



UNIVERSIDAD
DE GRANADA



Reinforcement Learning: Neural Networks

September 25th – 29th, 2023

E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Manuel Pegalajar Cuellar / Juan Gómez Romero

manupc@ugr.es / jgomez@ugr.es

Departamento de Ciencias de la
Computación e Inteligencia Artificial
<http://decsai.ugr.es>

Este documento está protegido por la Ley de
Propiedad Intelectual ([Real Decreto Ley
1/1996 de 12 de abril](#)).
Queda expresamente prohibido su uso o
distribución sin autorización del autor.



UNIVERSIDAD
DE GRANADA

Reinforcement Learning:

Neural Networks

Contents



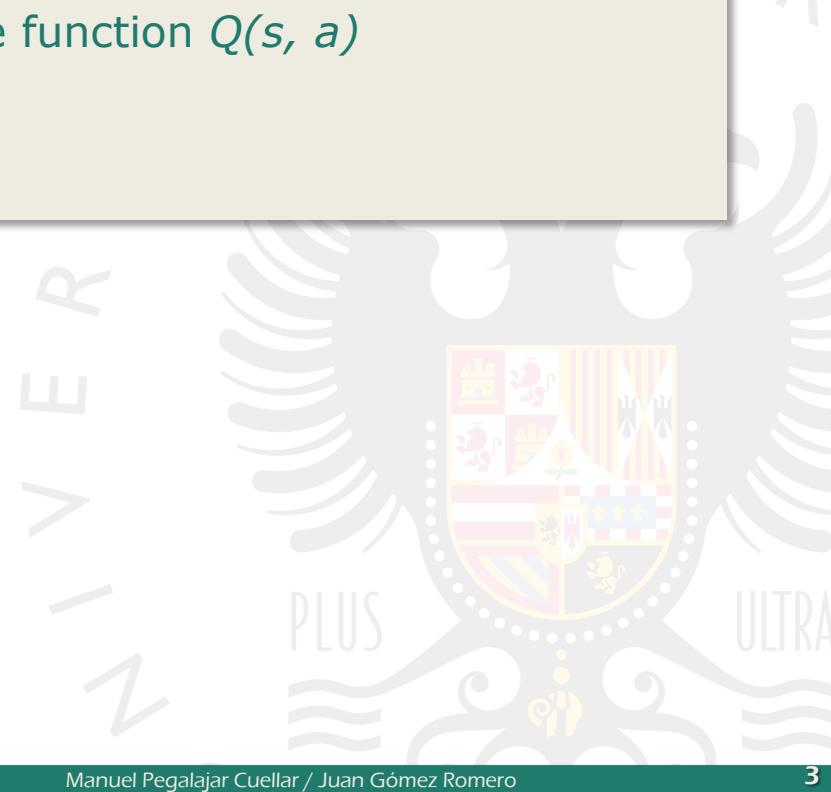
- 1. Introduction**
- 2. Training**
- 3. Implementation**



DECSAI

What we know so far

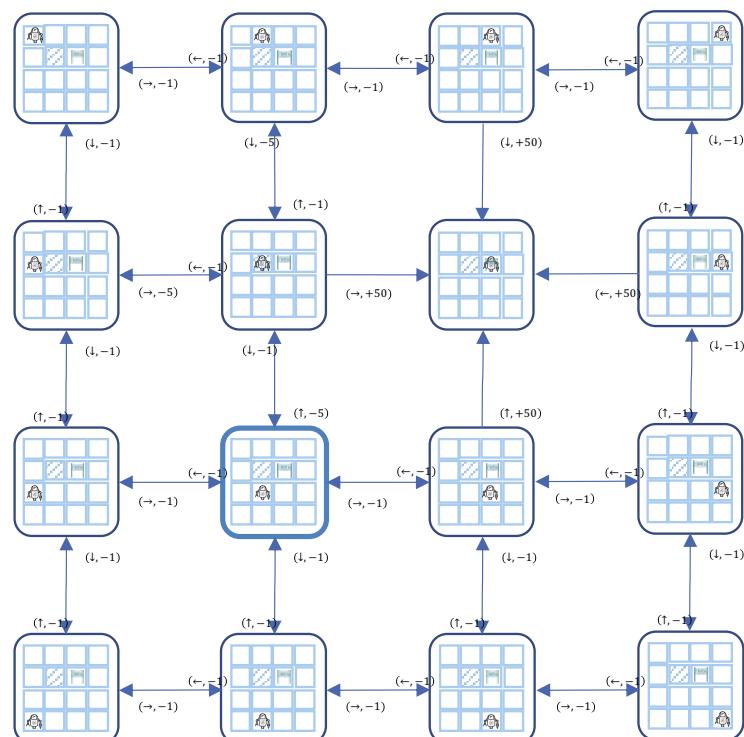
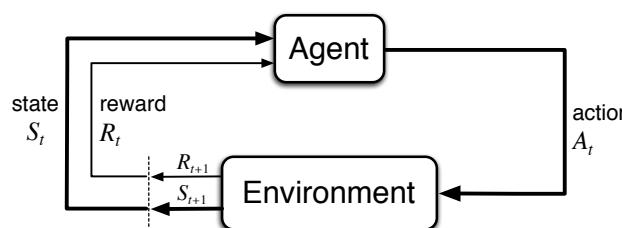
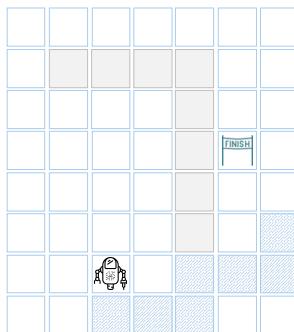
- Fundamental concepts of RL: state, action, reward, policy
- Formulation of a Markov Decision Process
- The value function $V(s)$ and state-value function $Q(s, a)$
- Value iteration algorithm
- Q-Learning algorithm



Reinforcement Learning

Find paths in the state/transition graph

Optimize the future accumulated reward (*return*)



