Deep learning with PyTorch

Francesca Manni

Agenda

Deep learning

PyTorch

Beyond conventional learning

Time for hands-on session

Introduction

Goals of today:

- Starting from basic
- Understand complexity
- Sharing knowledge
- Learning by doing

About me

Francesca

AI Scientist

- Data & AI Scientist at Philips Research
- PhD in AI for medical imaging
- MSc in Biomedical Engineering

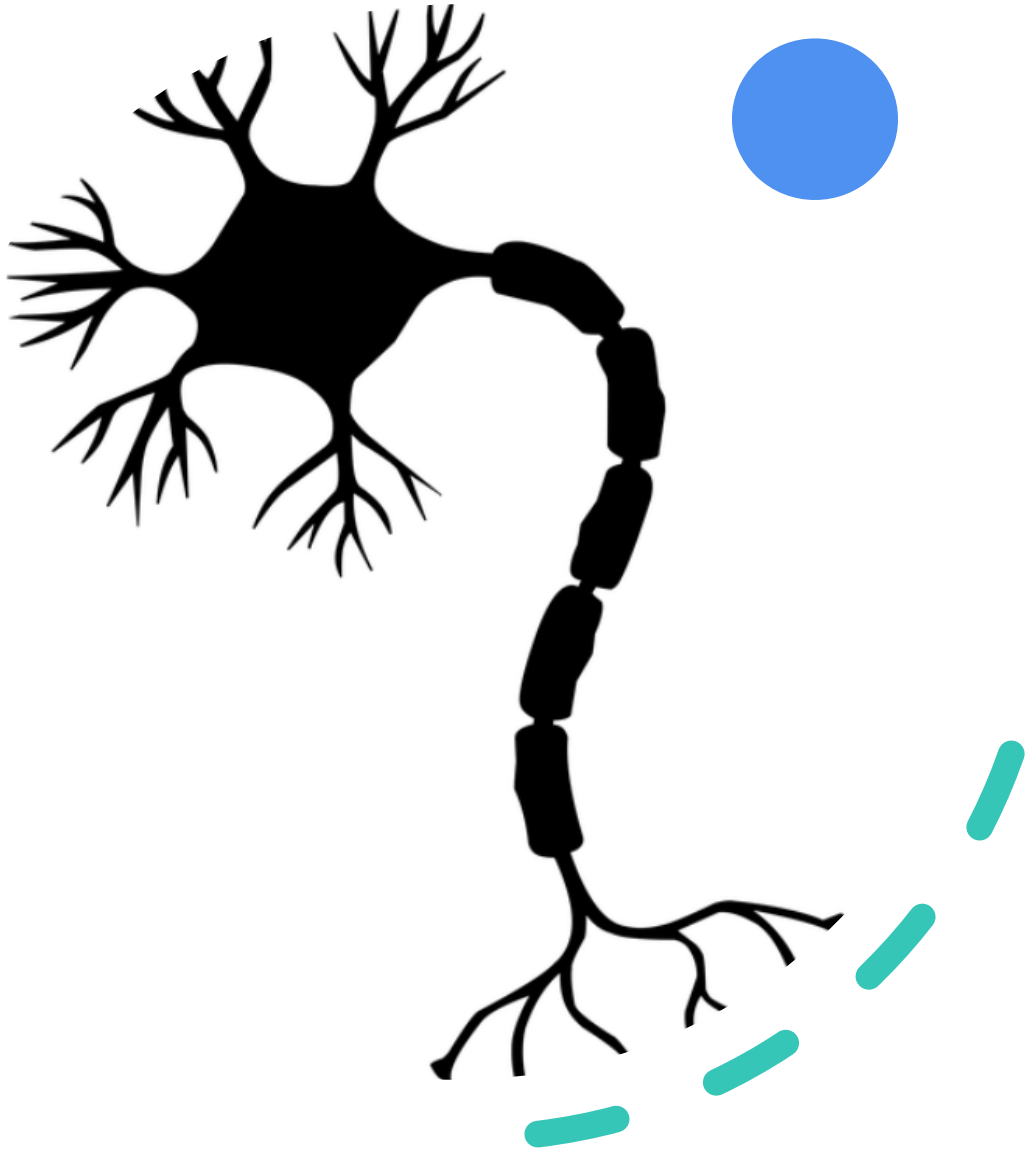
- Passion for healthcare
- A lot of curiosity ☺
- …and I am Italian!

