

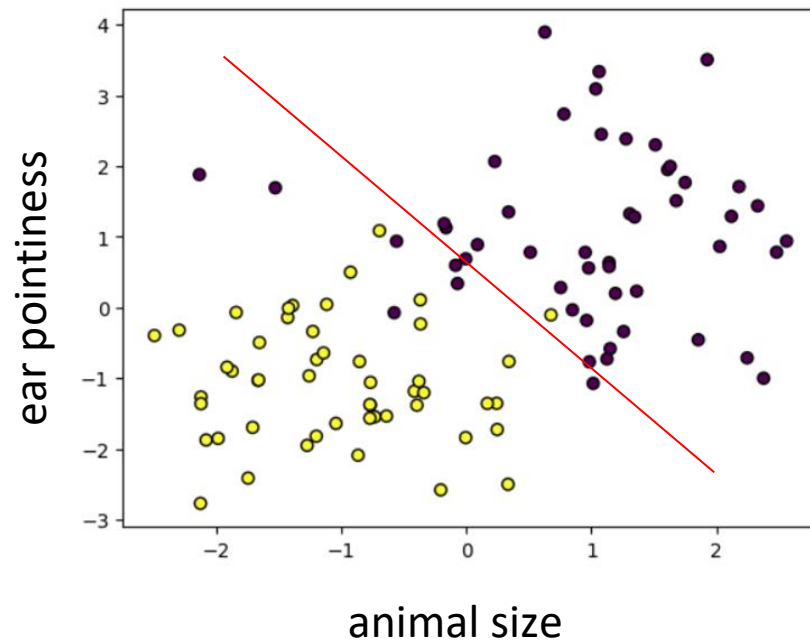
Tutorial 3: Supervised learning and feedforward neural networks

Plan

- Perceptron.
- Multi-layer Perceptron.
- Convolutional Neural Network and digit recognition
- Analogy between convolutional networks and visual cortex