Q-Learning

Estimate $Q(s, a)$

State	Action	Q		
		A_1	\dots	A^m
S^1				
\dots				
S^n				

Limitations with continuous action or state functions

Experience is generated sequentially

From interactions with the environment

$$S_0 A_0 R_1 S_1 \quad | \quad A_1 R_2 S_2$$

→
$$Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \alpha \left(R_1 + \gamma \max_{a \in \mathcal{A}} Q(S_1, a) - Q(S_0, A_0) \right)$$

$$Q(S_1, A_1) \leftarrow Q(S_1, A_1) + \alpha \left(R_2 + \gamma \max_{a \in \mathcal{A}} Q(S_2, a) - Q(S_1, A_1) \right)$$

The action can be selected based on a policy (on-policy) or by maximizing the Q (off-policy)

A random action may be selected with probability ϵ (epsilon-greedy)

Deep Q-Learning

Model $Q(s, a)$ as a (non-linear) function

$$Q(s, a) \rightarrow \mathbb{R}$$

Q can be approximated with a parametric model
 $Q(s, a; \theta)$

Parameters of $Q(s, a; \theta)$ can be adjusted to samples

$$S_0 \ A_0 \ R_1 \ S_1 \rightarrow \hat{Q}(S_0, A_0)$$

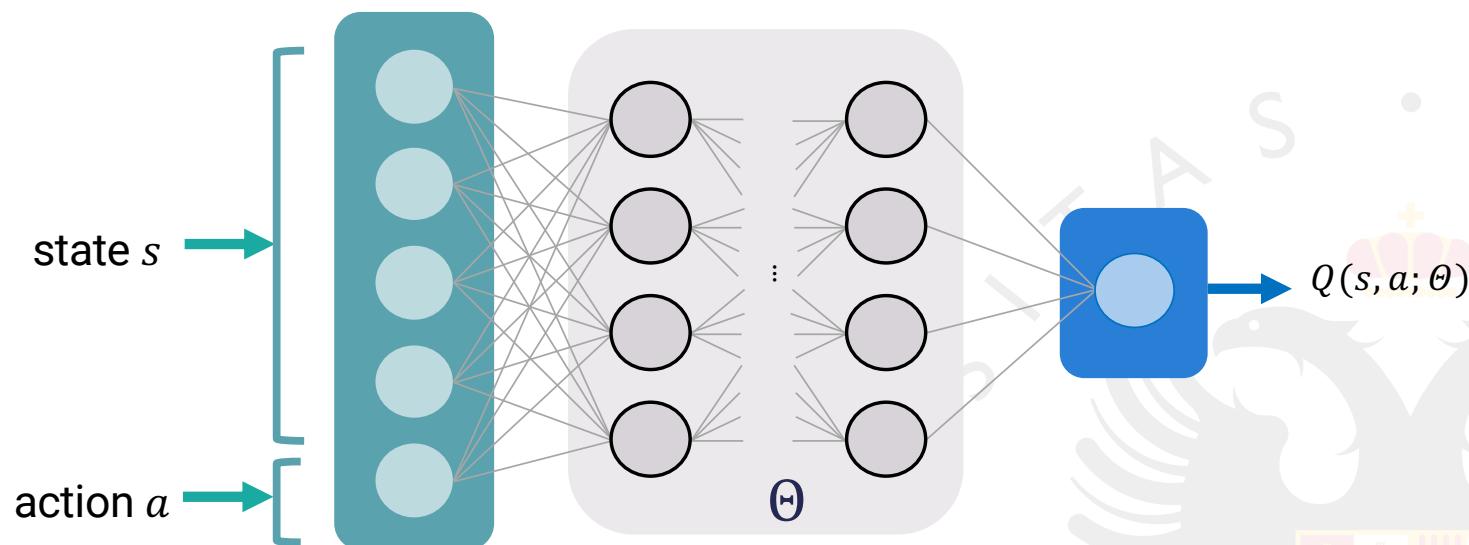
$$S_1 \ A_1 \ R_2 \ S_2 \rightarrow \hat{Q}(S_1, A_1)$$

...

We can keep an experience memory with the Q values calculated at each step

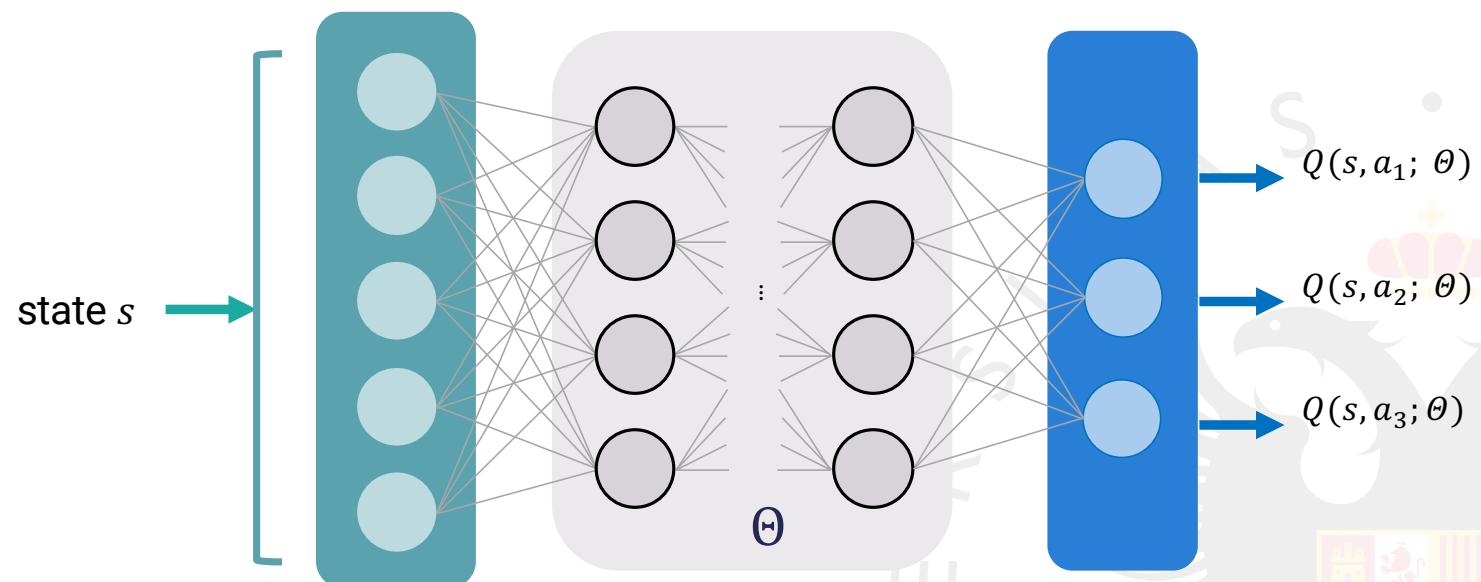
Deep Q-Learning

Feed forward neural network



Deep Q-Learning

Feed forward neural network





UNIVERSIDAD
DE GRANADA

Reinforcement Learning:

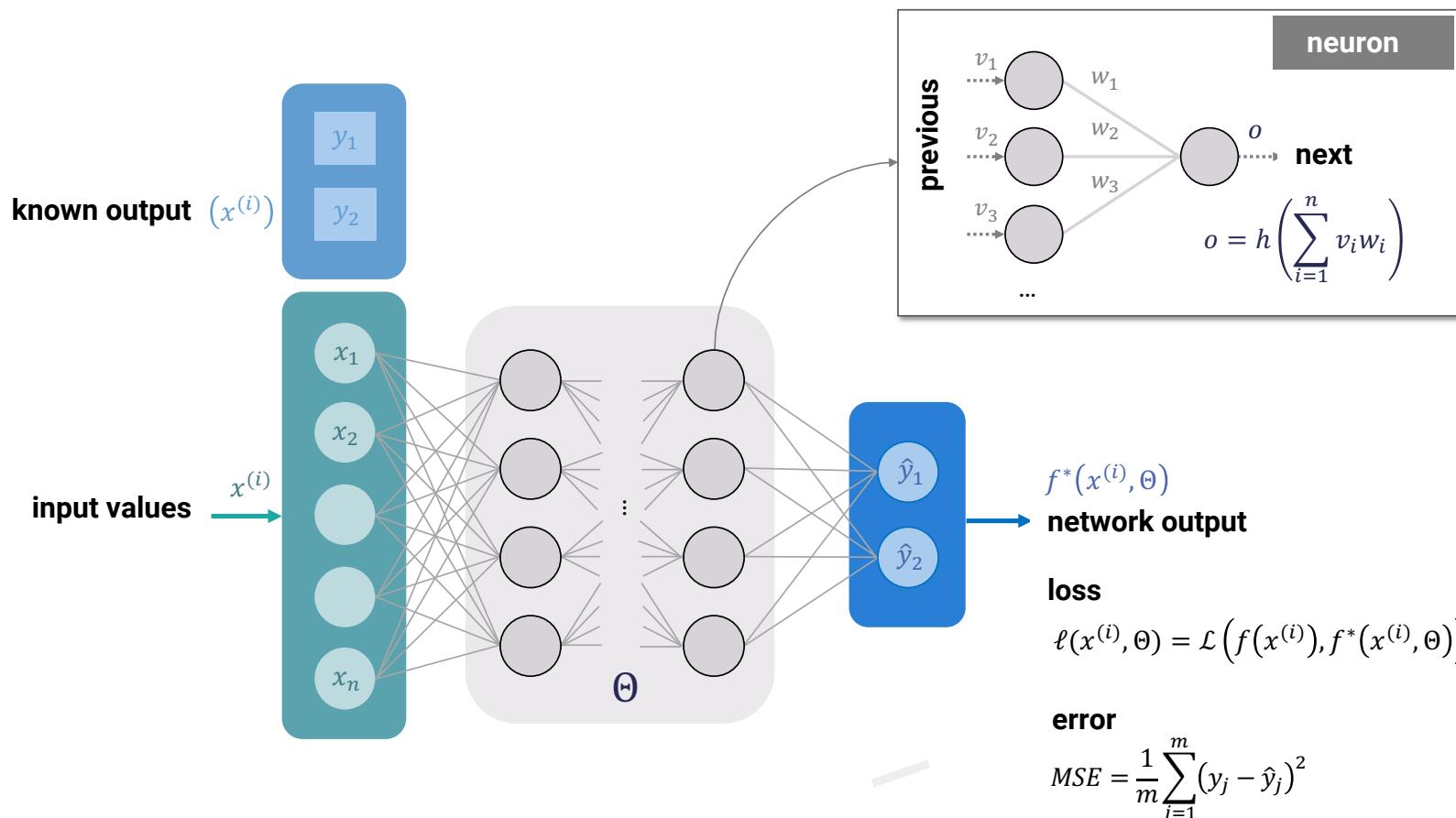
Neural Networks

Contents

1. Introduction
- » 2. Training
3. Implementation

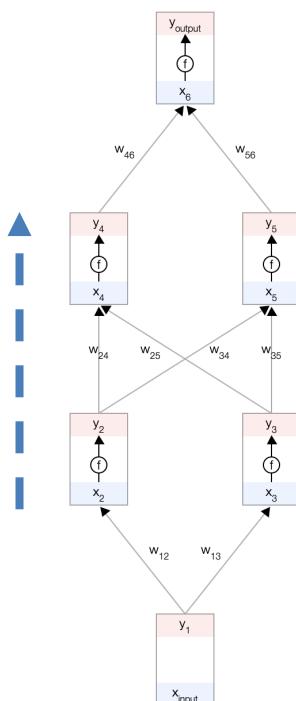


Architecture of feed-forward neural networks

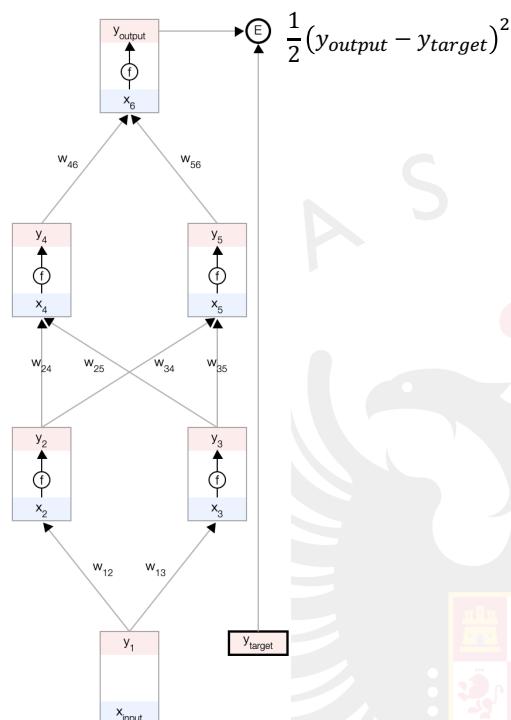


Output & error calculation

<https://developers.google.com/machine-learning?hl=en>



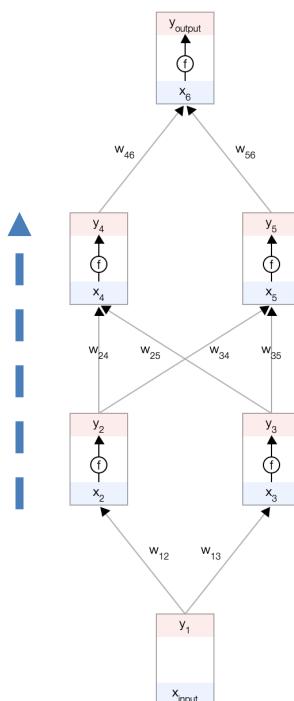
forward pass



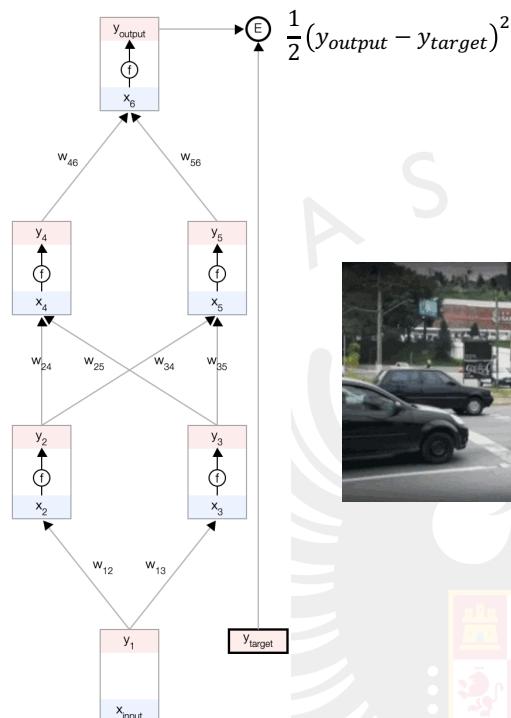
