Learning &
Adaptive Systems

# Neural Networks

Zhenrong Lang[1] and Rajesh Sharma[2]

Introduction to Machine Learning

[1]Department of Information Technology and Electrical Engineering, ETH Zürich
[2]Department of Computer Science, ETH Zürich

# Where we are

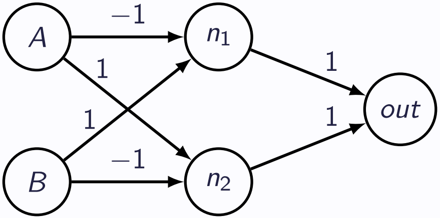
- Lectures
  - Week 5 Kernels
  - Week 6 Neural Networks I, II
  - Week 7 **(This Week!) Neural Networks III, IV**
- Tutorials
  - Week 5 Classification and Model Selection
  - Week 6 Neural Network Basics
  - Week 7 **(Today!) Homework Review, CNN and other Networks**

# Outline

- First session (Zhenrong)
    - Neural Network Learns Boolean Fucntions
    - MLP vs CNN
    - Activation functions
- Second session (Rajesh)
    - **Click here for Notebook**
    - Convolution
    - Convolutional Neural Network
    - Training Loop
    - Prediction and Analysis

# Neural Network Learns Boolean Fucntions

# Neural Network Learns Boolean Fucntions 1

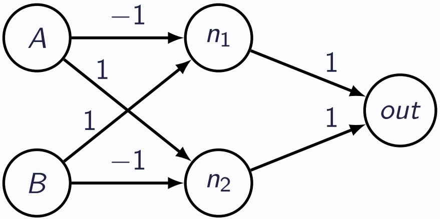



What are weight matrices of layers 1 ($W^{(1)}$) and 2 ($W^{(2)}$)?

$$W^{(1)} \in \mathbb{R}^{2\times 2} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$W^{(2)} \in \mathbb{R}^{1\times 2} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

# Neural Network Learns Boolean Fucntions 2




What Boolean function does this neural network compute?

$$out = W^{(2)}\sigma(W^{(1)}\vec{x})$$

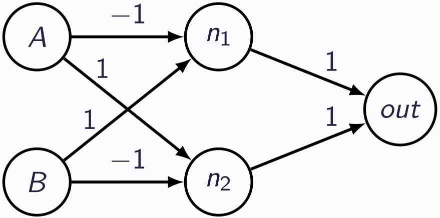
Or in scalar notation:

$$out = \sigma(-A + B) + \sigma(A - B)$$

| $A$ | $B$ | Network |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

The network computes the $XOR$ of $A$ and $B$.

# Neural Network Learns Boolean Fucntions 3




What is $\frac{\partial L}{\partial W_{11}^{(2)}}$?

$$\frac{\partial L}{\partial W_{11}^{(2)}} = \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial W_{11}^{(2)}}$$

$$= \frac{\partial (y - out)^2}{\partial out} \cdot \frac{\partial (W_{11}^{(2)} n_1 + W_{12}^{(2)} n_2)}{\partial W_{11}^{(2)}}$$

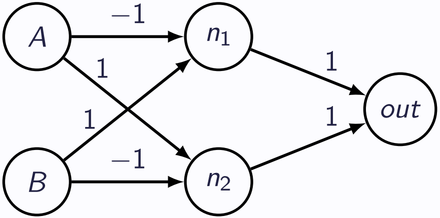
$$= 2(out - y) \cdot n_1$$

$$\frac{\partial L}{\partial W_{12}^{(2)}} = 2(out - y) \cdot n_2$$

$$\frac{\partial L}{\partial W^{(2)}} = \begin{pmatrix} \frac{\partial L}{\partial W_{11}^{(2)}} \\ \frac{\partial L}{\partial W_{12}^{(2)}} \end{pmatrix} = \begin{pmatrix} 2(out - y) \cdot n_1 \\ 2(out - y) \cdot n_2 \end{pmatrix}$$

$$= \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} \cdot 2(out - y)$$

# Neural Network Learns Boolean Fucntions 4




What is $\frac{\partial L}{\partial W_{12}^{(1)}}$?

