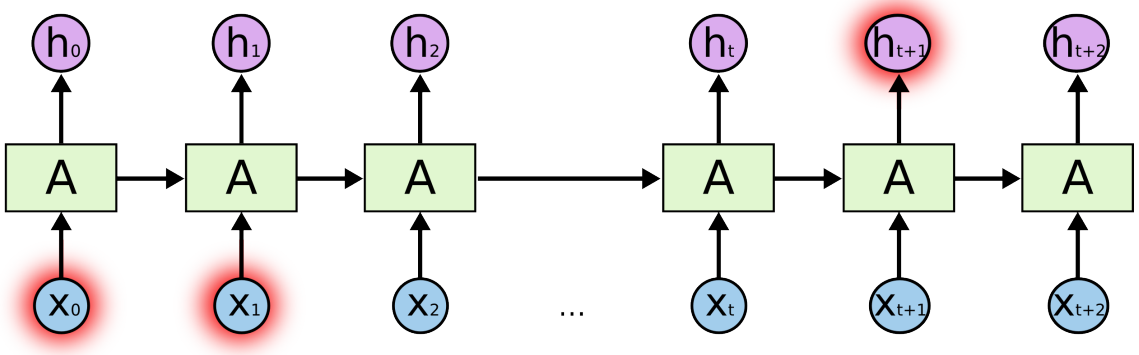


Long short-term Memory

Long-Term Dependencies Problem

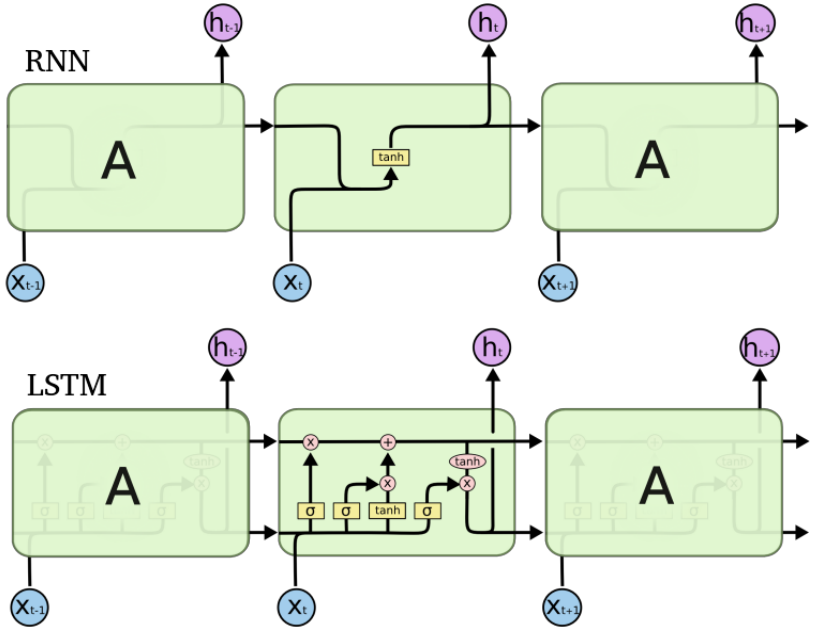


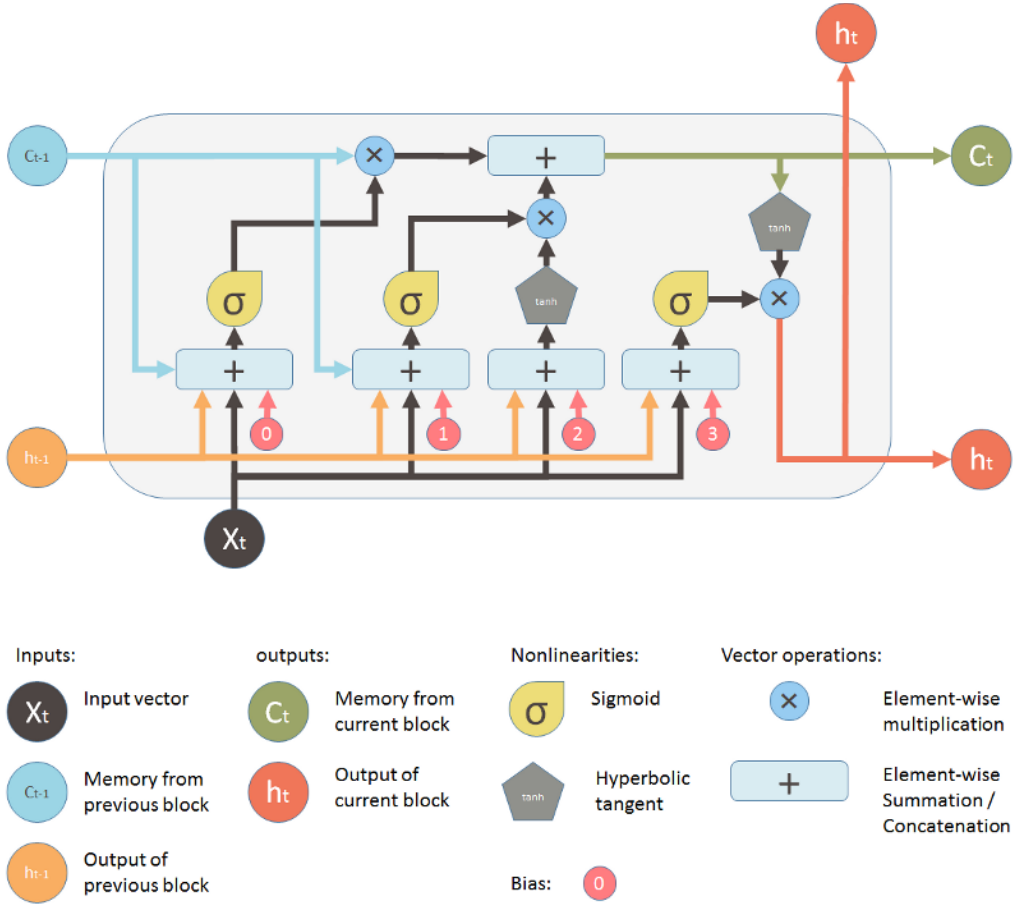
RNN looks at recent information



RNN unable to learn to connect long term information

