Insight
Centre for Data Analytics

DEEP LEARNING WORKSHOP

Dublin City University
21-22 May 2018

DCU

Day 1 Lecture 2

Deep Neural Networks

Eva Mohedano
eva.mohedano@insight-centre.org

Postdoctoral Researcher
Insight Centre for Data Analytics
Dublin City University

Overview

- Limitations the perceptron model
- Principle of deep learning
- Multilayer perceptron
- Convolutional neural networks

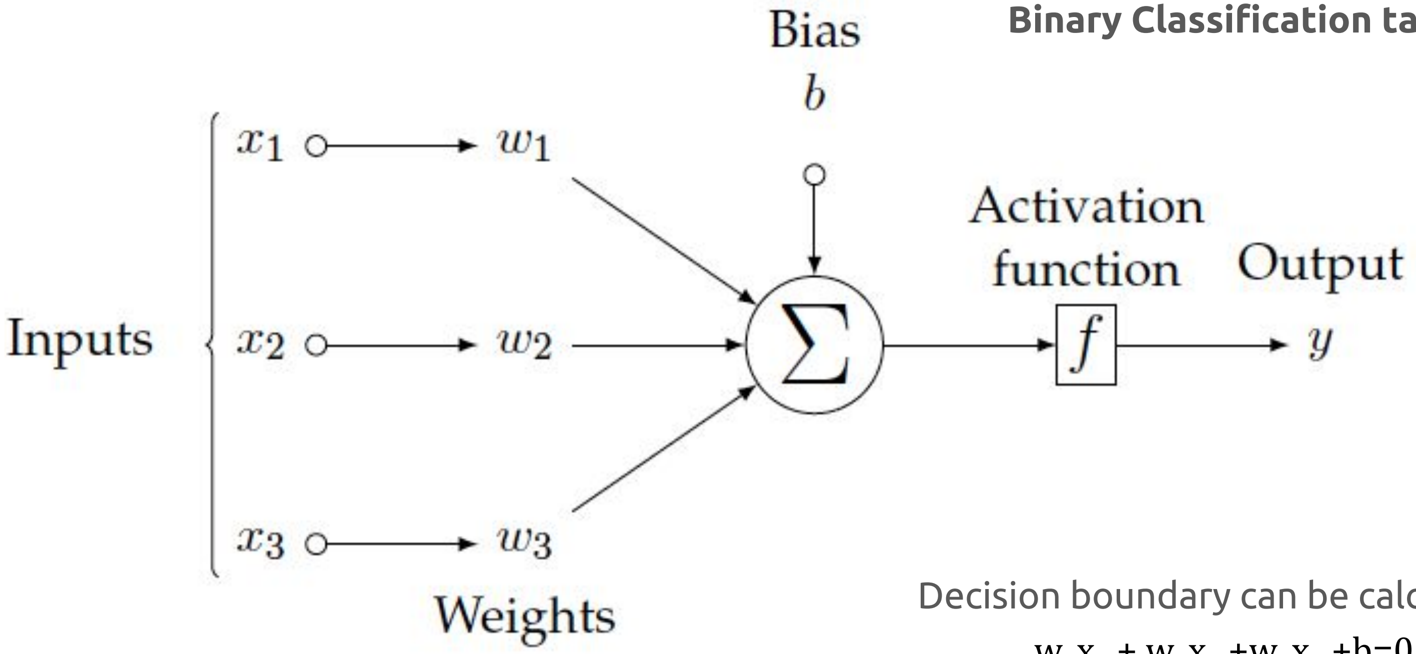
Overview

- **Limitations the perceptron model**
- Principle of deep learning
- Multilayer perceptron
- Convolutional neural networks

Perceptron (Neuron)

If the weighted sum of the input exceeds a threshold the neuron fires a signal.

Binary Classification task

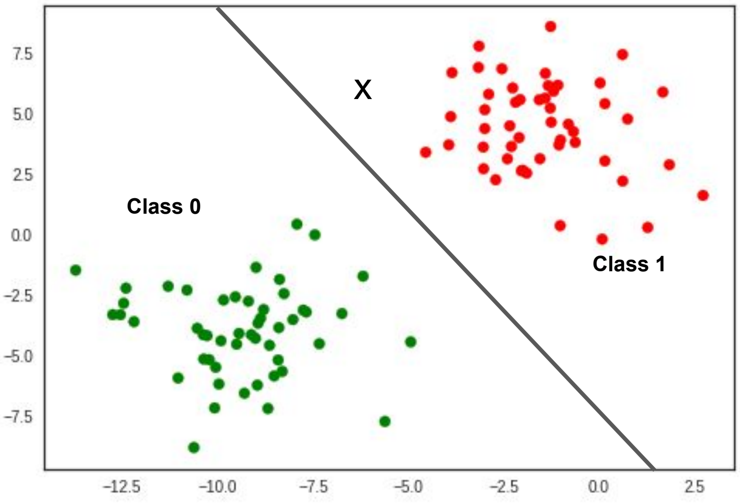


Decision boundary can be calculated by:

$$w_1x_1 + w_2x_2 + w_3x_3 + b = 0$$

Linear decision decision boundary

2D input space data



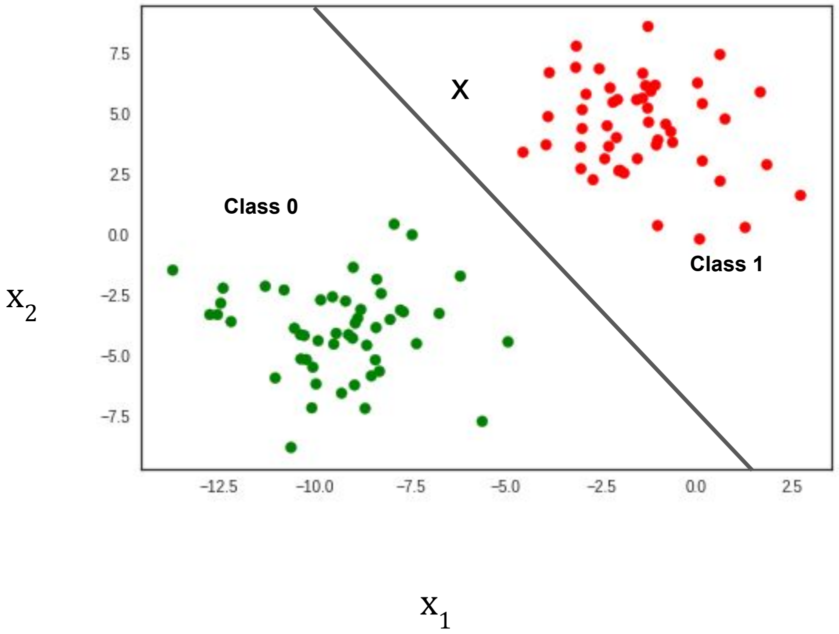
x_2

x_1

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Linear decision decision boundary

2D input space data



Parameters of the line.
They are find based on training data
- *Learning Stage*.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

