Machine Learning and Artificial Intelligence

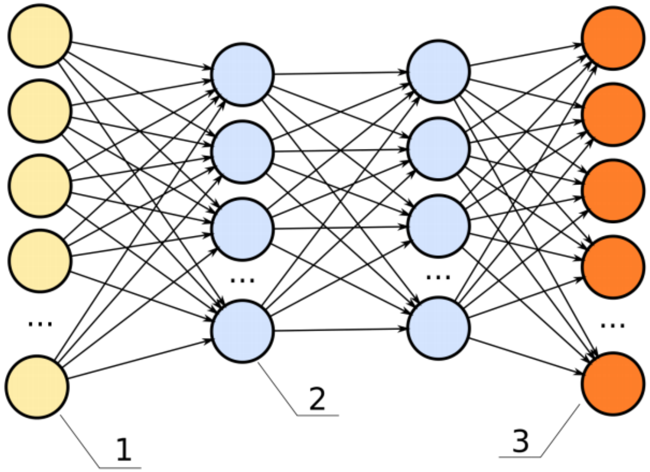
Lab 10 – Introduction to Deep Learning and PyTorch

31/05/2022

(Deep) Neural Networks

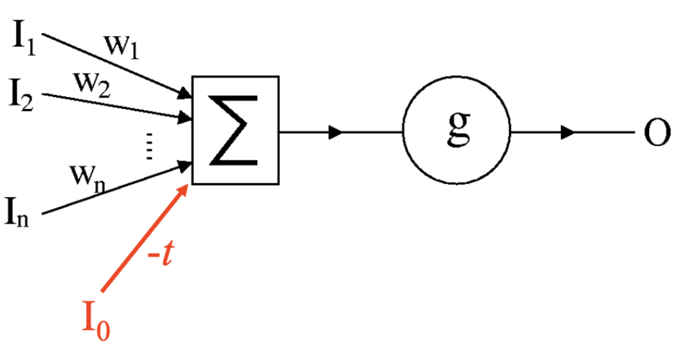
- Neural networks:
  - Are complex structures
  - Composed of many elementary computing units (neurons)
  - Neurons are connected to each other through weighted connections (synapses)

- Neurons are arranged in layers, which can communicate with the outside (input or output) or be internal to the network (hidden) in the case of deep networks.



The Neuron (Perceptron)

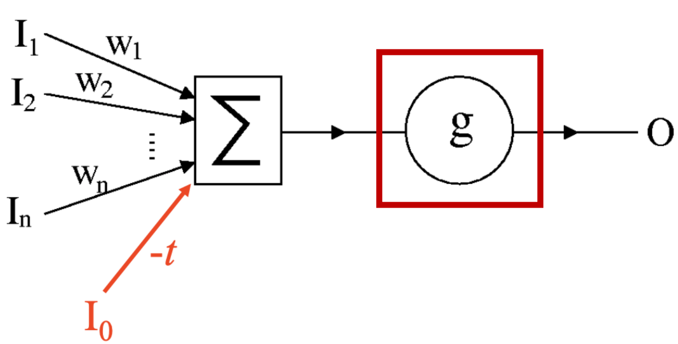
- Input I_i: Information entering the neuron.
- Weights (synapses) w_i: weight of each input to the neuron, provides a measure of how much the input in the neuron counts.
- Summation Σ: module that performs a weighted sum of the inputs
- Activation (transfer) function g: function that determines the output of the neuron based on the output of the summation



$$O = g\left(\sum_{i=1}^{n} w_i I_i - t\right)$$

The Neuron (Perceptron)

- Input I_i: Information entering the neuron.
- Weights (synapses) w_i: weight of each input to the neuron, provides a measure of how much the input in the neuron counts.
- Summation Σ: module that performs a weighted sum of the inputs
- Activation (transfer) function g: function that determines the output of the neuron based on the output of the summation



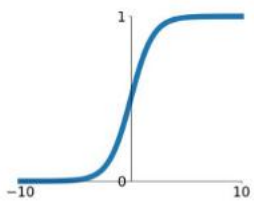
$$O = g\left(\sum_{i=1}^{n} w_i I_i - t\right)$$

Activation functions

- They provide the non-linearity which makes these methods so powerful
- Crucial element of any architecture.