Long short-term Memory

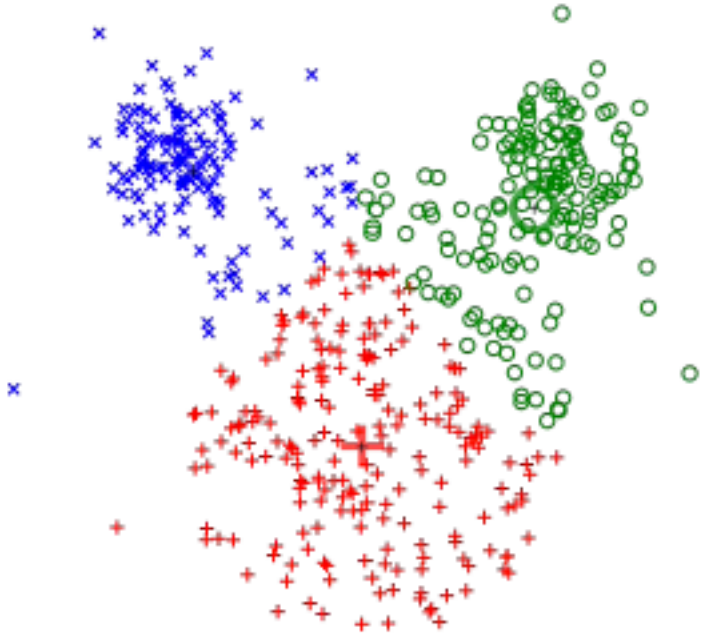




Generative Model

Unsupervised learning

- Data: x
  - Just data, no labels!
- Goal: Learn some underlying hidden structure of the data
- Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