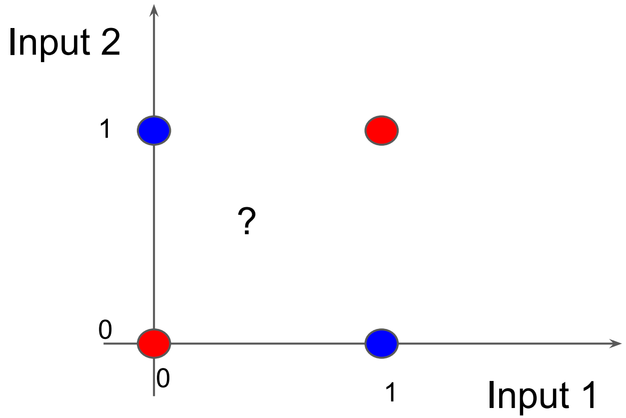
Limitations: XOR problem

Data might be **non linearly separable**

→ One single neuron is not enough

XOR logic table

Input 1	Input 2	Desired Output
0	0	0
0	1	1
1	0	1
1	1	0

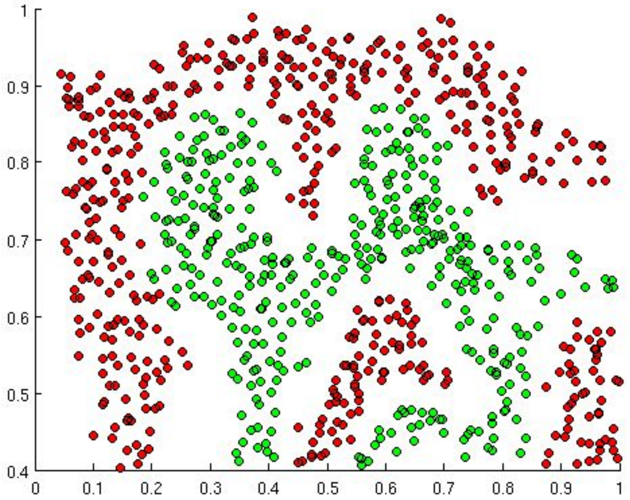


Non-linear decision boundaries

Linear models can only produce linear decision boundaries

Real world data often needs a non-linear decision boundary

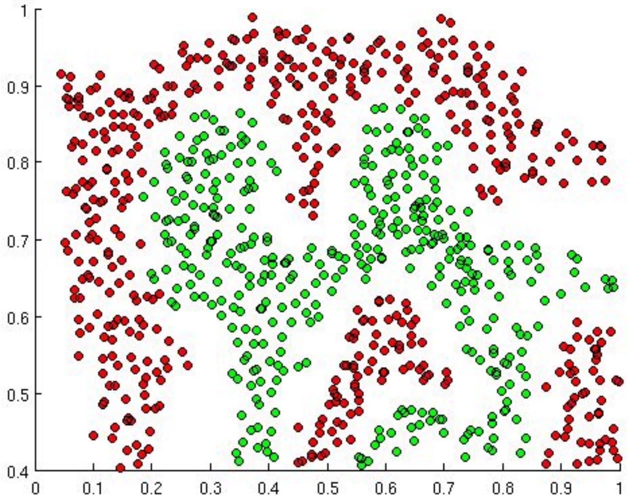
- Images
- Audio
- Text



Non-linear decision boundaries

What can we do?

1. Use a non-linear classifier
   - Decision trees (and forests)
   - K nearest neighbours
2. Engineer a suitable representation
   - One in which features are more linearly separable
   - Then use a linear model
3. Engineer a kernel
   - Design a kernel $\mathbf{K(x_1, x_2)}$
   - Use kernel methods (e.g. SVM)
4. Learn a suitable representation space from the data
   - Deep learning, deep neural networks
   - Boosted cascade classifiers like Viola Jones also take this approach

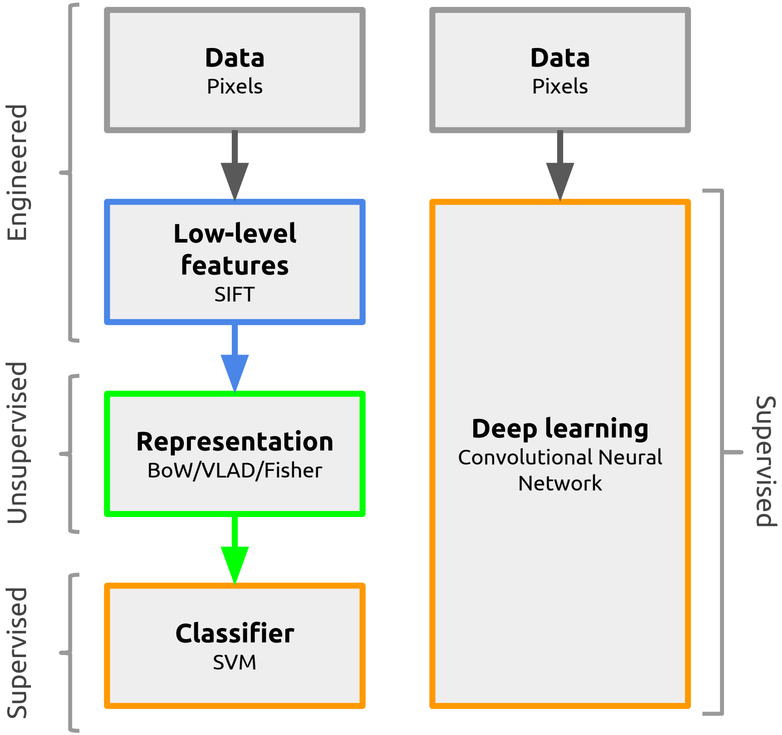


Overview

- Limitations the perceptron model
- **Principle of deep learning**
- Multilayer perceptron
- Convolutional neural networks

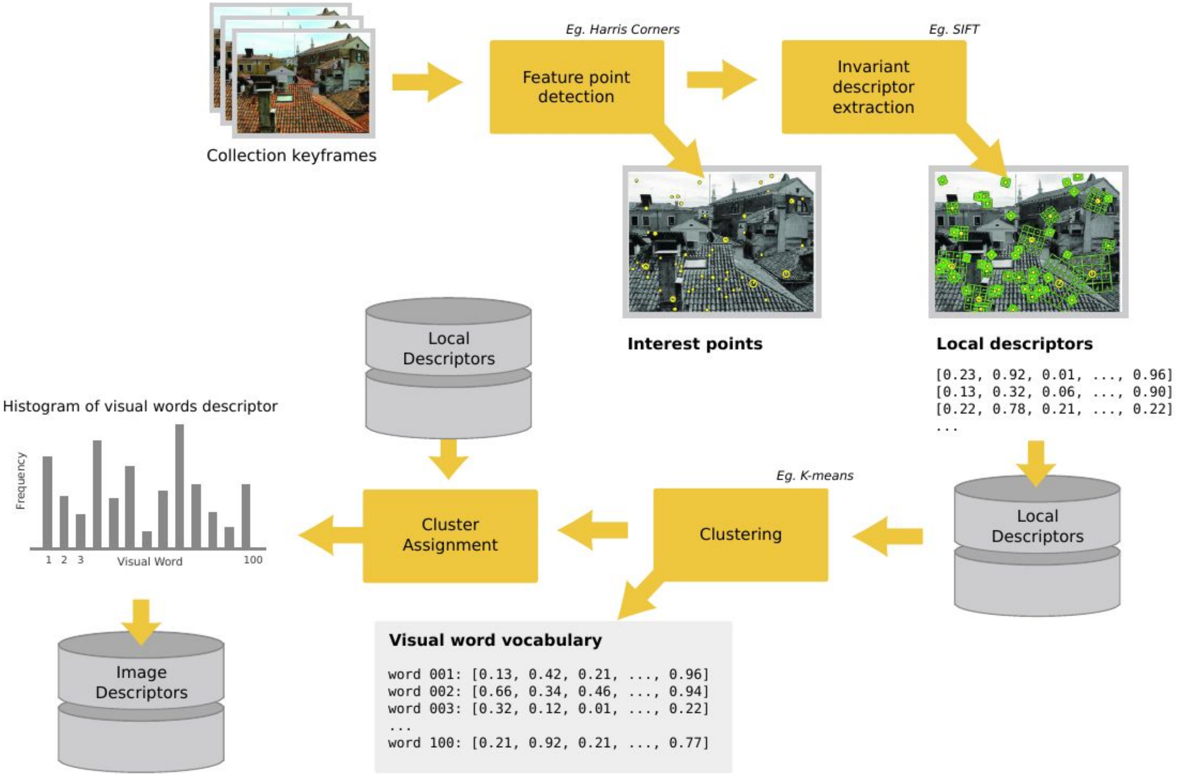
Principle of deep learning

- Old style machine learning:
  - Engineer features (by some unspecified method)
  - Create a representation (descriptor)
  - Train shallow classifier on representation
- Example:
  - SIFT features (engineered)
  - BoW representation (engineered + unsupervised learning)
  - SVM classifier (convex optimization)
- Deep learning
  - Learn layers of features, representation, and classifier in one go based on the data alone
  - Primary methodology: deep neural networks (non-convex)