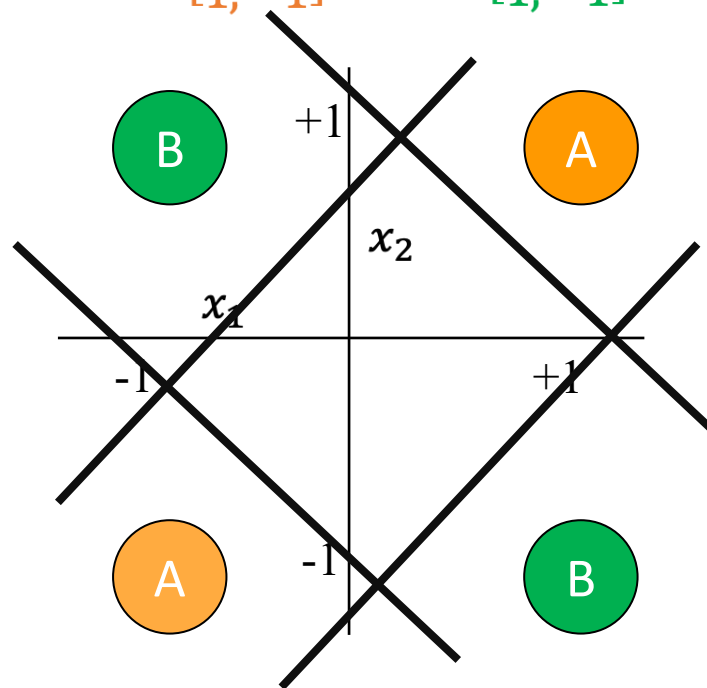


Linearly separable data

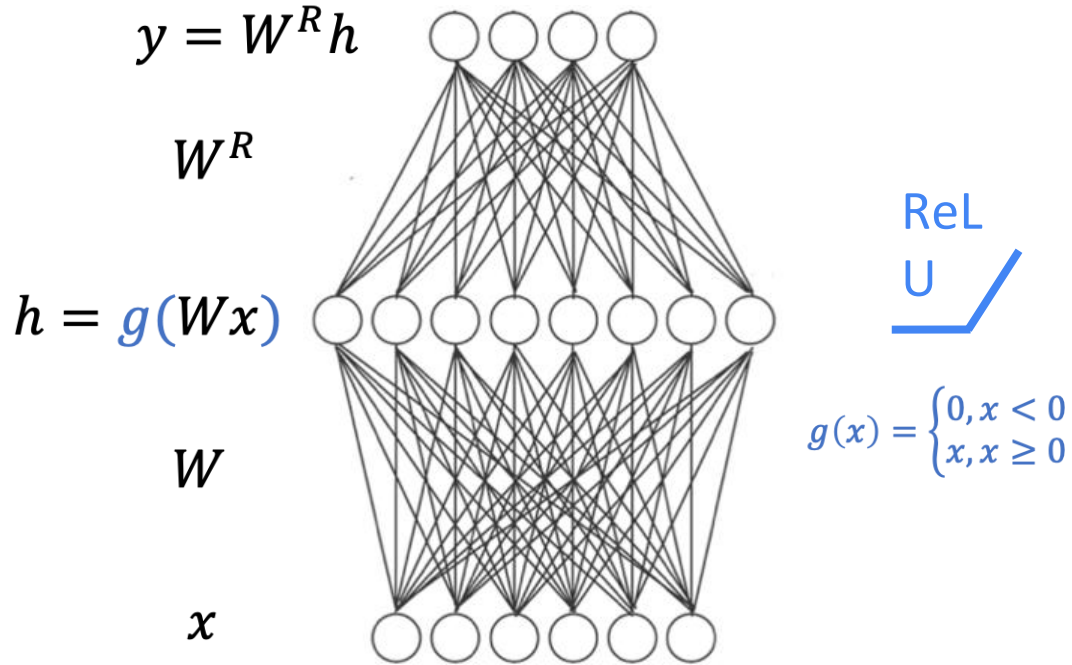


The XOR problem

$$x^A = \begin{bmatrix} 1, -1 \\ 1, -1 \end{bmatrix} \quad x^B = \begin{bmatrix} -1 & 1 \\ 1, -1 \end{bmatrix}$$