Deep Learning

What is PyTorch?

DL basis

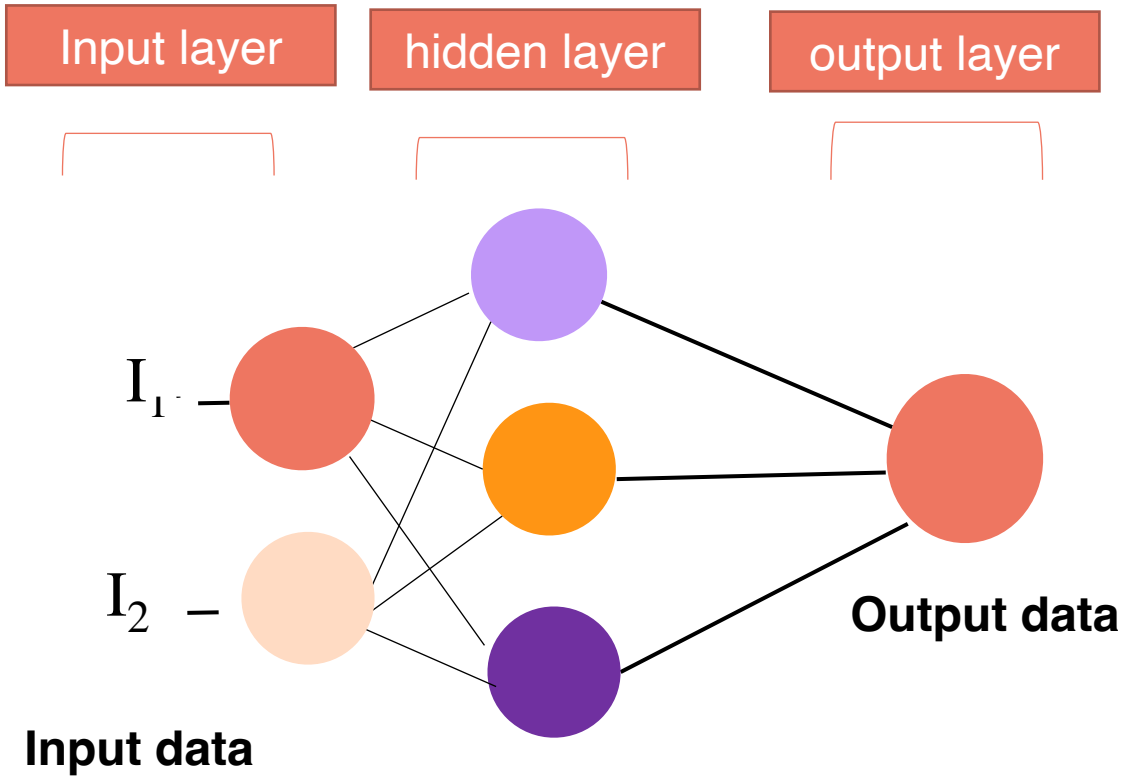
- DL is a machine learning technique
- It injects inputs through layers to learn a prediction
- Inputs can be images, text, signals, etc.
- Inspired by how human brain filters and process information.



DL basis

In DL models we have connected layers.

1- The first layer is called the Input Layer
2- The last layer is called the Output Layer
3 - All layers in between are called Hidden Layers.
NB: When a network is deep neural network it means that we might have more than two layers!