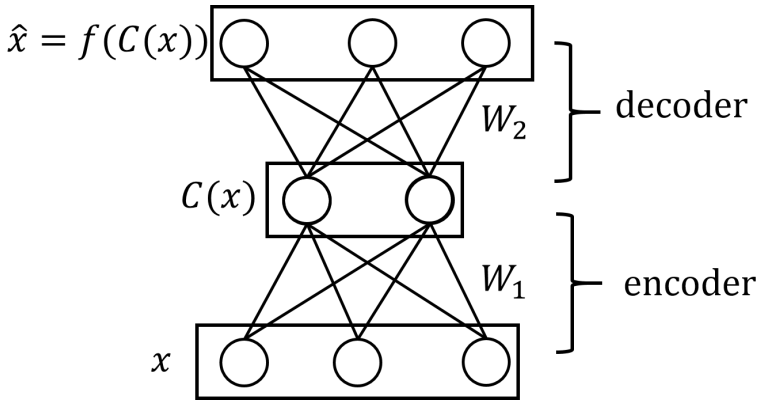
K-means clustering

Auto Encoder

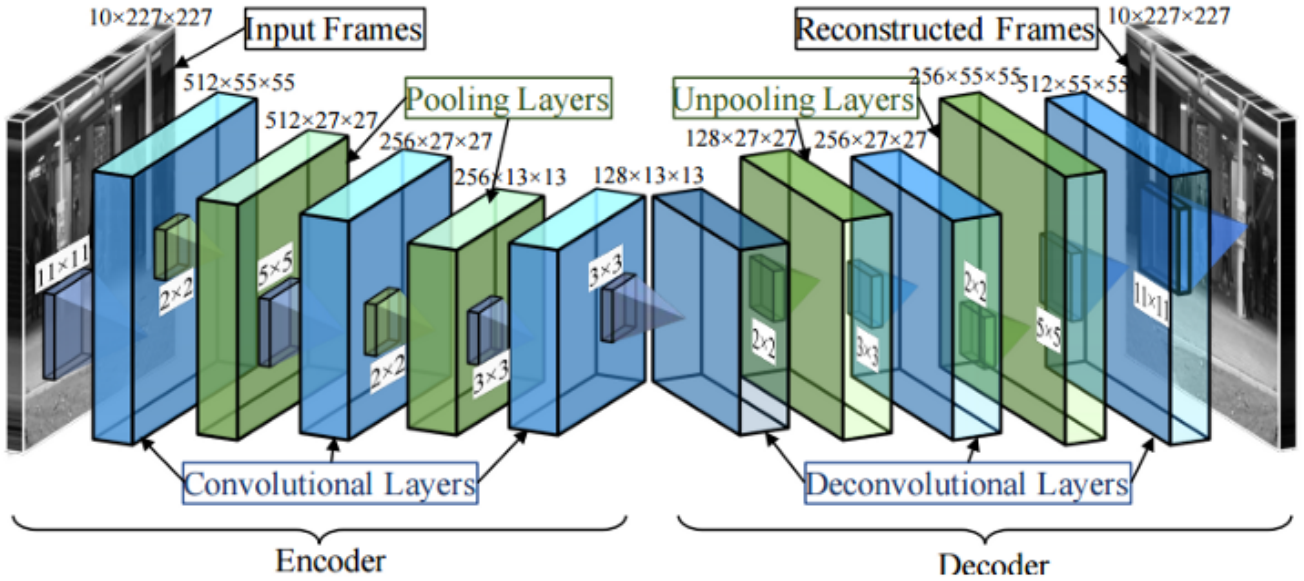
Unsupervised learning: no labels for training data

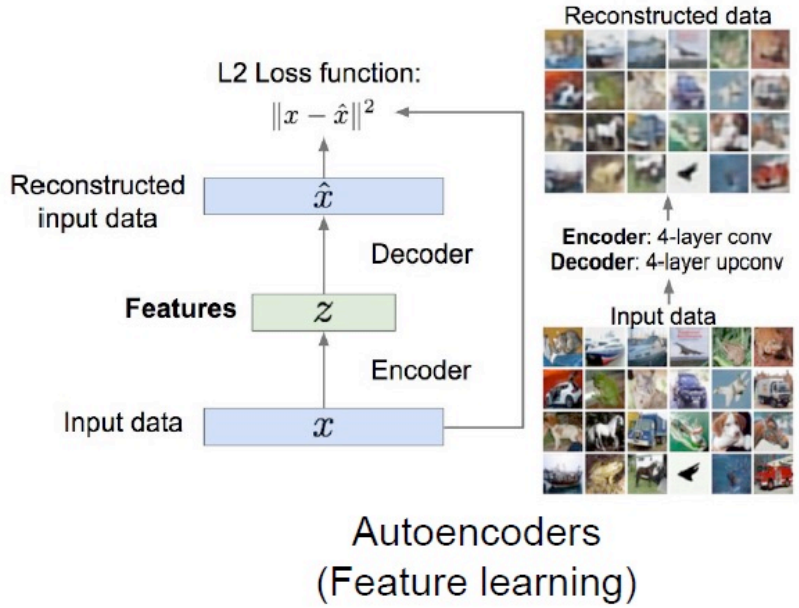
Unsupervised Encode Input Vector

- Reconstruct input: Numbers neural in input layer and output layer are equal
- The middle layer may have:
  - less neurons: undercomplete autoencoder
  - more neurons: overcomplete autoencoder
  - W_1 and W_2 are usually (but not always) tied, i.e. $W_2 = W_1^T$