Example: feature engineering in computer vision

$\phi(x)$



Neural networks: single neuron

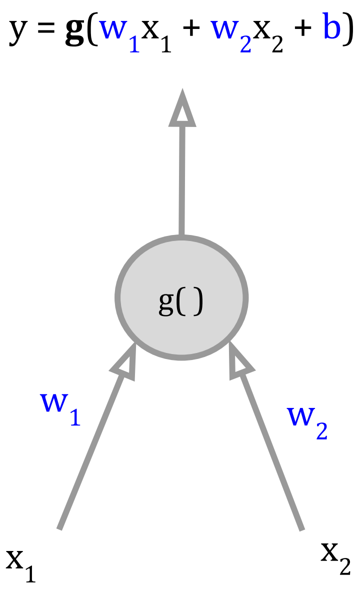
We already seen the single neuron. This is just a linear classifier (or regressor)

Inputs:

- x_1, x_2

Parameters

- w_1, w_2, b



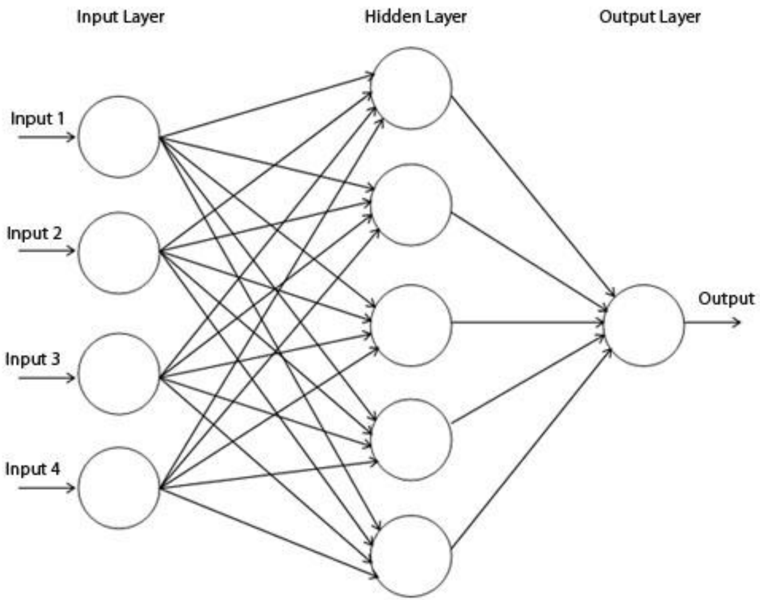
Neural networks

A **composition** of these simple neurons into several layers

Each neuron simply computes a **linear combination** of its inputs, adds a bias, and passes the result through an **activation function** g(x)

The network can contain one or more **hidden layers**. The outputs of these hidden layers can be thought of as a new **representation** of the data (new features).

The final output is the **target** variable (y = f(x))



Activation functions

g( ) - transfer functions, nonlinearities, units

- They act as a **threshold**

Desirable properties

- Mostly smooth, continuous, differentiable
- Fairly linear

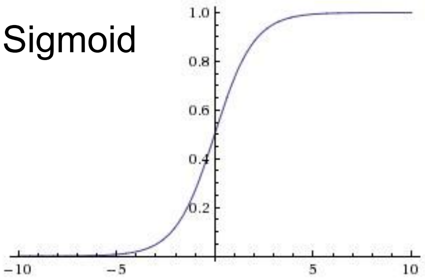
Common nonlinearities

- Sigmoid
- Tanh
- ReLU = max(0, x)

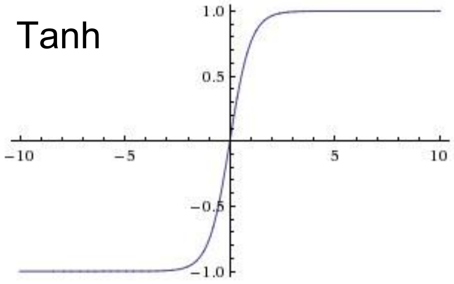
Why do we need them?

If we only use linear layers we are only able to learn linear transformations of our input.

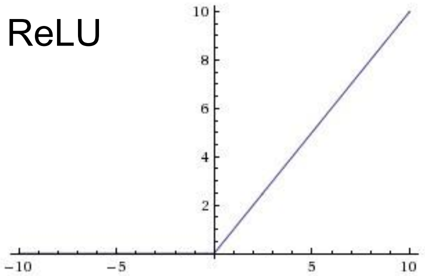
Sigmoid



Tanh



ReLU



Overview

- Limitations the perceptron model
- Principle of deep learning
- **Multilayer perceptron**
- Convolutional neural networks

Multilayer perceptrons

When each node in each layer is a linear combination of **all inputs from the previous layer** then the network is called a multilayer perceptron (MLP)

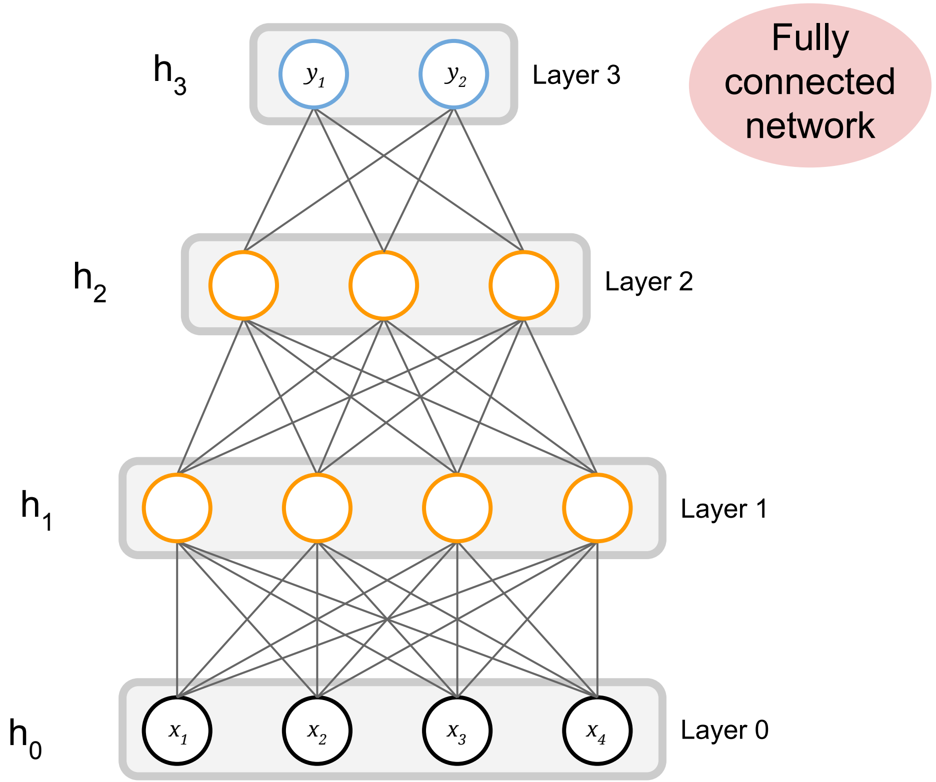
Weights can be organized into matrices.

Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$



Multilayer perceptrons

W_1

w_{11}	w_{12}	w_{13}	w_{14}
w_{21}	w_{22}	w_{23}	w_{24}
w_{31}	w_{32}	w_{33}	w_{34}
w_{41}	w_{42}	w_{43}	w_{44}

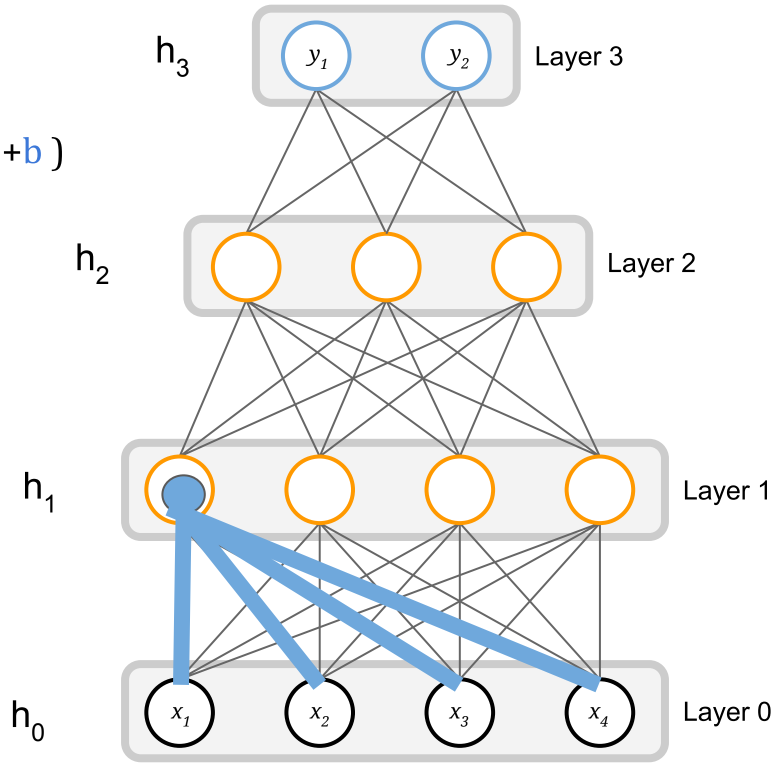
h_0

x_1
x_2
x_3
x_4

b_1

b_1
b_2
b_3
b_4

$h_{11} = g(\, \mathbf{w}\mathbf{x} + b \,)$



Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$

Multilayer perceptrons

W_1

w_{11}	w_{12}	w_{13}	w_{14}
w_{21}	w_{22}	w_{23}	w_{24}
w_{31}	w_{32}	w_{33}	w_{34}
w_{41}	w_{42}	w_{43}	w_{44}

h_0

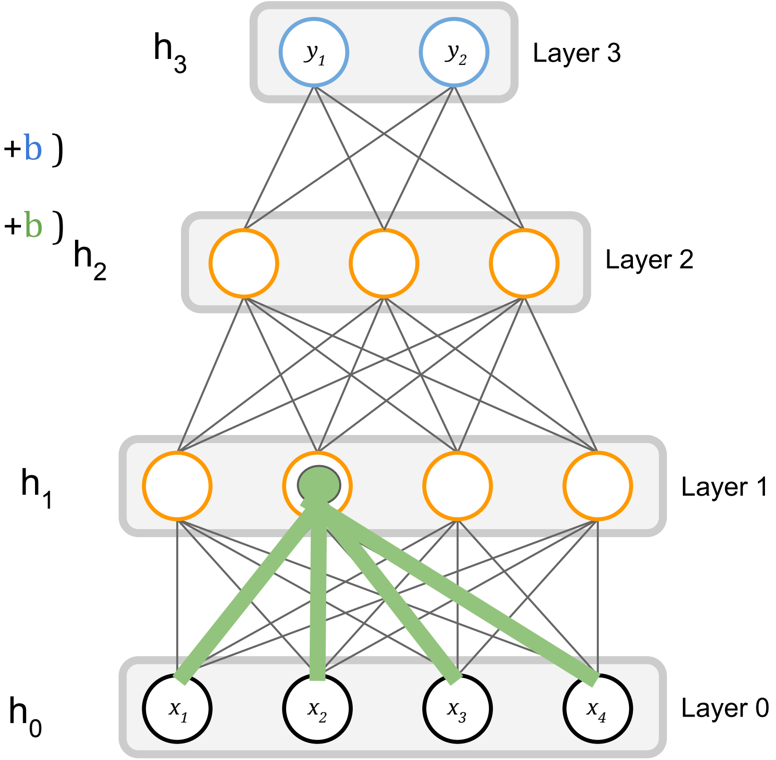
x_1
x_2
x_3
x_4

b_1

b_1
b_2
b_3
b_4

$h_{11} = g(\mathbf{w}\mathbf{x} + b)$

$h_{12} = g(\mathbf{w}\mathbf{x} + b)$



Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$

Universal approximation theorem

Universal approximation theorem states that "the standard multilayer feed-forward network with **a single hidden layer**, which contains **finite number of hidden neurons**, is a **universal approximator** among continuous functions on compact subsets of R^n, under mild assumptions on the activation function."

If a 2 layer NN is a universal approximator, then why do we need deep nets??

The universal approximation theorem:

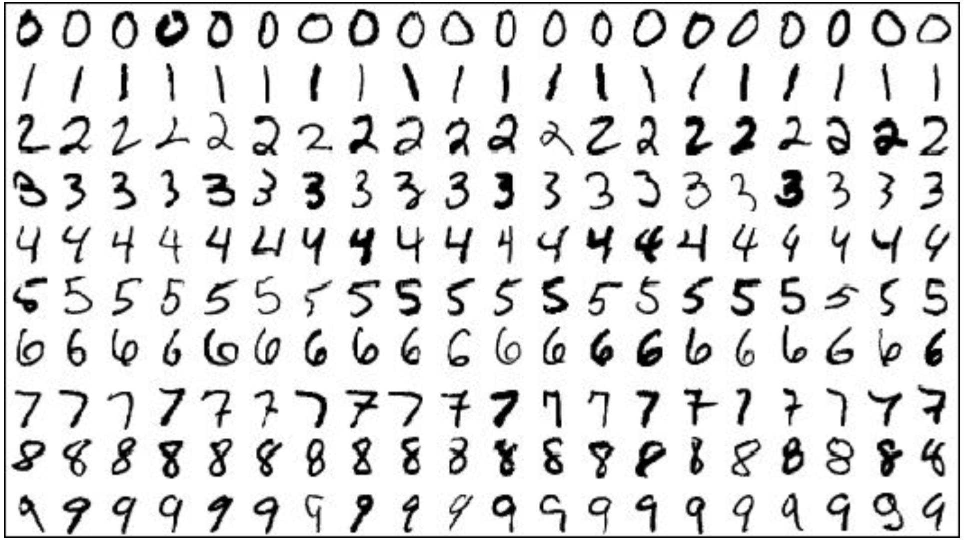
- Says nothing about the how easy/difficult it is to fit such approximators
- Needs a "finite number of hidden neurons": finite may be extremely large

In practice, deep nets can usually represent more complex functions with less total neurons (and therefore, less parameters)

Example: MNIST digit classification

MNIST

- Popular dataset of handwritten digits
- 60,000 training examples
- 10,000 test examples
- 10 classes (digits 0-9)
- http://yann.lecun.com/exdb/mnist/
- 28x28 grayscale images (784D)