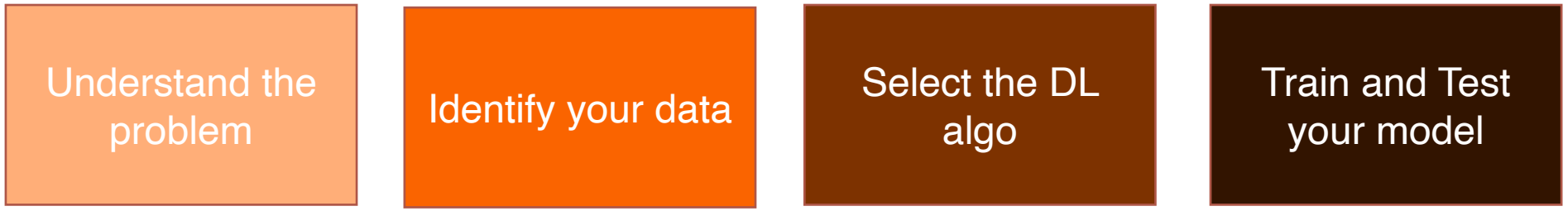
DL basis

How does it learn? The neural net applies a non-linear transformation to the input and create an output (model). Afterwards, this model is improved by back-prop (derivatives). This is repeated many many times until a good level of accuracy is reached.

- The simplest network is the **feed-forward neural networks**. The information's flows starts at the input layer, goes to the "hidden" layers, and end at the output layer. The network

- More complex networks are e.g**. convolutional neural network**: typical for unstructured data (like images), it uses a layer that performs a dot product of the convolution kernel.

It's not all about coding!



PyTorch

What is PyTorch?

PyTorch documentation — PyTorch 1.13 documentation

PyTorch basics

- Deep learning framework
- It's Python-based
- GPU accelerated (fast and flexible)
- Tensors are the PyTorch hearth
- Beneficial as machine learning solutions

Data scientists

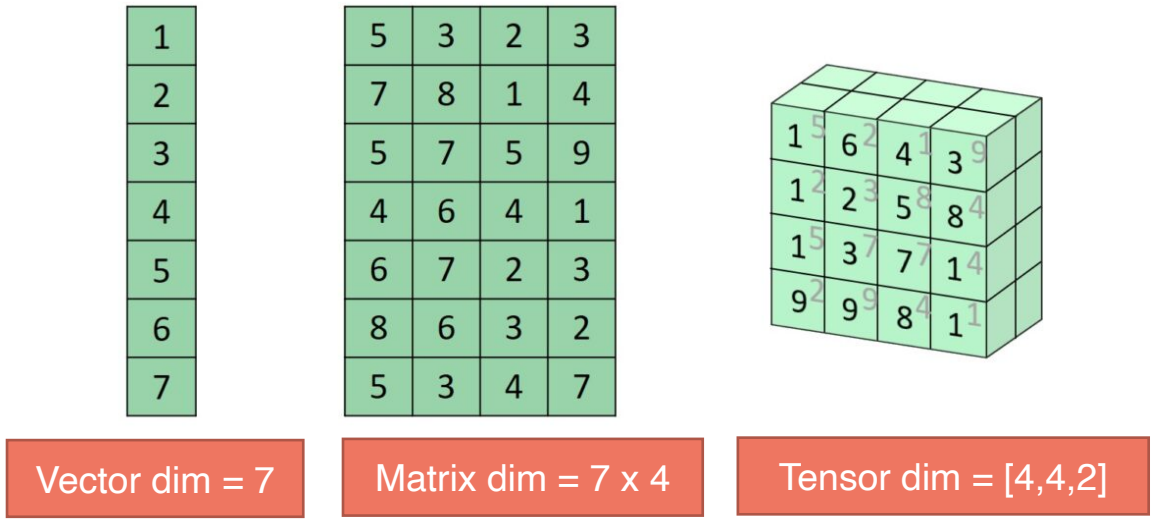
Researchers

Software engineers

Software developers

Tensors

- Tensors are data structure (similar to arrays and matrices, e.g. NumPy arrays).
- PyTorch uses tensors to encode the inputs and outputs of a model, as well as the model's parameters.
- Tensors can run on GPUs and are also optimized for automatic differentiation (torch.autograd)!