Deep Convolutional Autoencoder





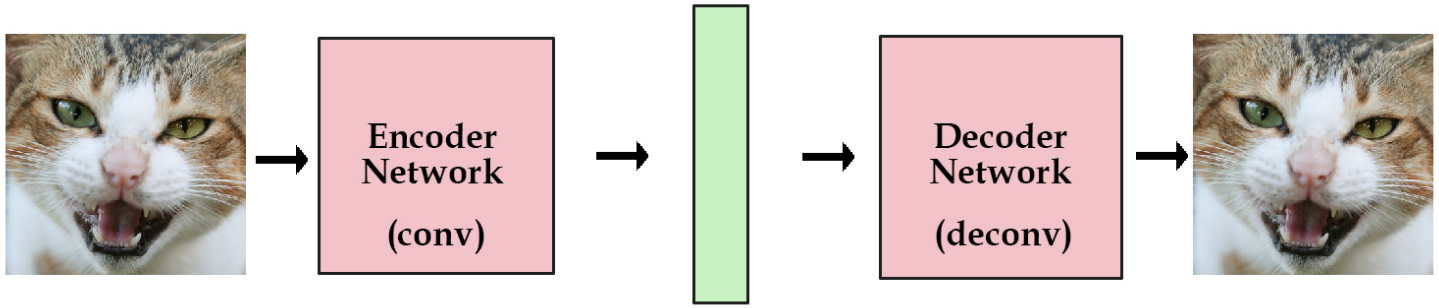
Autoencoders
(Feature learning)

Variational Autoencoders

Encoder to represent latent feature of image

Decoder:

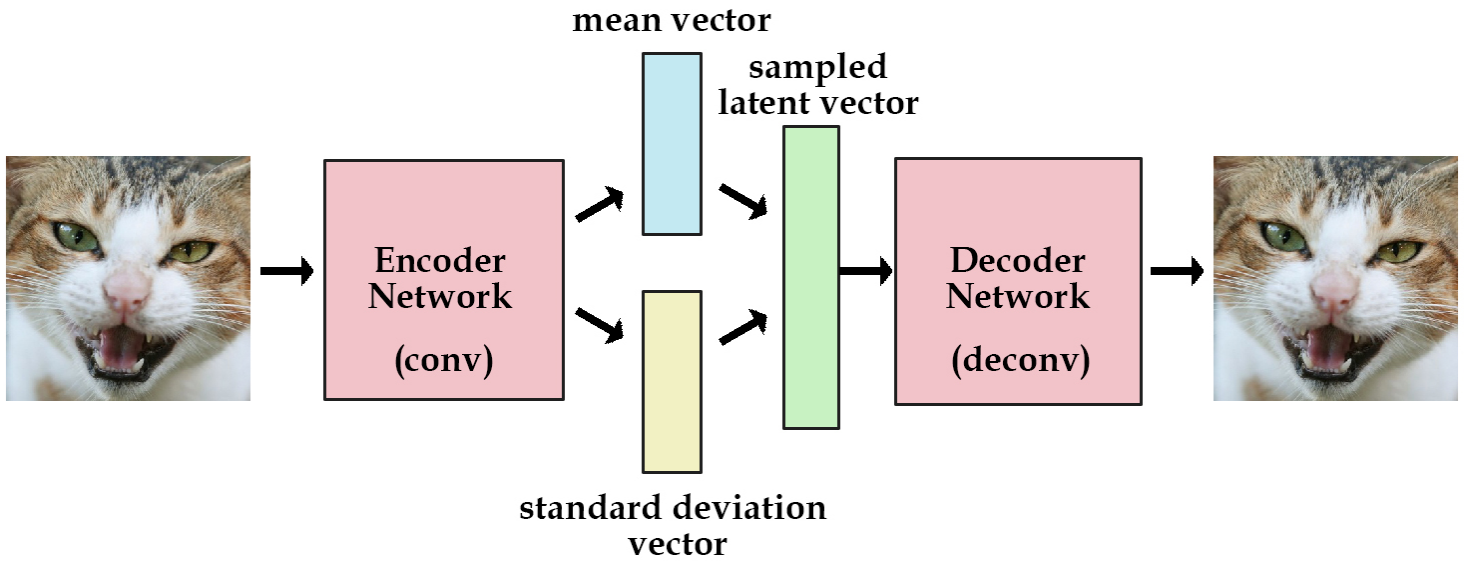
- Use as generator to generate images from a vector
- Problem: Just remember learned images, not gaussian distribution