error calculation

Weight adjustment

<https://developers.google.com/machine-learning?hl=en>



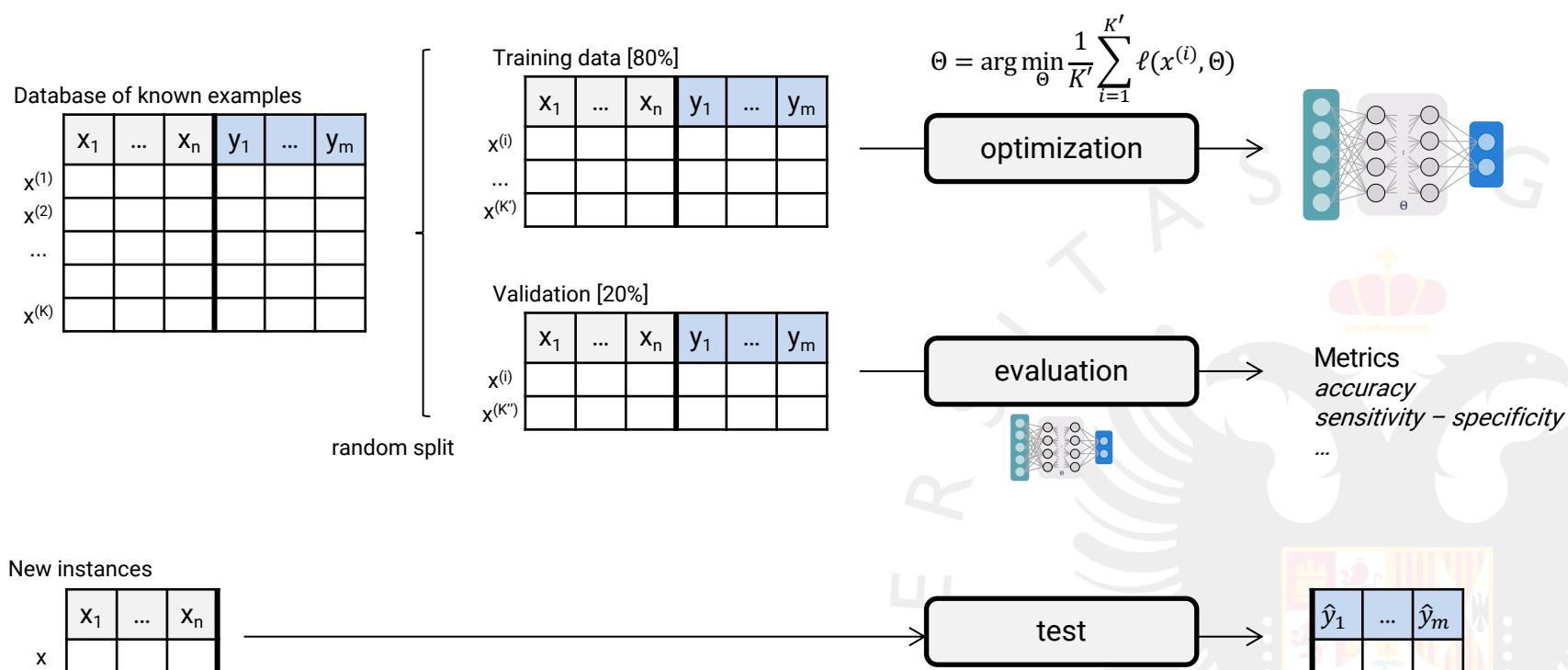
forward pass



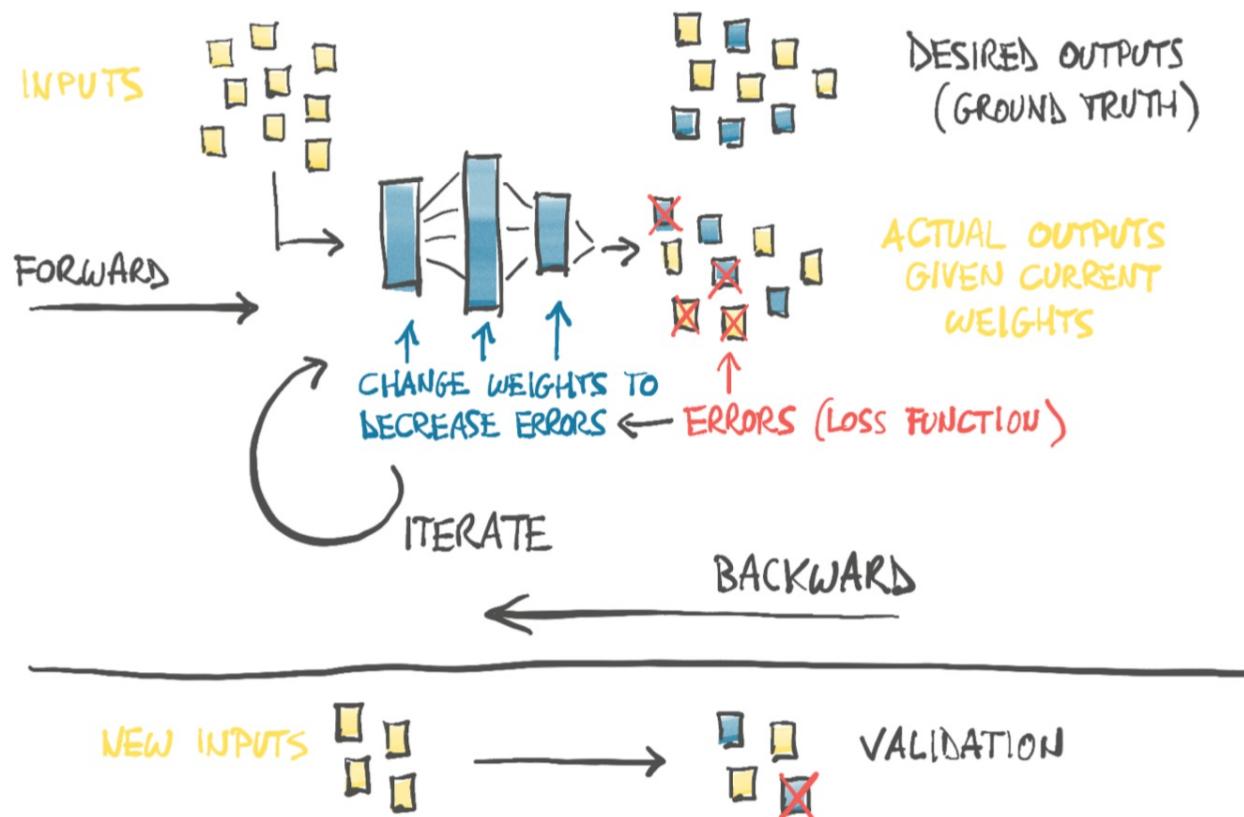
error calculation



Training methodology

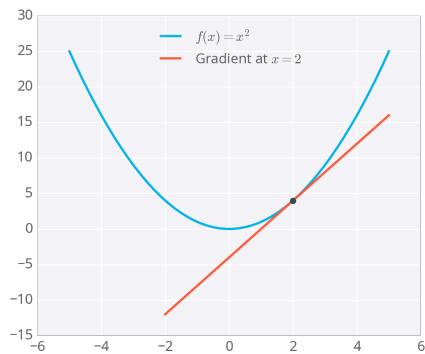


THE LEARNING PROCESS



E. Stevens, L. Antiga (2019) **Deep Learning with PyTorch**. Manning.

Gradient descent

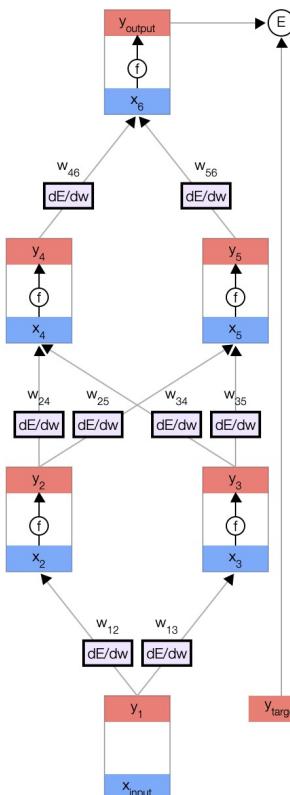


$$\nabla E = \frac{\partial f(w)}{\partial w}$$

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}}$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Backpropagation



$$\begin{aligned}\frac{\partial E}{\partial w_{ij}} &= \frac{\partial \left(\frac{1}{2} (y - \hat{y}(w))^2 \right)}{\partial w_{ij}} \\ &= -(y - \hat{y}) h'(x_j) y_i\end{aligned}$$



UNIVERSIDAD
DE GRANADA

Reinforcement Learning:

Neural Networks

Contents

1. Introduction
2. Neural networks
3. Implementation



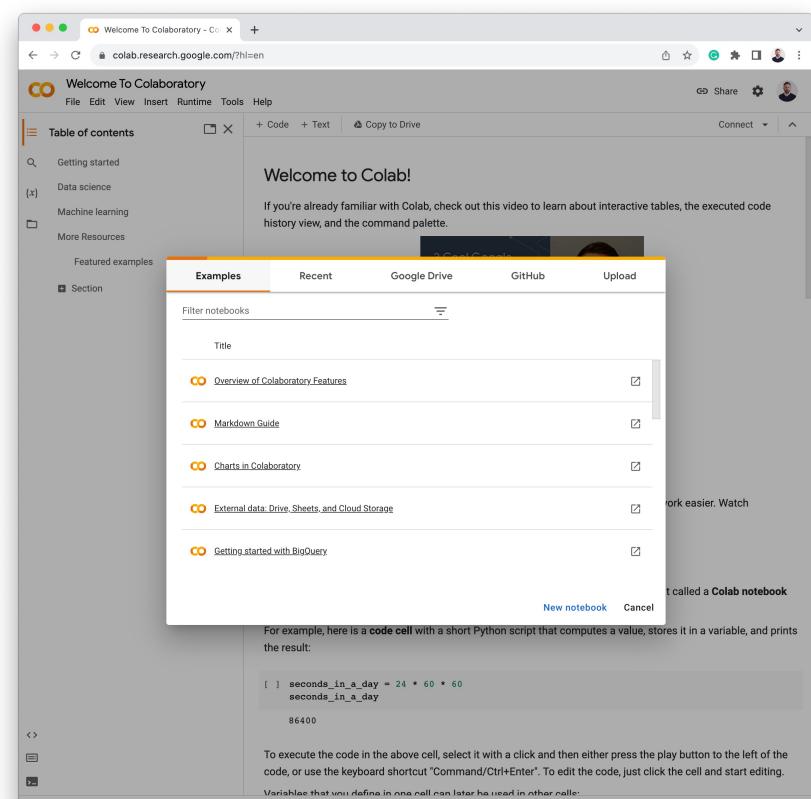
Google Colaboratory

 <https://colab.research.google.com>

- Cloud-based environment for notebooks *a la* Jupyter (code + text)
- Write Python code, storage in Google account
- GPU-enabled execution
- PyTorch is pre-installed

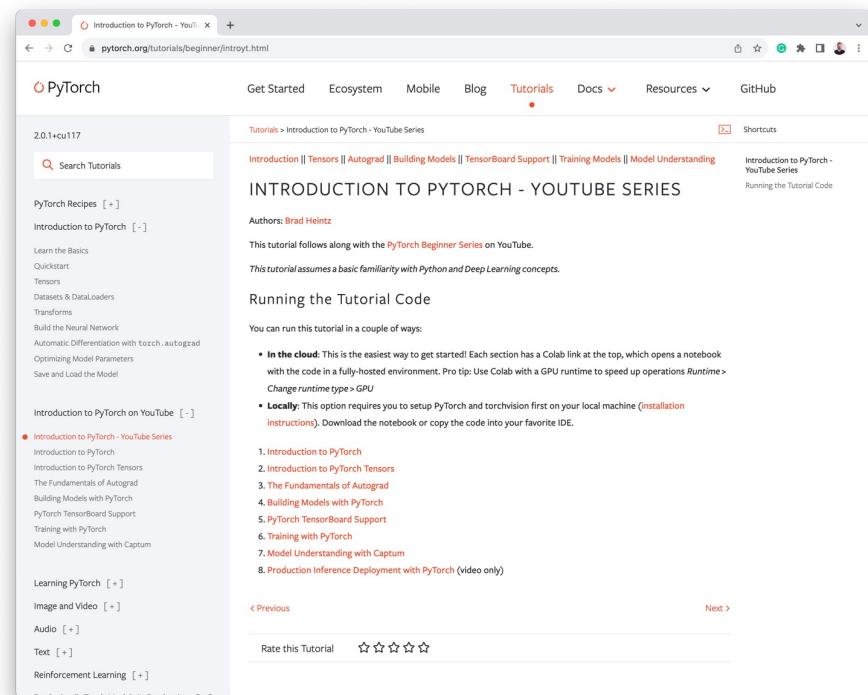
 Open in Colab

https://github.com/jgromero/rl_seminar_2023/code

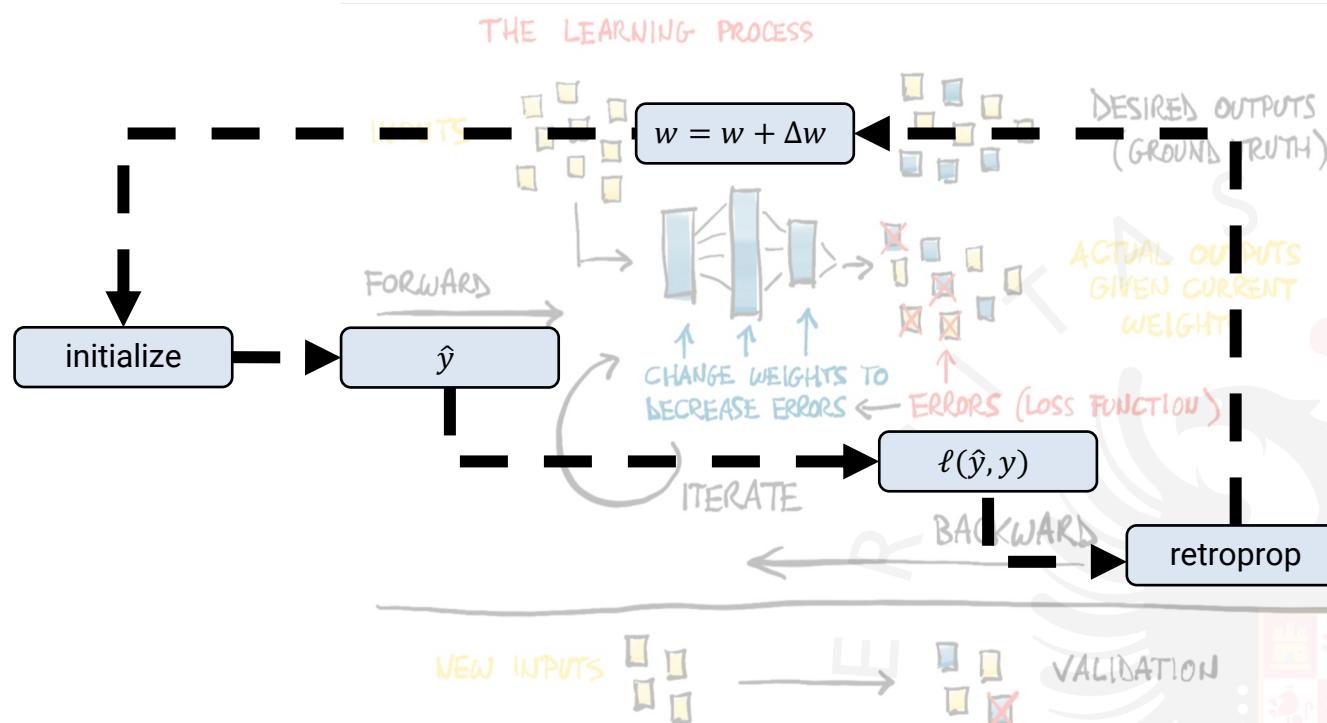


PyTorch

- Python library for machine learning with neural networks
- Based on *tensors* and *computation graphs*
- Automatic differentiation & optimization
- GPU-enabled
- Repository of pre-trained models