Datasets and DataLoaders

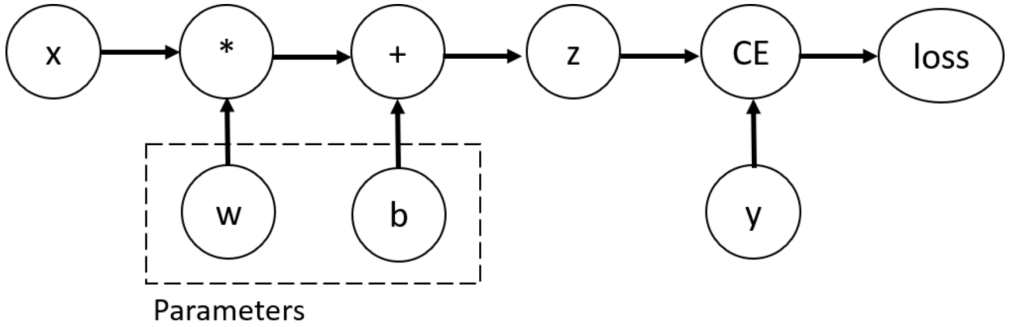
- PyTorch provides two data loaders:
- torch.utils.data.DataLoader and torch.utils.data.Dataset that allow you to use pre-loaded datasets as well as your own data. Dataset stores the samples and their corresponding labels
- DataLoader wraps an iterable around the Dataset to enable easy access to the samples. Main features of DataLoader are:
- The __init__ function is run once when instantiating the Dataset object.
- The __len__ function returns the number of samples in our dataset.
- The __getitem__ function loads and returns a sample from the dataset at the given index idx.

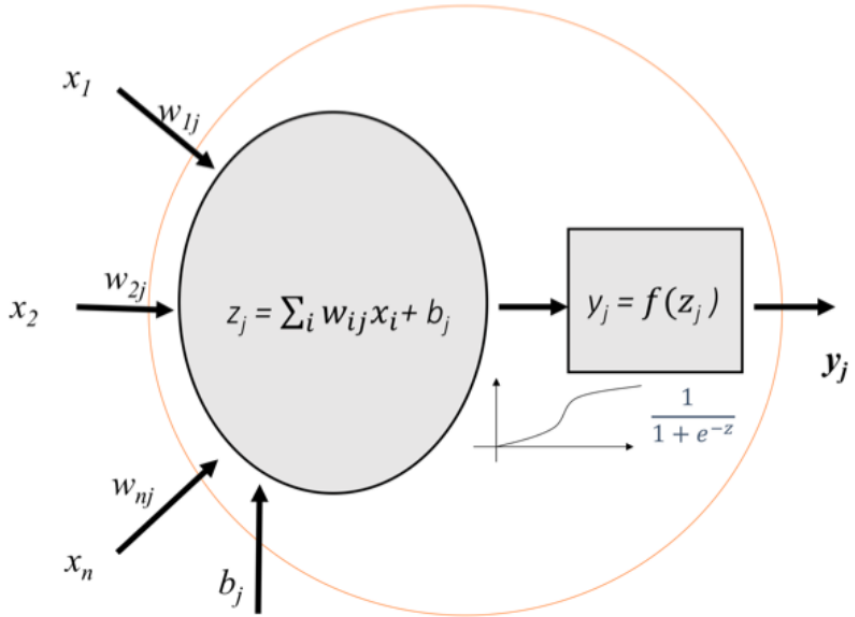
Build a model

- We define our neural network class with nn.Module, and initialize the neural network layers in __init__.
- Every nn.Module subclass implements the operations on input data in the forward method.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.lin1 = torch.nn.Linear(10,4)
        self.lin2 = torch.nn.Linear(4,1)
        self.relu = nn.ReLU()
        self.flatten = torch.nn.Flatten(0, 1)
        self.batchnorm1 = nn.BatchNorm1d(4)

    def forward(self, data_batch):

        x = data_batch
        x = self.lin1(x)
        x = self.relu(x)
        x = self.batchnorm1(x)
        x = self.lin2(x)
        x = self.flatten(x)
        pred = torch.sigmoid(x)
        return pred
```

Training





w and **b** are parameters, which we need to optimize and to do so we need to compute the derivatives of our loss function with respect to parameters. We stop tracking the computation activate with required_grad=True with torch.no_grad()

```
loss.backward()
```

Hyperparameters

- **Epochs** - the number times to iterate over the dataset
- **Batch Size** - data samples propagated through the network before the parameters are updated
- **Learning Rate** - how much to update model parameters at each batch/epoch.

Loss function and optimizer

- **Loss function** measures the error made by the predicted output and the target value, and it is the loss function that we want to minimize during training.