Variational Autoencoder

Solution: separate two losses

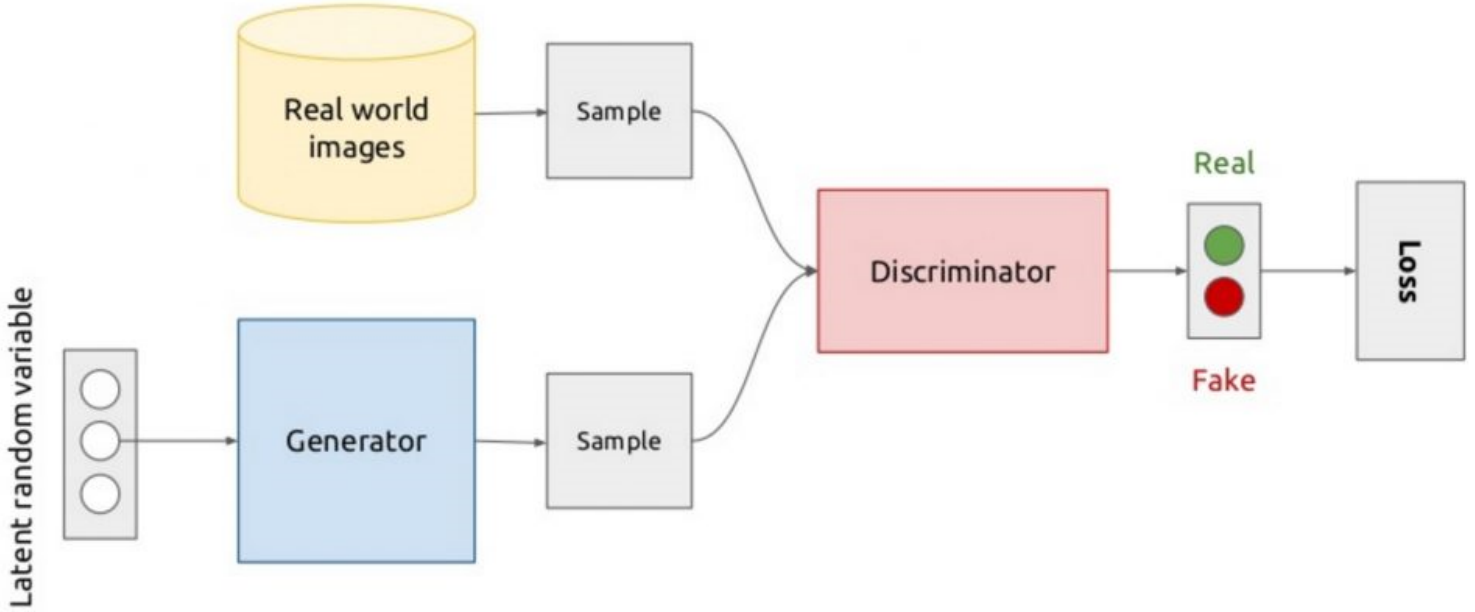
- Generative loss: mean squared error of accurately reconstructed images
- Latent loss: measure how closely latent variable match a unit gaussian



Generative Adversarial Network

Generator to create fake image

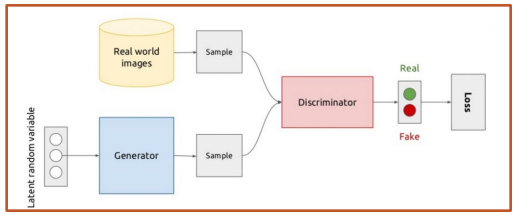
Discriminator to classify real or fake image