Objective

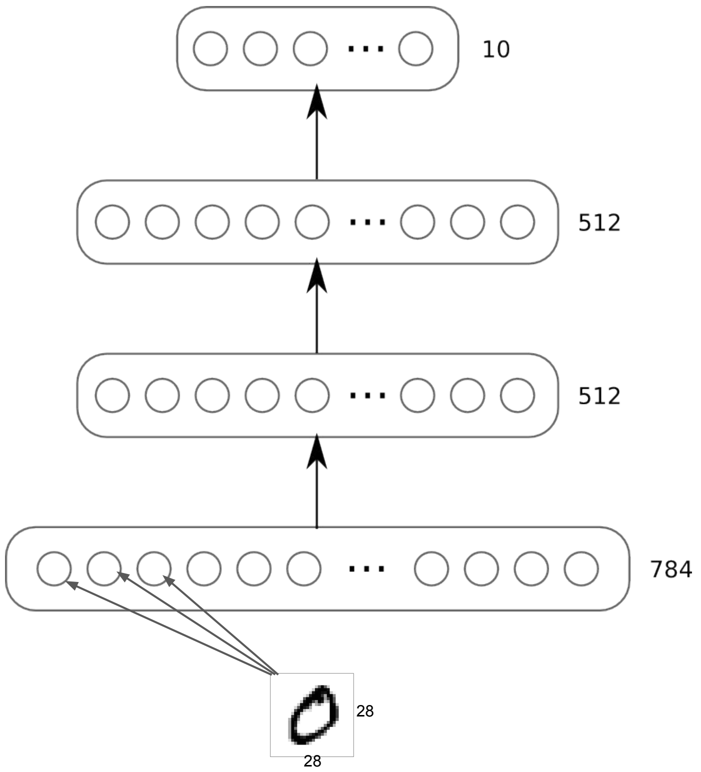
- Learn a function y = f(x) that predicts the digit from the image
- Measure accuracy on test set

Example: MNIST digit classification

Model

- 3 layer neural network (2 hidden layers)
- Tanh units (activation function)
- 512-512-10
- **Softmax** on top layer
- **Cross entropy** loss

Layer	#Weights	#Biases	Total
1	784 x 512	512	401,920
2	512 x 512	512	262,656
3	512 x 10	10	5,130
			669,706



Example: MNIST digit classification

Training:

- 40 epochs using mini-batch SGD
- Batch size: 128
- Learning rate: 0.1 (fixed)
- Weight decay λ = 1e-5
- Takes about 5 mins to train on a GPU

Accuracy:

- **98.12%** (188 errors in 10,000 test examples)
- Linear classifier: 88% accuracy (1200 errors)
- Sigmoid units give 95.5%

Improving accuracy and speeding convergence:

- Replace sigmoid with ReLU
- Use RMSprop optimizer
- Add dropout (0.2) after each hidden layer
- Accuracy ~98.4%
- Trains in 20 epochs

Try it yourself!

- https://github.com/fchollet/keras/blob/master/examples/mnist_mlp.py

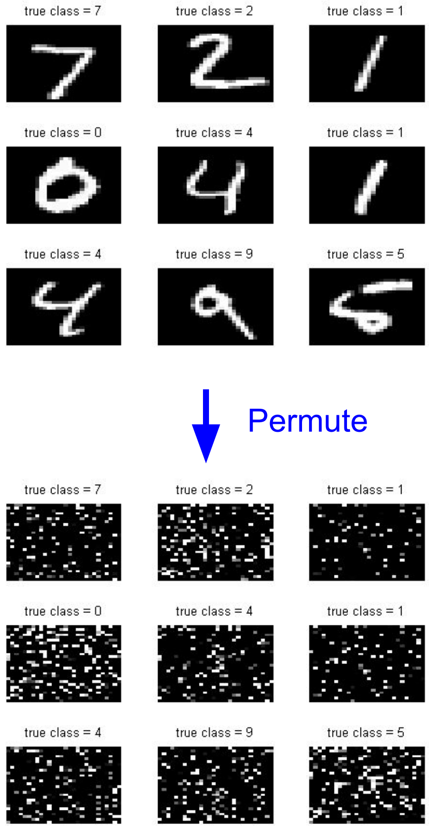
Permutation invariant MNIST

There is something interesting about our previous MNIST classifier example

- It is possible to apply a fixed permutation to the pixels in the image (shuffle them around)
- This does **NOT** in any way affect the classification accuracy!
- Yet the resulting images are completely unintelligible to humans
- It is difficult to imagine that a human could learn to recognize permuted images of images

What’s going on?

- Fully connected layers assume no spatial neighbourhood relationships
- Maybe we can do better if we somehow embed these structural relationships into the algorithm…



Overview

- Limitations the perceptron model
- Principle of deep learning
- Multilayer perceptron
- **Convolutional neural networks**

Convolutional neural networks (CNNs, convnets)

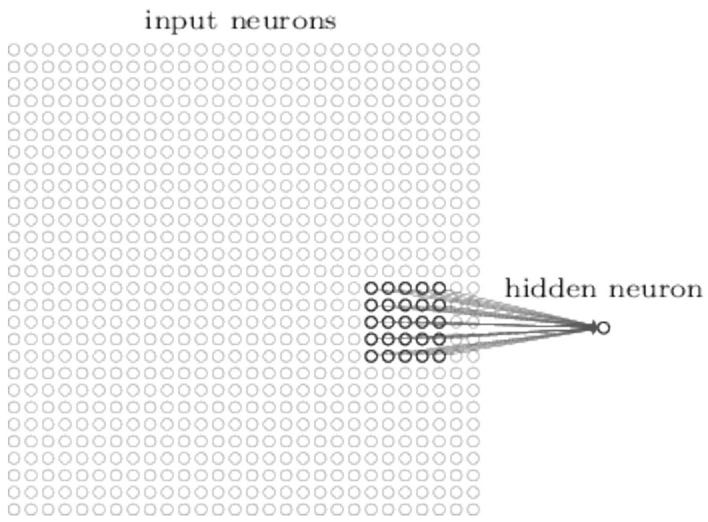
Key idea: good features to learn for images are:

- **Local**: only depend on a small part of the image, not the whole image
- **Translation invariant**: if a feature is good for one part of an image, it should be good for others too.

Instead of a big matrix multiplication on the whole image, apply a whole lot of little matrix multiplications against each image patch and store the local activations.

This is called **convolution**

Parameters are **shared** across these convolutional **kernels** (translation invariance)



1D convolution

Signal	0	1	5	0	2	8	1	0

dot product

Kernel/filter	-1	2	-1

=

	-3					

1D convolution

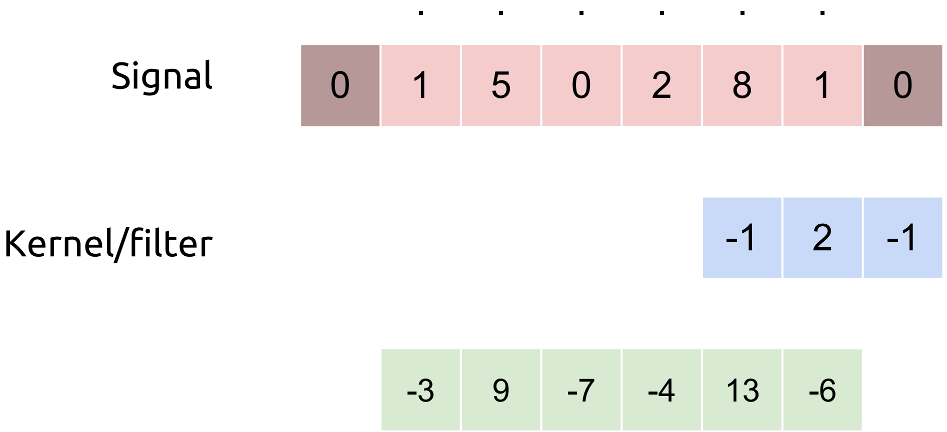
Signal	0	1	5	0	2	8	1	0
Kernel/filter		-1	2	-1				
		-3	9					

1D convolution

. . .