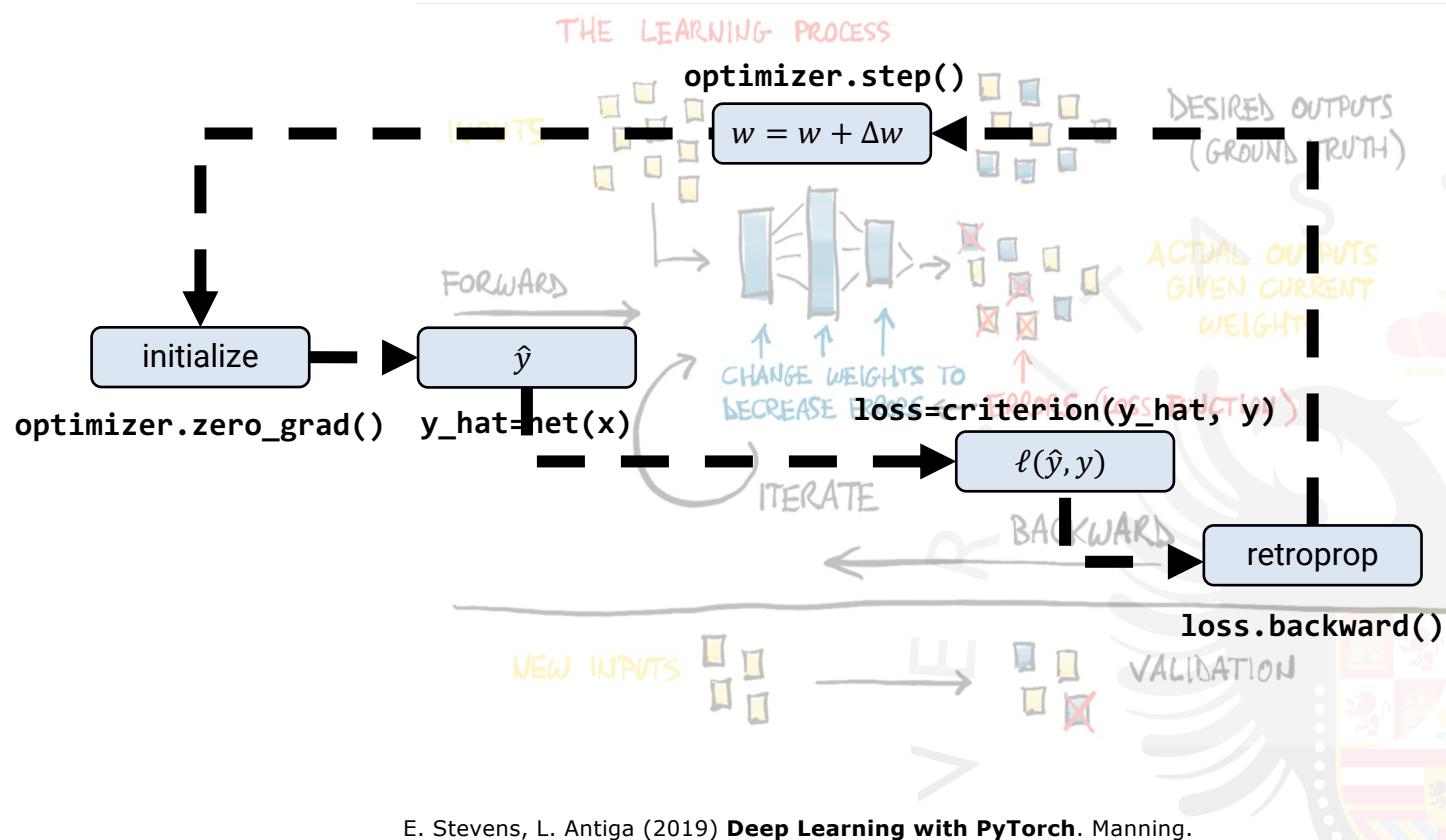


<https://pytorch.org/tutorials/beginner/introyt.html>
[images]

 PyTorch

E. Stevens, L. Antiga (2019) **Deep Learning with PyTorch**. Manning.

PyTorch



E. Stevens, L. Antiga (2019) **Deep Learning with PyTorch**. Manning.

 PyTorch

Example: classification with the *iris* dataset

https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

Predict flower class from shape measurements

R. Fisher (1936) *The use of multiple measurements in taxonomic problems*

Data

3 classes: iris setosa, iris virginica, iris versicolor

50 samples of each class

4 features: length and width of petals and sepals

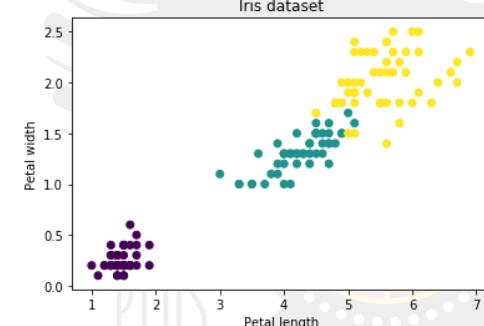
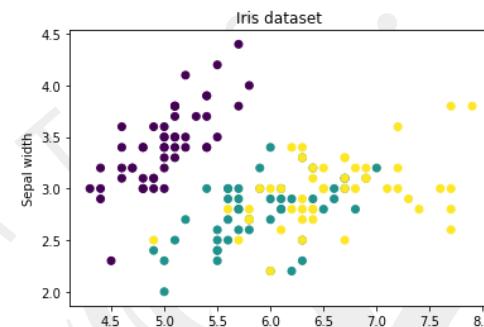
Target: find the class of an unknown sample given the values of the 4 features

PyTorch

Example: classification with the *iris* dataset

https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

sepal length	sepal width	petal length	petal width	class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
7.0	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4.0	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor
6.7	2.5	5.8	1.8	Iris-virginica
7.2	3.6	6.1	2.5	Iris-virginica
6.5	3.2	5.1	2.0	Iris-virginica
6.4	2.7	5.3	1.9	Iris-virginica
6.8	3.0	5.5	2.1	Iris-virginica
...				