Multi-layer perceptron

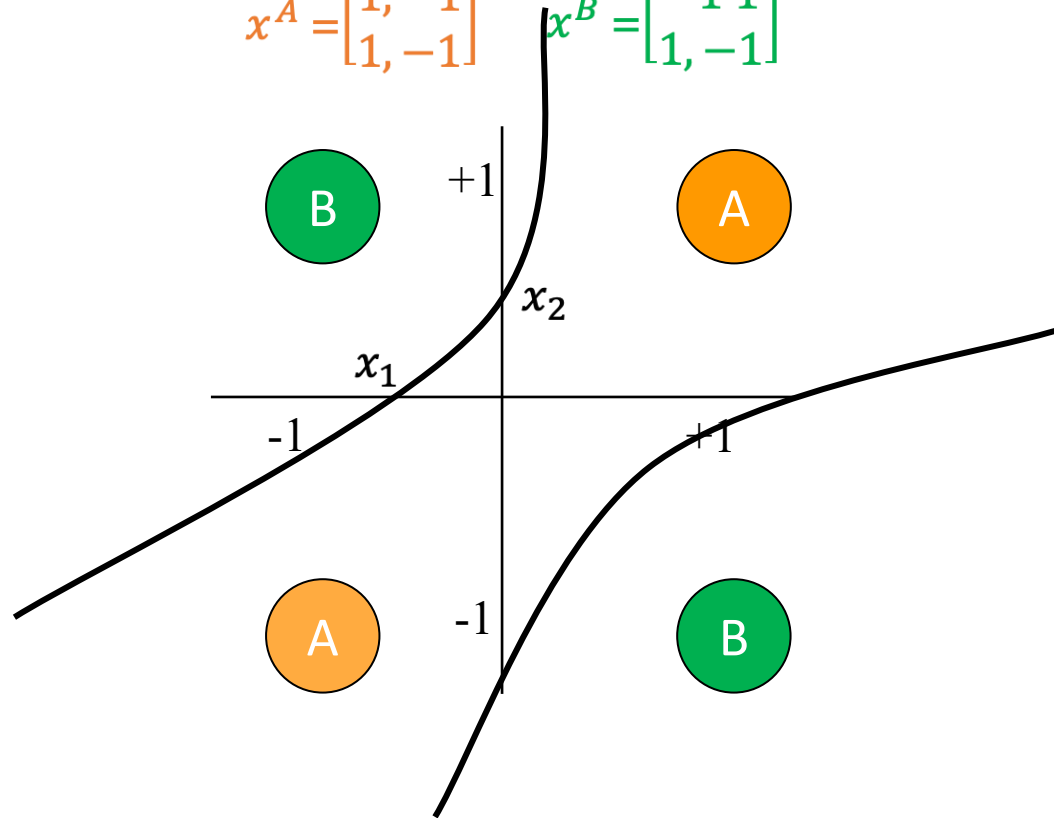


deep nonlinear network

The XOR problem

$$x^A = \begin{bmatrix} 1, -1 \\ 1, -1 \end{bmatrix}$$

$$x^B = \begin{bmatrix} -1 & 1 \\ 1, -1 \end{bmatrix}$$

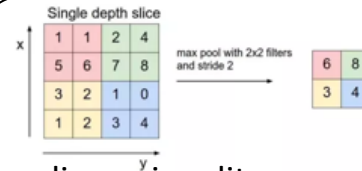




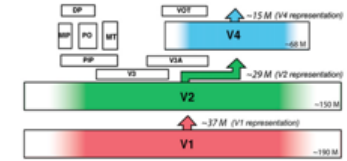
Convolutional neural networks

$$b_i = \frac{a_i}{(k + \alpha \cdot \sum a_j^2)^\beta}$$

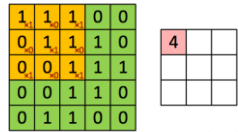
divisive
normalisation



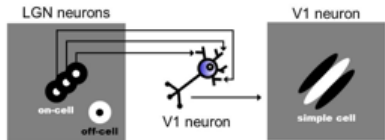
dimensionality
reduction



layerwise
depth



convolution



local receptivity

Conv_1
Convolution
(5 x 5) kernel
valid padding

Max-Pooling
(2 x 2)

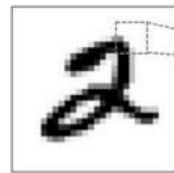
Conv_2
Convolution
(5 x 5) kernel
valid padding

Max-Pooling
(2 x 2)

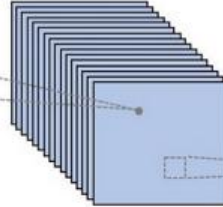
fc_3
Fully-Connected
Neural Network
ReLU activation

fc_4
Fully-Connected
Neural Network

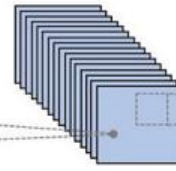
(with dropout)



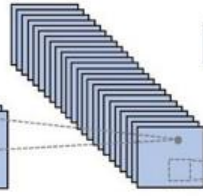
(28 x 28 x 1)



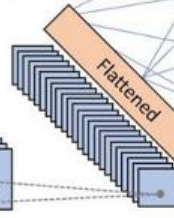
(24 x 24 x n1)



(12 x 12 x n1)



(8 x 8 x n2)



(4 x 4 x n2)



n3 units





```
import torch, torch.nn as nn
```

```
x = torch.tensor([[1.0], [2.0]])
```

```
y = torch.tensor([[3.0], [5.0]])
```

```
model = nn.Linear(1, 1)
```