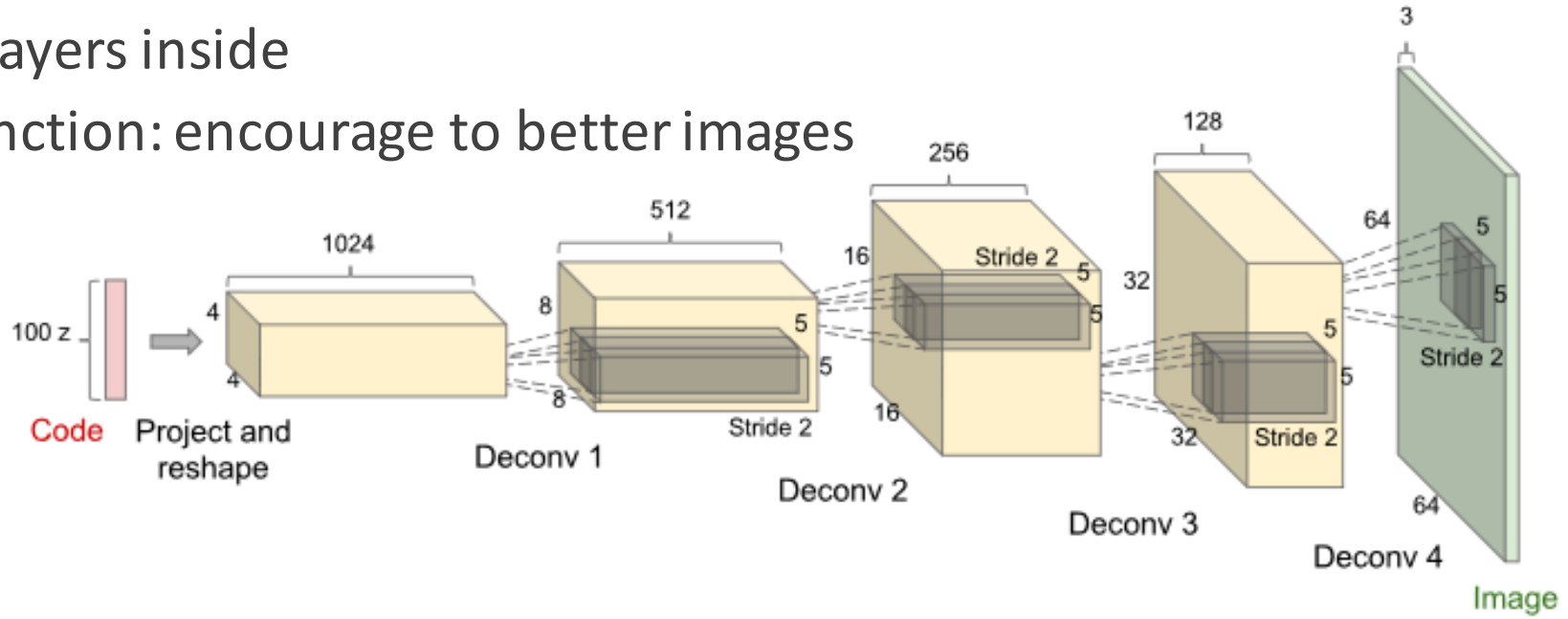


Deconvolutional Generator

Generator

- Takes code: parameter of noise
- Output synthetic image accordingly
- Deconvolutional Layers inside
- Generator loss Function: encourage to better images





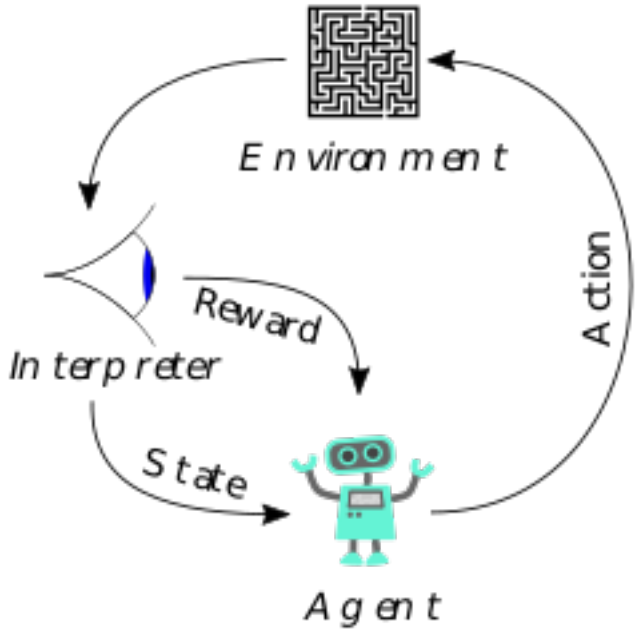
Reinforcement learning

Making good decision to do new task: fundamental challenge in Artificial Intelligence and Machine learning

Learn to make good sequence of decisions

Intelligent agents learning and acting

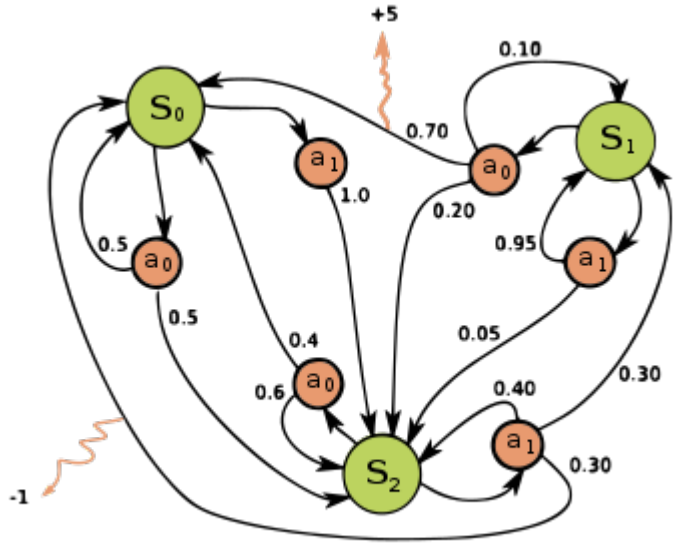
- Learning by trial-and-error, in real time
- Improve with experience
- Inspired by psychology:
  - Agents + environment
  - Agents select action to maximize *cumulative* rewards



Reinforcement Learning: Approaches

The goal is to learn a policy of behaviour

- Usually, environment is a model
- (At least) three possibilities:
  - Learn policy directly
  - Learn values of each action - infer policy by inspection
  - Learn a model - infer policy by planning
- Agents therefore typically have at least one of these components:
  - Policy - maps current state to action
  - Value function - prediction of value for each state and action
  - Model - agent's representation of the environment.