Signal	0	1	5	0	2	8	1	0
Kernel/filter			-1	2	-1			
		-3	9	-7				

1D convolution

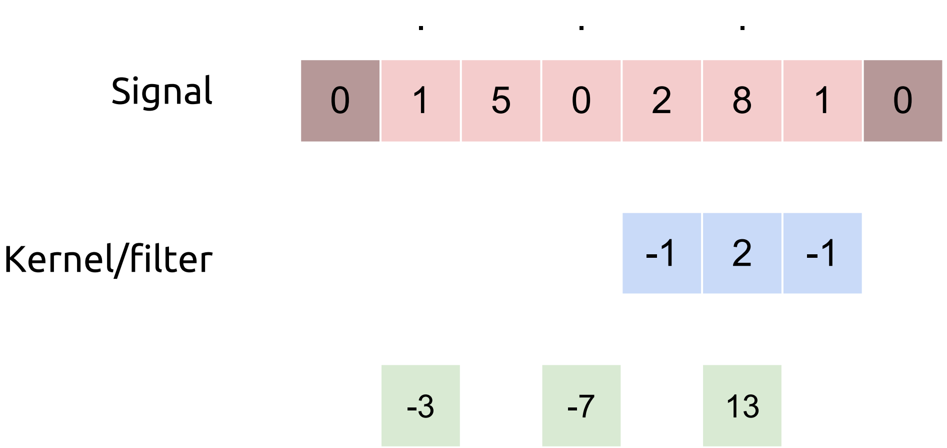


Zero padding=1 + Stride=1

↓

Convolved signal has same dimension as the input signal

1D convolution

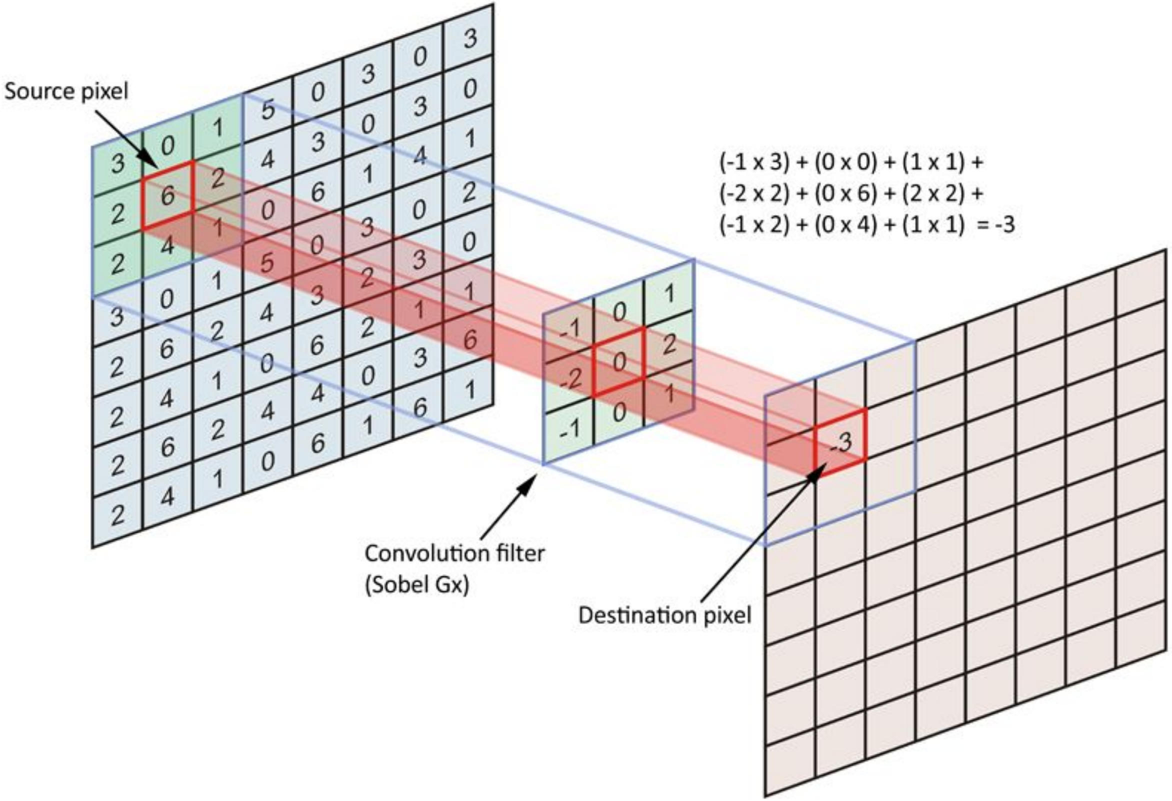


Hyperparameters

Zero padding=1 + Stride=2

Convolved signal has lower dimension (half) then the input signal

Convolution on a grid

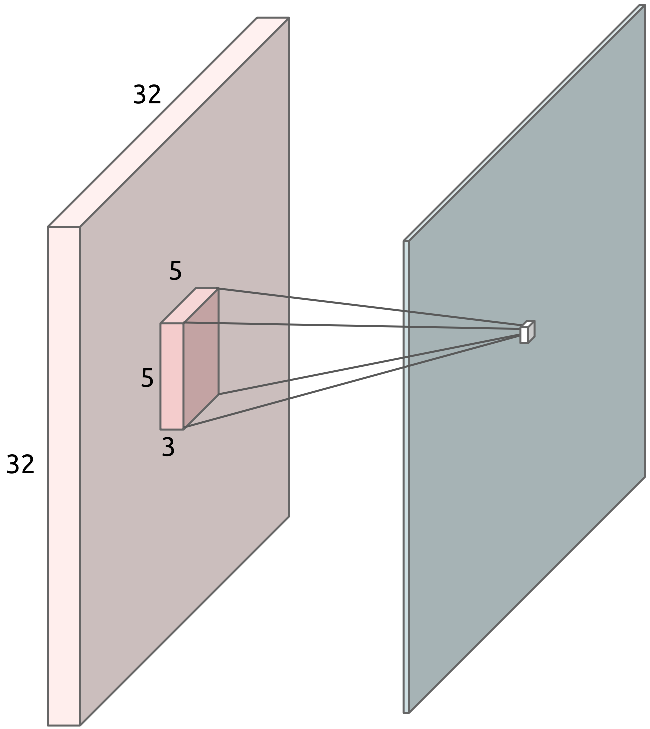


[Image source](#)