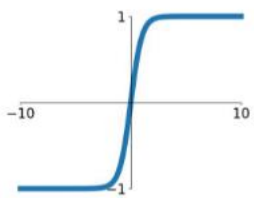
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



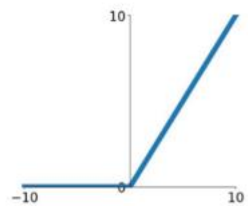
tanh

$\tanh(x)$



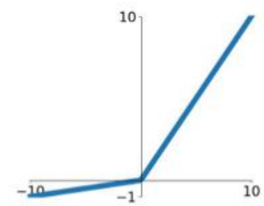
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

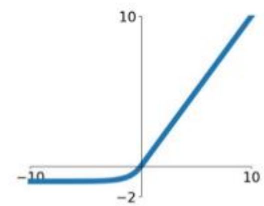


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

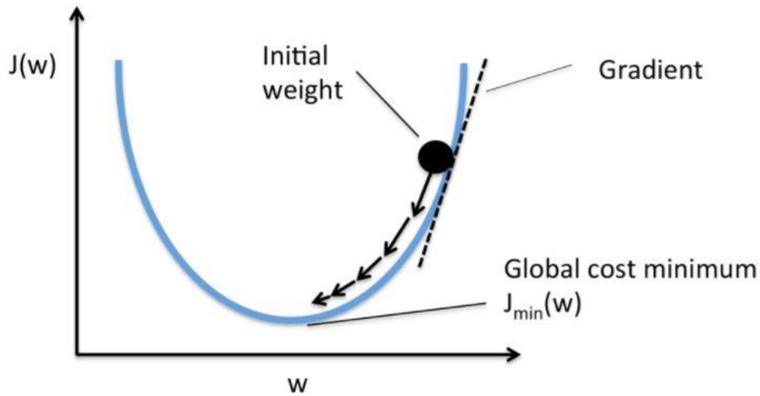


Loss function

- Given the training set $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, the goal is to adjust the weights of the net based on the training examples by minimizing a certain loss (error) function **which is differentiable.**

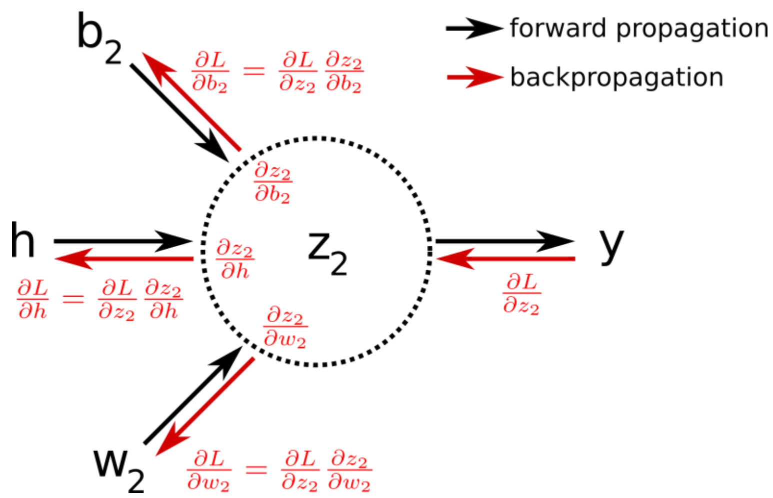
$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \widehat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