Markov Decision Process

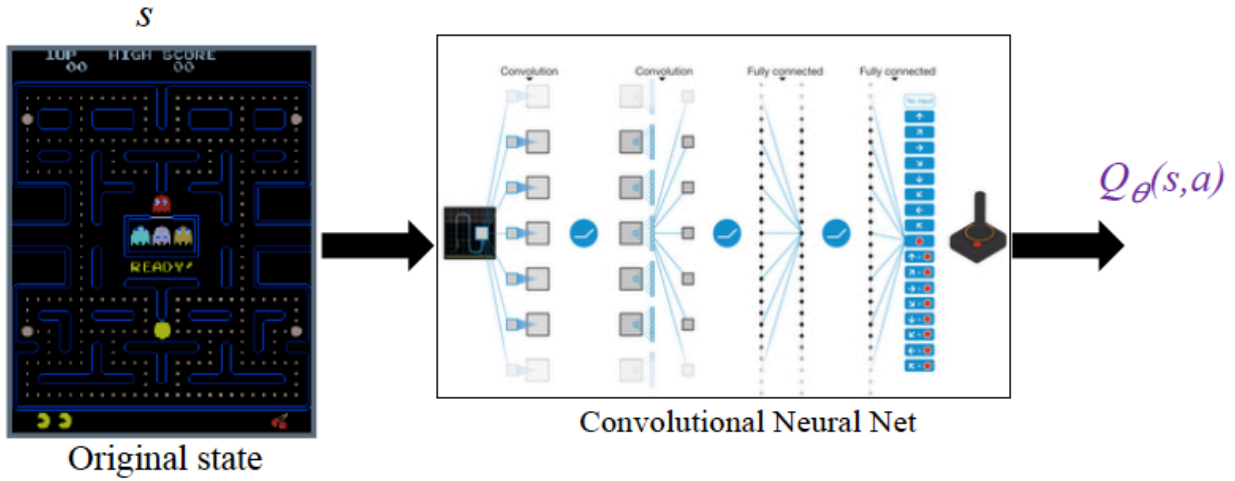
Deep Reinforcement Learning: DeepNN + RL

Real applications: very large environment, many states, noise

- Use deep learning to learn policies, values or models to use in a reinforcement learning domain
- Function estimation:

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$



Deep Q-Network trained with stochastic gradient descent.

[DeepMind: Mnih et al., 2015].

Deep Q-Network

Deep Q-Network: experience replay and target network

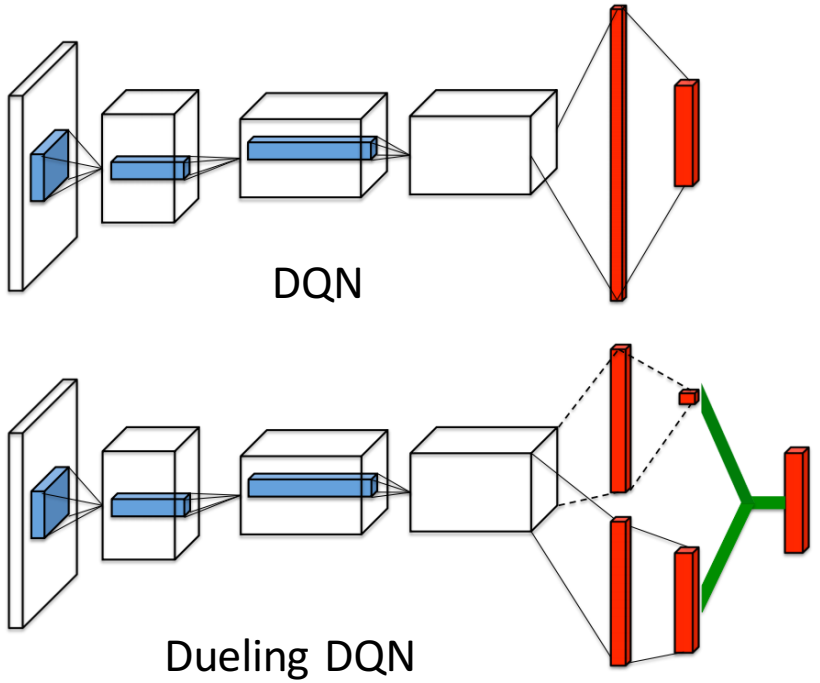
- Copy network from Q-value function
- Store experience and sample for training
- More like supervised learning