$$\frac{\partial L}{\partial W_{12}^{(1)}}$$

$$= \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial n_1} \cdot \frac{\partial n_1}{\partial z_1} \cdot \frac{\partial z_1}{W_{12}^{(1)}}$$

$$= \frac{\partial (y - out)^2}{\partial out} \cdot \frac{\partial (W_{11}^{(2)} n_1 + W_{12}^{(2)} n_2)}{\partial n_1}$$

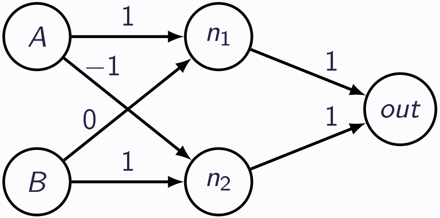
$$\cdot \frac{\partial \sigma(z_1)}{\partial z_1} \cdot \frac{\partial (W_{11}^{(1)} A + W_{12}^{(1)} B)}{W_{12}^{(1)}}$$

$$= 2(out - y) \cdot W_{11}^{(2)} \cdot \sigma'(W_{11}^{(1)} A + W_{12}^{(1)} B) \cdot B$$

# Neural Network Learns Boolean Fucntions 4 (cont.)

$$
\begin{aligned}
\frac{\partial L}{\partial W^{(1)}} &= \begin{pmatrix} \frac{\partial L}{\partial W_{11}^{(1)}} & \frac{\partial L}{\partial W_{21}^{(1)}} \\ \frac{\partial L}{\partial W_{12}^{(1)}} & \frac{\partial L}{\partial W_{22}^{(1)}} \end{pmatrix} \\
&= \begin{pmatrix} 2(out - y) \cdot W_{11}^{(2)} \cdot \sigma'(z_1) \cdot A & 2(out - y) \cdot W_{12}^{(2)} \cdot \sigma'(z_2) \cdot A \\ 2(out - y) \cdot W_{11}^{(2)} \cdot \sigma'(z_1) \cdot B & 2(out - y) \cdot W_{12}^{(2)} \cdot \sigma'(z_2) \cdot B \end{pmatrix} \\
&= 2(out - y) \cdot \begin{pmatrix} A \cdot W_{11}^{(2)} & A \cdot W_{12}^{(2)} \\ B \cdot W_{11}^{(2)} & B \cdot W_{12}^{(2)} \end{pmatrix} \begin{pmatrix} \varphi'(z_1) & 0 \\ 0 & \varphi'(z_2) \end{pmatrix} \\
&= \begin{pmatrix} A \\ B \end{pmatrix} \cdot 2(out - y) \cdot \begin{pmatrix} W_{11}^{(2)} & W_{12}^{(2)} \end{pmatrix} \cdot \begin{pmatrix} \varphi'(z_1) & 0 \\ 0 & \varphi'(z_2) \end{pmatrix}
\end{aligned}
$$

# Neural Network Learns Boolean Fucntions 5




Now, provide weights that implements the logical OR.

$$W^{(1)} \in \mathbb{R}^{2\times 2} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$$

$$W^{(2)} \in \mathbb{R}^{1\times 2} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

| $A$ | $B$ | Network |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# MLP vs CNN

# MLP vs CNN 1

You just took a picture which has $1920 * 1080$ pixels. Each pixel has a red, blue, and green value. You would like to design a neural network for this image.

Calculate the number of parameters needed for a multilayer perceptron (MLP) that has only one hidden layer of 256 nodes and an output layer of 10 nodes.

From input to hidden:

$$\#_{input}^{MLP} = 1920 * 1080 * 3 * 256 + 256 = 1'592'525'056$$

From hidden to output:

$$\#_{output}^{MLP} = 256 * 10 + 10 = 2570$$

The total number of parameters is the sum of both:

$$\#_{tot}^{MLP} = \#_{input}^{MLP} + \#_{output}^{MLP} = 1'592'525'056 + 2570 = 1'592'527'626 \approx 1.6 * 10^9$$

# MLP vs CNN 1 (cont.)

# MLP vs CNN 2

Now you would like to use a convolutional neural network (CNN). The network has 10 layers, each layer has 64 $4 \times 4$ filters for EACH channel. Calculate the number of parameters needed for this CNN.

The first layer receives 3 channels from the input (the RGB channels of the picture) and outputs 64 channels:

$$\#_{first}^{CNN} = (4 * 4 * 3 + 1) * 64 = 3136$$

The rest 9 layers receive 64 channels (the output from the previous layer) and output 64 channels:

$$\#_{rest}^{CNN} = (4 * 4 * 64 + 1) * 64 = 65600$$

The total number of parameters is

$$\#_{tot}^{CNN} = \#_{first}^{CNN} + 9 * \#_{rest}^{CNN} = 593'536$$

Note this is independent of the input size of the picture.