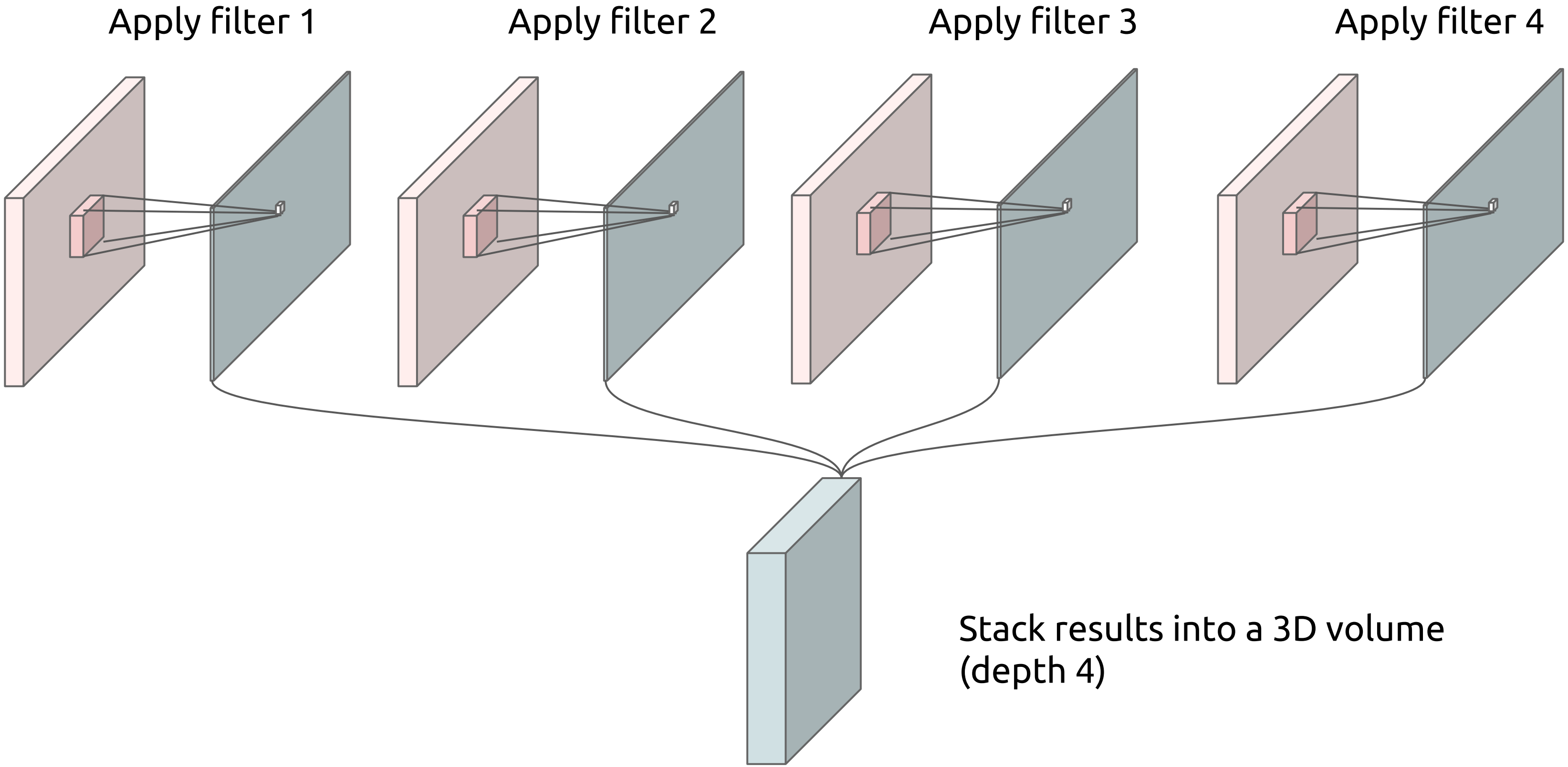
Convolution on a volume

A 5x5 convolution on a volume of depth 3 (e.g. an image) needs a filter (kernel) with 5x5x3 elements (weights) + a bias

Andrej Karpathy's demo:
http://cs231n.github.io/convolutional-networks/#conv



Convolution with multiple filters



Pooling layers

1	5	0	2	8	1
10	2	4	9	0	3
8	9	3	7	8	2
3	8	9	6	0	5
16	7	2	2	7	3
6	3	0	5	2	2

Max-pool kernel (3x3)

Stride 3

10	9
16	7

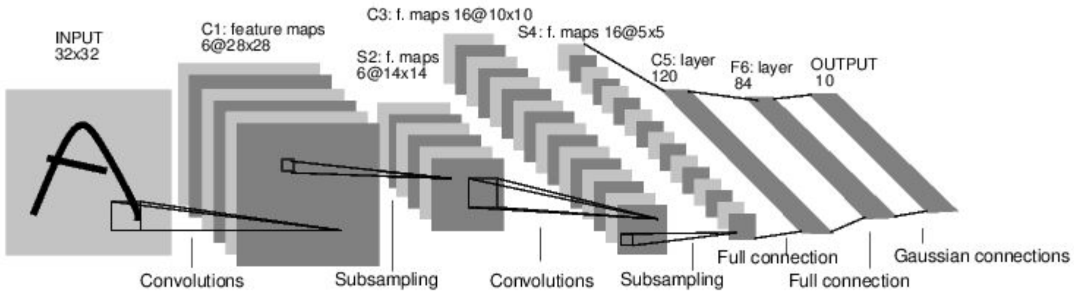
Convnets

Most convnets contain several convolutional layers, interspersed with pooling layers, and followed by a small number of fully connected layers

Pooling layers

- Reduce amount of data that needs to be processed by later layers
- Provide invariance to small local changes

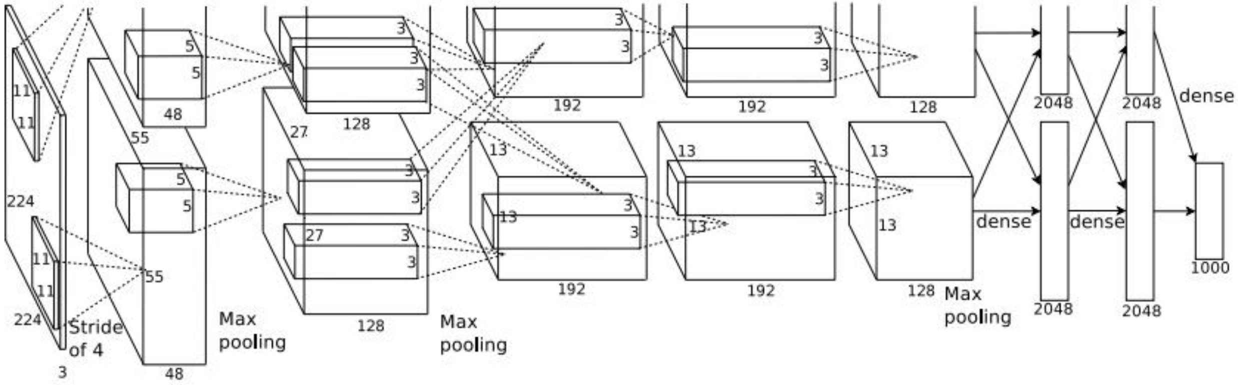
Max pooling usually used in practice.



LeNet-5 [LeCun et al. 1998]

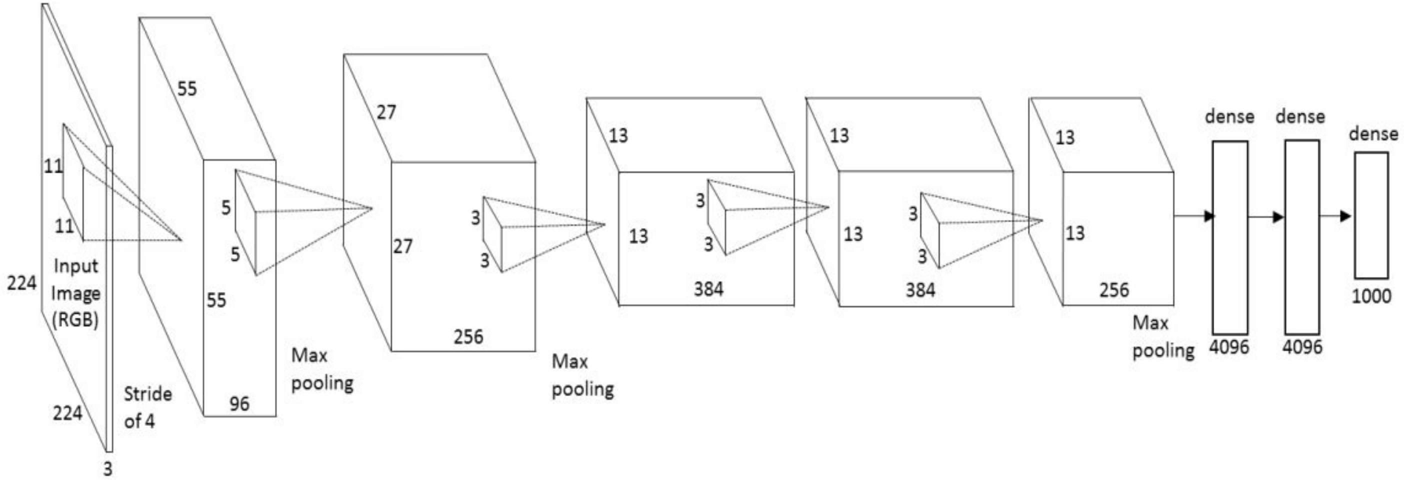
Alexnet

- 8 parameter layers (5 convolution, 3 fully connected)
- Softmax output
- 650,000 units
- 60 million free parameters
- Trained on two GPUs (two streams) for a week
- Ensemble of 7 nets used in ILSVRC challenge