Dueling Q-Learning

- Separate value and advance functions

  $Q(s,a) = V(s) + A(a)$

- Combine them back into a single Q-function at the final layer



Deep Learning Recap

Deep Learning Building Blocks

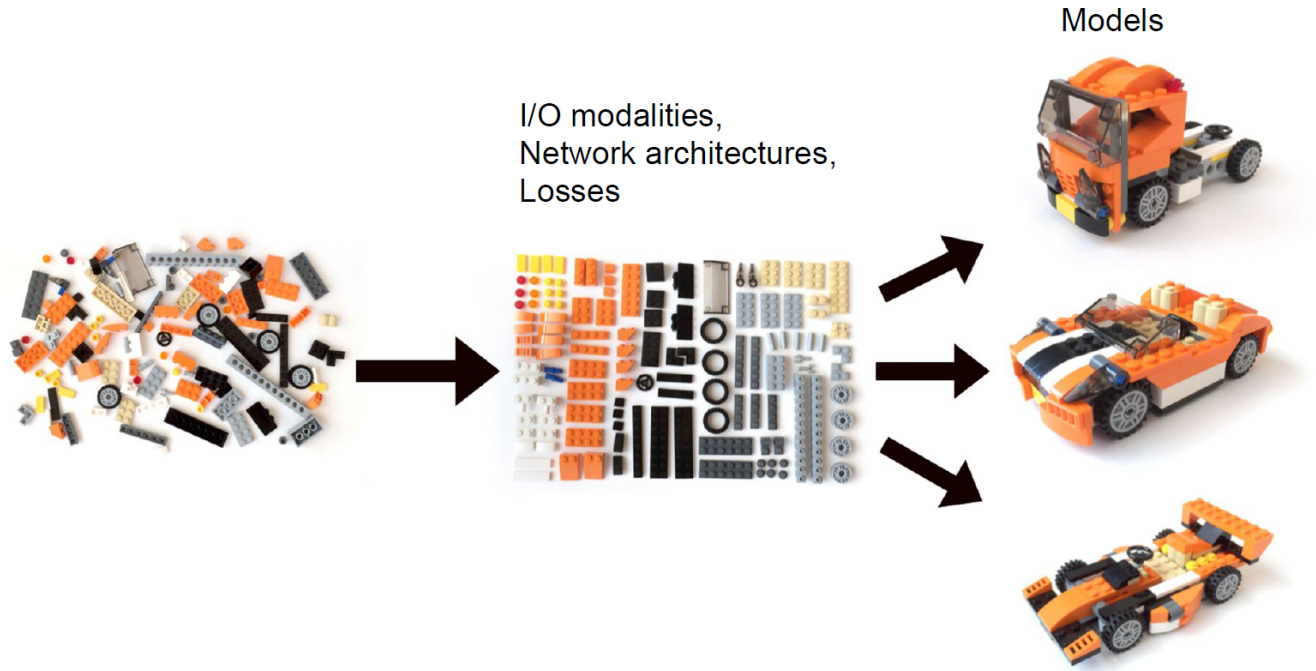


Image: Nagel, Wolfram. Multiscreen UX Design: Developing for a Multitude of Devices. Morgan Kaufmann, 2015.

5

Deep Learning Recap

Deep Learning: Zooming Out

Platforms

Microsoft Azure

aws

Google Cloud

Frameworks

Caffe K Keras

Microsoft CNTK mxnet theano

PYTORCH Caffe2

Datasets

Caltech 101 IMAGENET

LDC Linguistic Data Consortium

UCI Machine Learning Repository

WMT Workshop 2014

6

Deep Learning Recap

Deep Learning: Zooming In

Non-Linearities
Relu
Sigmoid
Tanh
GRU
LSTM
Linear
...

Optimizer
SGD
Momentum
RMSProp
Adagrad
Adam
Second Order (KFac)
...

Connectivity Pattern
Fully connected
Convolutional
Dilated
Recurrent
Recursive
Skip / Residual
Random
...

Loss
Cross Entropy
Adversarial
Variational
Max. Likelihood
Sparse
L2 Reg
REINFORCE
...

Hyper Parameters
Learning Rate
Decay
Layer Size
Batch Size
Dropout Rate
Weight init
Data augmentation
Gradient Clipping
Beta
Momentum

7

Q&A

THANK YOU!