# MLP vs CNN 2 (cont.)

# MLP vs CNN 3

Now consider a simple CNN with only one layer and one $4 \times 4$ filter per channel. What dimensions do the outputs of this layer have, if we choose a stride of 2 and apply 2-pixel padding to the input?

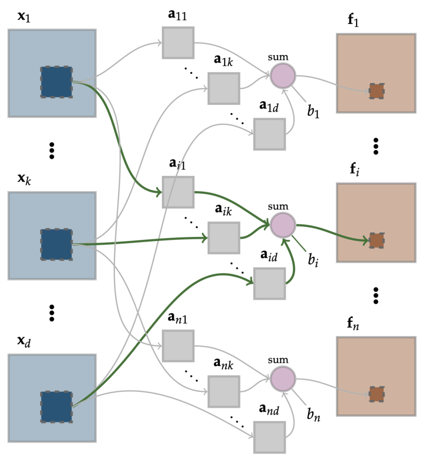
$$\left(\frac{1920 + 2 * padding - filter}{stride} + 1\right) * \left(\frac{1080 + 2 * padding - filter}{stride} + 1\right) * 3$$

$$= \left(\frac{1920 + 2 * 2 - 4}{2} + 1\right) * \left(\frac{1080 + 2 * 2 - 4}{2} + 1\right) * 3$$

$$= 961 * 541 * 3$$

# MLP vs CNN 4

Prove that there exists a fully connected linear layer of input size and output size that is functionally equivalent to the described convolutional network.



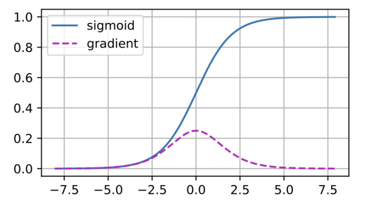
# MLP vs CNN 5

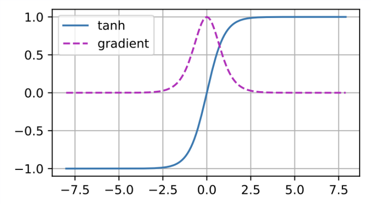
Deduce that the family of functions written as convolutional layers is a subset of those written as fully connected linear layers.

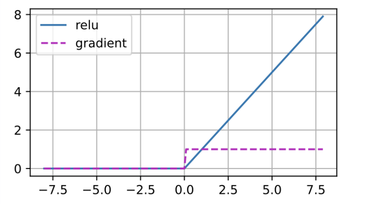
On our previous statement we proved that every convolutional layer can be written equivalently as a fully connected linear layer. In essence, that means that every function that can be expressed with a convolutional layer, can also be expressed with a linear layer.

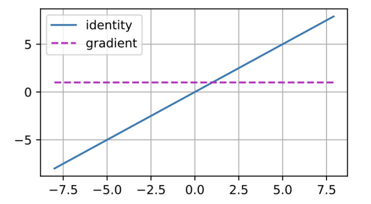
# Activation Functions

# Activation Functions 1



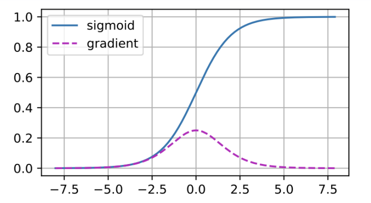



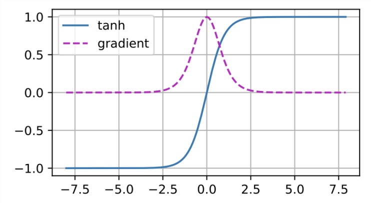



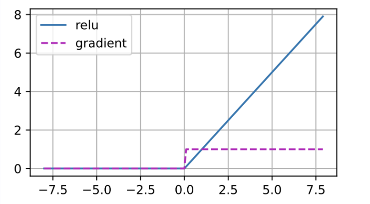

more prone to vanishing gradients?