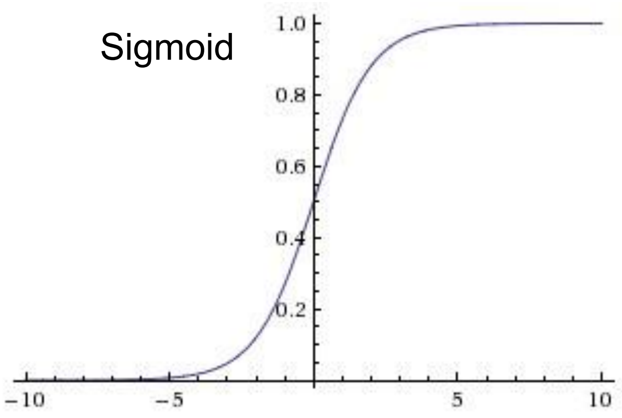


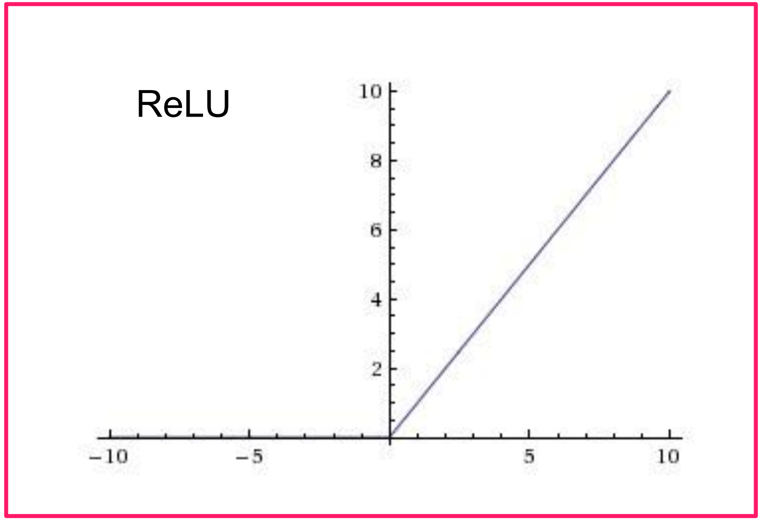
Krizhevsky et al. **ImageNet classification with deep convolutional neural networks**. NIPS, 2012.

Features of Alexnet: Convolutions



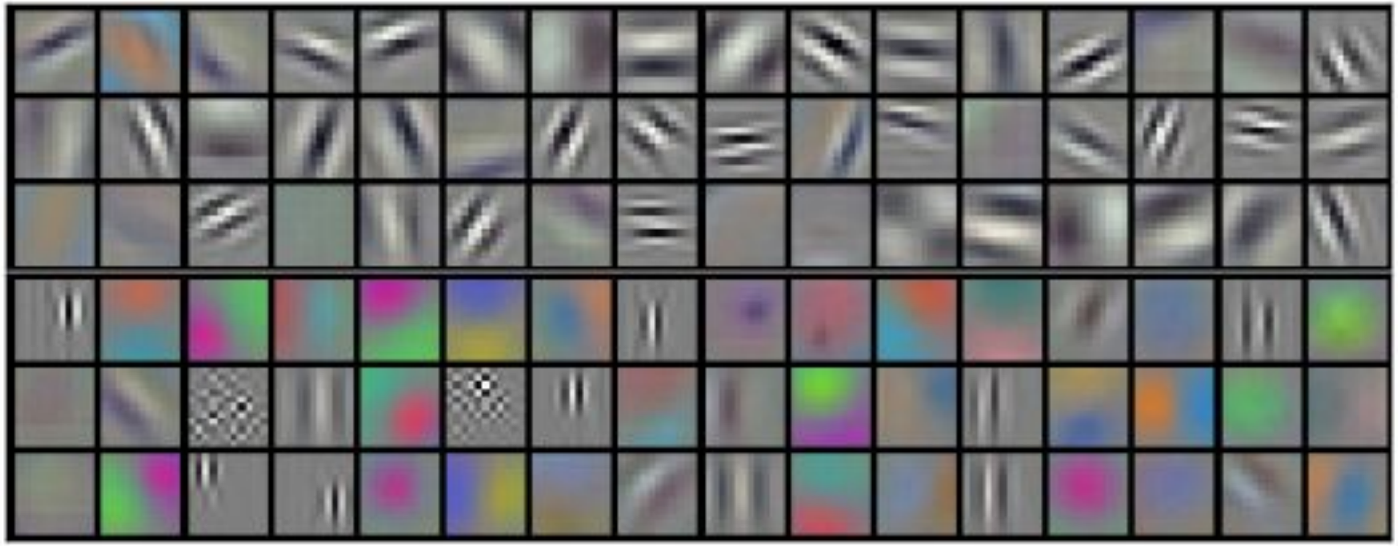
Features of Alexnet: ReLu





Filters learnt by Alexnet

Visualization of the 96 11 x 11 filters learned by bottom layer



Krizhevsky et al. **ImageNet classification with deep convolutional neural networks**. NIPS, 2012.

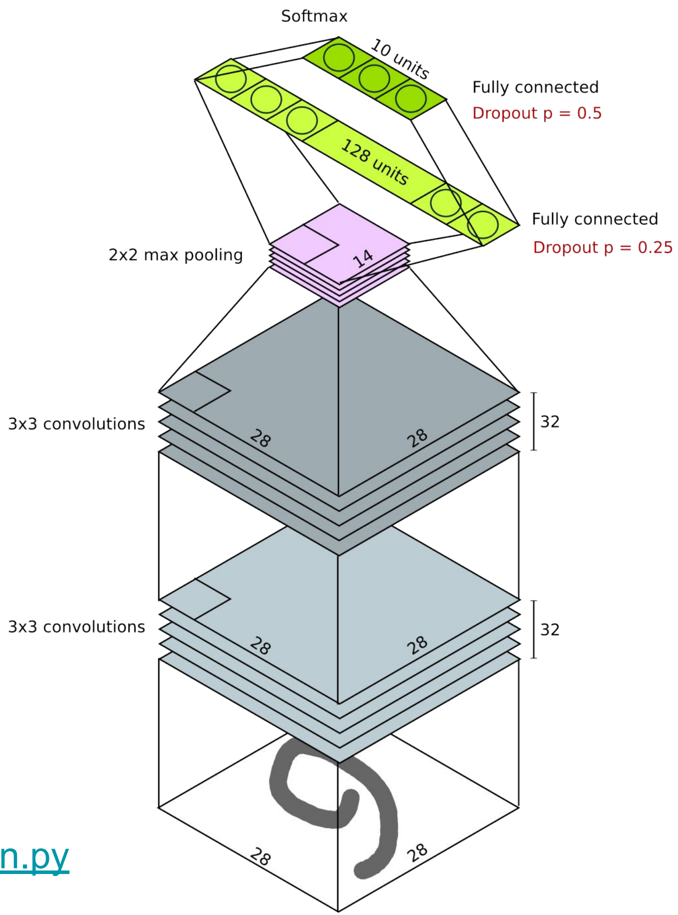
Example: a convnet on MNIST

Layers:

- 32 (3x3) convolutions + ReLU
- 32 (3x3) convolutions + ReLU
- 2x2 max pooling
- Dropout p=0.25
- Fully connected 128 units
- Dropout p=0.5
- Fully connected 10 units
- Softmax
- Cross entropy loss

Train for 12 epochs

- Accuracy **99.22%** (78 errors in 10000)



https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py

Advantages of convnets

- Significantly less parameters to learn:
  - Small local kernels
  - Shared parameters
- Faster training
  - Weight sharing means gradients are averaged for every location of the kernel
- Local features
  - Can be used to detect object location
- Interpretability
  - Can visualize the little learned filters
- Accuracy
  - Structural neighborhood assumption: not permutation invariant. Usually results in better accuracy
- Biological plausibility

Summary

- A single perceptron (neuron) can only define linear decision boundaries.
- Multilayer neural networks are compositions of simple linear models with element-wise nonlinearities.
- Deep networks focus in learning non-linear transformation of the input data
- Fully connected neural networks (MLP) are permutation invariant
- Convolutional neural networks

Thank you!