- Minimization carried out by **gradient descent**



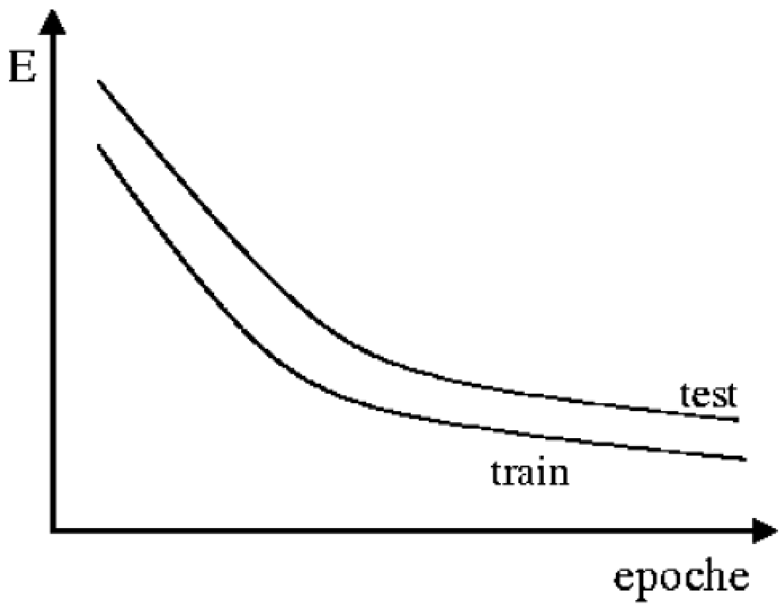
The Backprogation algorithm

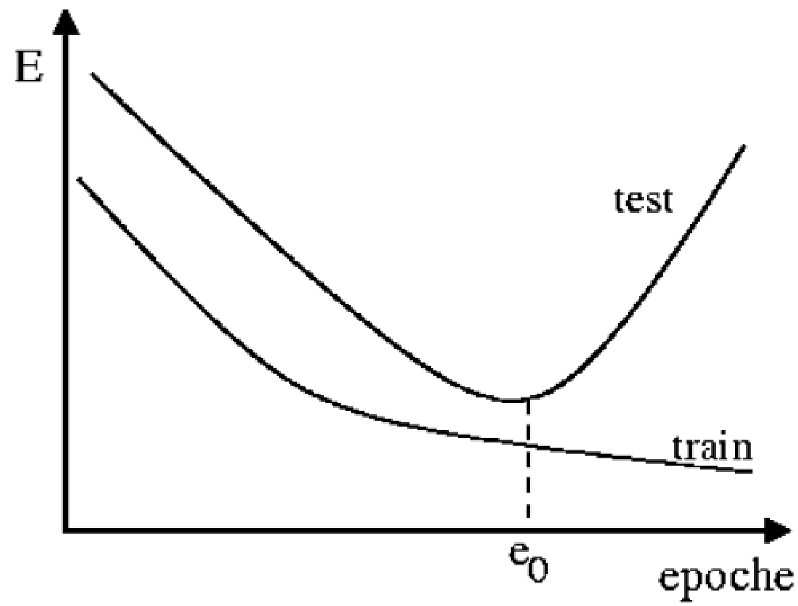
- Neural network training technique.
-
- Based on gradient optimization techniques.
- Optimizes derivative calculation.
- Divided into 2 phases:
  - Forward phase: An example is presented to the network, the output is determined and error is calculated.
  - Backward phase: The error is propagated back into the network, progressively adjusting the weights.



When do we stop training?

- A network is considered trained when the loss function converges at a low value.
- Beware of over-fitting!





When should we use NNs?

- The use of neural networks is recommended if:
  - You have **lots** of (preferably labeled) data.
  - Long training times are accepted.
  - It is not important that the determined decision function is interpretable by a human.
  - The task you are trying to solve is relatively complex and you have already tried classic Machine Learning methods. This also means knowing which architecture is best for your task: FCNs, CNNs, RNNs etc

PyTorch

- Open-source python library developed by Meta (Facebook), dedicated to the development of Deep Learning models.

- Why PyTorch?
  - Easy to use and allows for both high- and low-level implementations
  - Has strong support with GPUs and TPUs
  - Many algorithms are already implemented
  - Similar to NumPy

PyTorch vs NumPy

```python
import torch
torch.tensor([[2, 3, 5], [1, 2, 9]])
```

```
tensor([[ 2,  3,  5],
        [ 1,  2,  9]])
```

```python
torch.rand(2, 2)
```

```
tensor([[ 0.0374, -0.0936],
        [ 0.3135, -0.6961]])
```

```python
a = torch.rand((3, 5))
a.shape
```

```
torch.Size([3, 5])
```

```python
import numpy as np
np.array([[2, 3, 5], [1, 2, 9]])
```

```
array([[ 2,  3,  5],
       [ 1,  2,  9]])
```

```python
np.random.rand(2, 2)
```

```
array([[ 0.0374, -0.0936],
       [ 0.3135, -0.6961]])
```

```python
a = np.random.randn(3, 5)
a.shape
```

```
(3, 5)
```

Matrix operations

```
a = torch.rand((2, 2))
b = torch.rand((2, 2))
```

```
tensor([[-0.6110,  0.0145],
        [ 1.3583, -0.0921]])
tensor([[ 0.0673,  0.6419],
        [-0.0734,  0.3283]])
```

```
torch.matmul(a, b)
```

```
tensor([[-0.0422, -0.3875],
        [ 0.0981,  0.8417]])
```

```
a * b
```

```
tensor([[-0.0411,  0.0093],
        [-0.0998, -0.0302]])
```

```
a = np.random.rand(2, 2)
b = np.random.rand(2, 2)
```

```
array([[-0.6110,  0.0145],
        [ 1.3583, -0.0921]])
array([[ 0.0673,  0.6419],
        [-0.0734,  0.3283]])
```

```
np.dot(a, b)
```

```
array([[-0.0422, -0.3875],
       [ 0.0981,  0.8417]])
```

```
np.multiply(a, b)
```

```
array([[-0.0411,  0.0093],
        [-0.0998, -0.0302]])
```

Examples of what we have seen

```
torch.matmul(a, b)   # multiples torch tensors a and b

*                    # element-wise multiplication between two torch tensors

torch.eye(n)         # creates an identity torch tensor with shape (n, n)

torch.zeros(n, m)    # creates a torch tensor of zeros with shape (n, m)

torch.ones(n, m)     # creates a torch tensor of ones with shape (n, m)

torch.rand(n, m)     # creates a random torch tensor with shape (n, m)
```

Exercise

We will go through one of official PyTorch tutorials available [here]