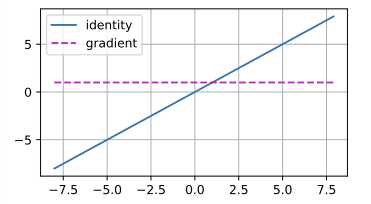
sigmoid and tanh

# Activation Functions 2



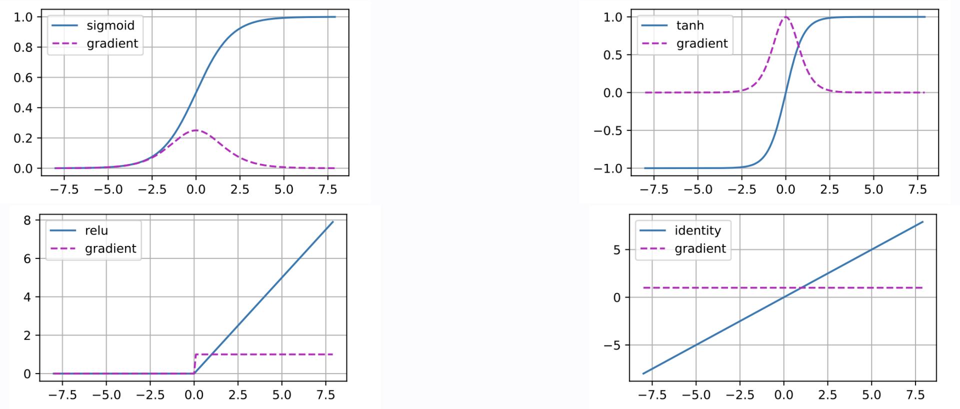




differentiable?

sigmoid, tanh and identity

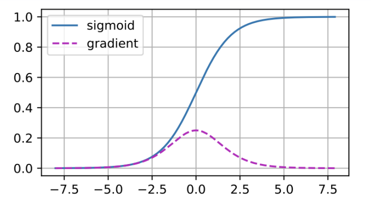
# Activation Functions 3

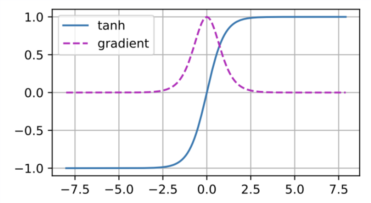


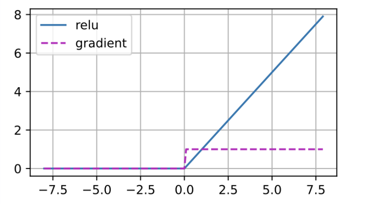

non-linear?

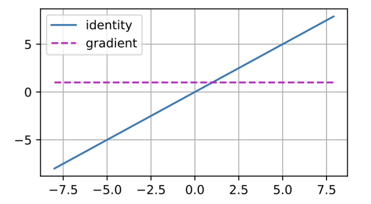
sigmoid, tanh and ReLU

# Activation Functions 4



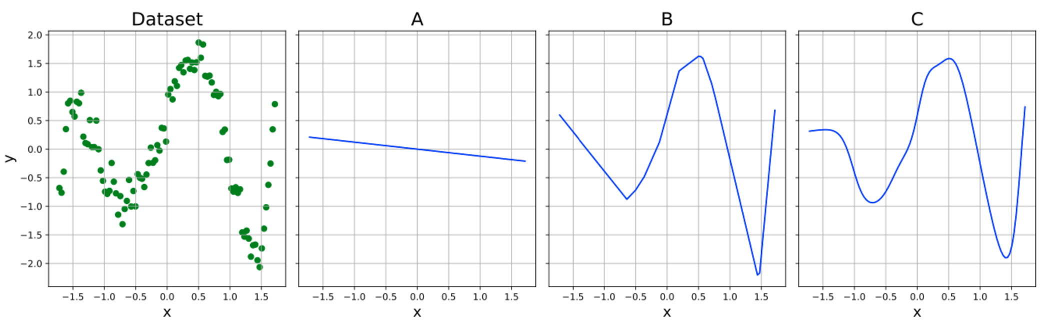




zero-centered?

tanh and identity

# Activation Funcions 5

All neural networks have one hidden layer with 20 units but differ in the choice of the activation function that are either the sigmoid, ReLU or identity function.




(A, B, C) = (identity, ReLU, sigmoid)

# References

**Neural Networks and Deep Learning** by Michael Nielsen

**Dive into Deep Learning** by Aston Zhang

**Deep Learning** by Ian Goodfellow