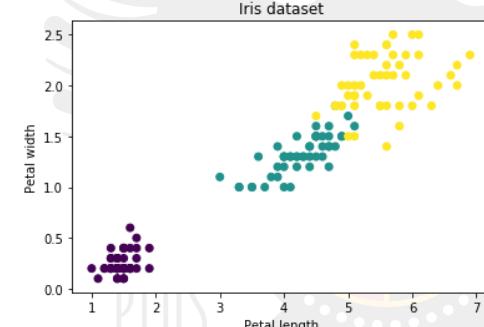
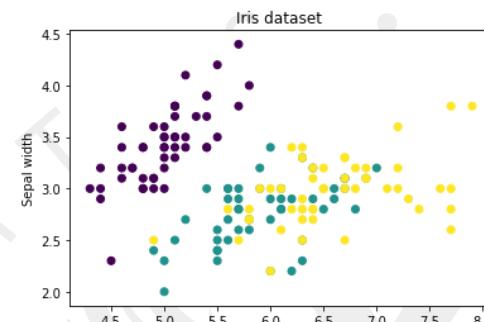


PyTorch

Example: classification with the *iris* dataset

https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

sepal length	sepal width	petal length	petal width	class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
7.0	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4.0	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor
6.7	2.5	5.8	1.8	Iris-virginica
7.2	3.6	6.1	2.5	Iris-virginica
6.5	3.2	5.1	2.0	Iris-virginica
6.4	2.7	5.3	1.9	Iris-virginica
6.8	3.0	5.5	2.1	Iris-virginica
...				



PyTorch

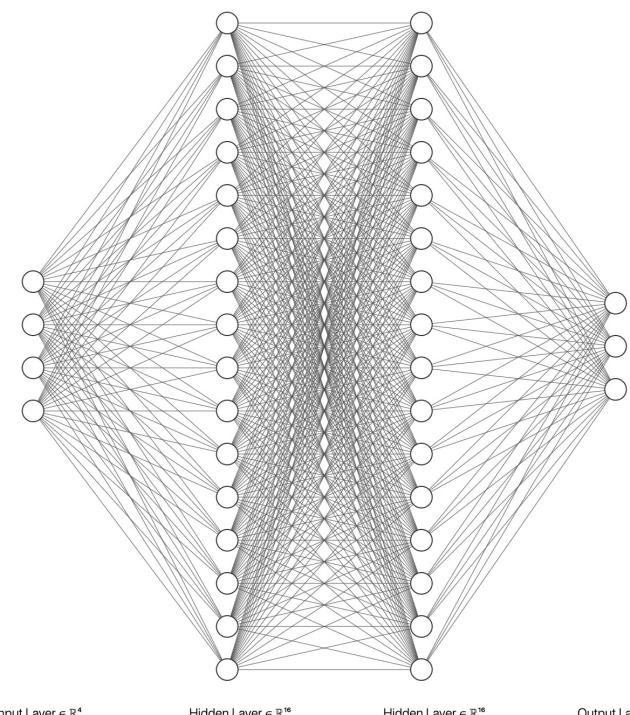
Example: classification with the *iris* dataset

https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

1. Load dataset
2. Define network architecture
3. Split training and validation datasets
4. Train network with train split
5. Validate with validation Split

Hyperparameters

- batch size: 50
- epochs: 1000



PyTorch

Example: classification with the *iris* dataset

https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

iris.ipynb

Archivo Editor Ver Insertar Entorno de ejecución Herramientas Ayuda No se pueden guardar cambios

+ Código + Texto Copiar en Drive

Classification with the iris dataset

Juan Gómez Román

Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada

This work is licensed under the [GNU General Public License v3.0](#).

Activate GPU

Runtime > Change runtime type

[] ▾ 1 célula ejecutada

Load iris data

Load iris from scikit-learn.

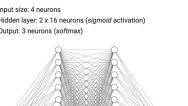
- Features: input (150 x 4)
- Labels: output (150 x 1)

[] ▾ 3 células ejecutadas

Define network architecture

The architecture is quite simple:

- Input size: 4 neurons
- Hidden layer: 2 x 10 neurons (sigmoid activation)
- Output: 3 neurons (softmax)



completado a las 20:14

 Open in Colab

https://github.com/igromero/rl_seminar_2023/code

iris-nn.ipynb

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Colab research.google.com
- Toolbar:** Archivo, Editor, Ver, Insertar, Entorno de ejecución, Herramientas, Ayuda, No se pueden guardar cambios.
- Header:** Colab, nn(jpylab)
- Menu:** Archivo, Editor, Ver, Insertar, Entorno de ejecución, Herramientas, Ayuda, No se pueden guardar cambios.
- Cell 1:** Train network with `train_set`
Defines loss function, optimization algorithm and other hyperparameters.

```
[1]: import torch.optim  
  
criterion = nn.CrossEntropyLoss()  
optimizer = torch.optim.SGD(net.parameters(), lr=0.1)  
  
batch_size = 10  
n_epochs = 1000
```
- Cell 2:** Launch training.
Creates a dataloader and iterates through epochs.

```
[2]: train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=True)  
  
for epoch in range(n_epochs):  
    epoch_loss = 0.0  
  
    for i, data in enumerate(train_loader, 0):  
        x, y = data  
  
        x = x.to(device)  
        y = y.to(device)  
  
        optimizer.zero_grad() # reset gradients  
  
        y_hat = net(x) # calculate forward output  
        loss = criterion(y_hat, y) # calculate loss  
        loss.backward() # calculate backpropagation  
        optimizer.step() # optimize weights  
  
        epoch_loss += loss.item()  
  
    if epoch % 100 == 0:  
        print(f'Epoch {epoch+1}/{n_epochs}, epoch loss: {loss.item():.3f} ({float(epoch_loss) / batch_size})')
```
- Output:** Shows the training progress for 1000 epochs, with loss values decreasing over time.