```
# Initialize the loss function
loss_fn = nn.CrossEntropyLoss()
```

- We adjust the model parameters by implementing **optimization algorithms** at each training round.

```
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

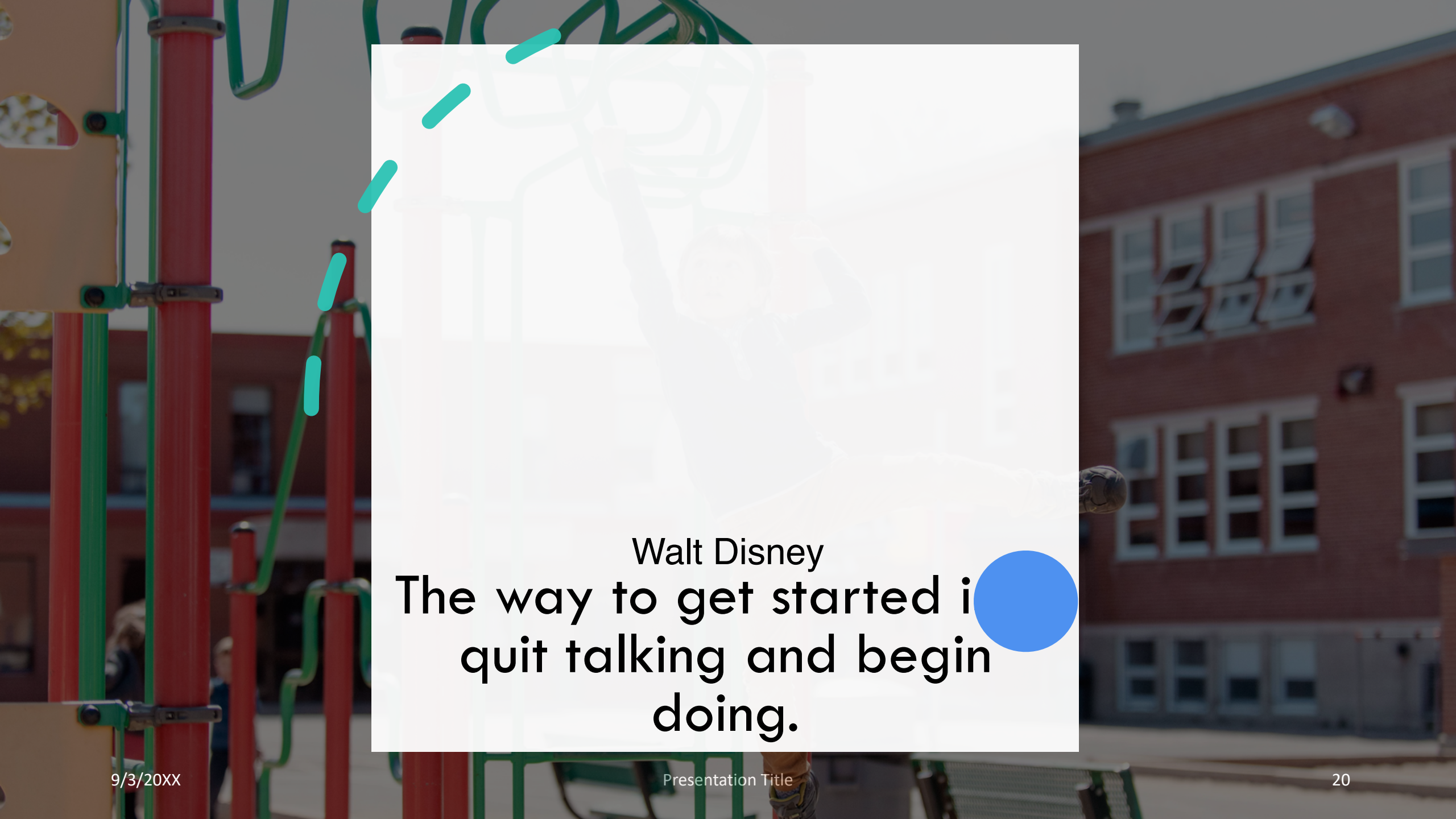
The full training with PyTorch

```python
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

Familiarize with PyTorch

Basic steps to start learning

- Learn how to load data
- Build a deep neural network
- Train and test your model

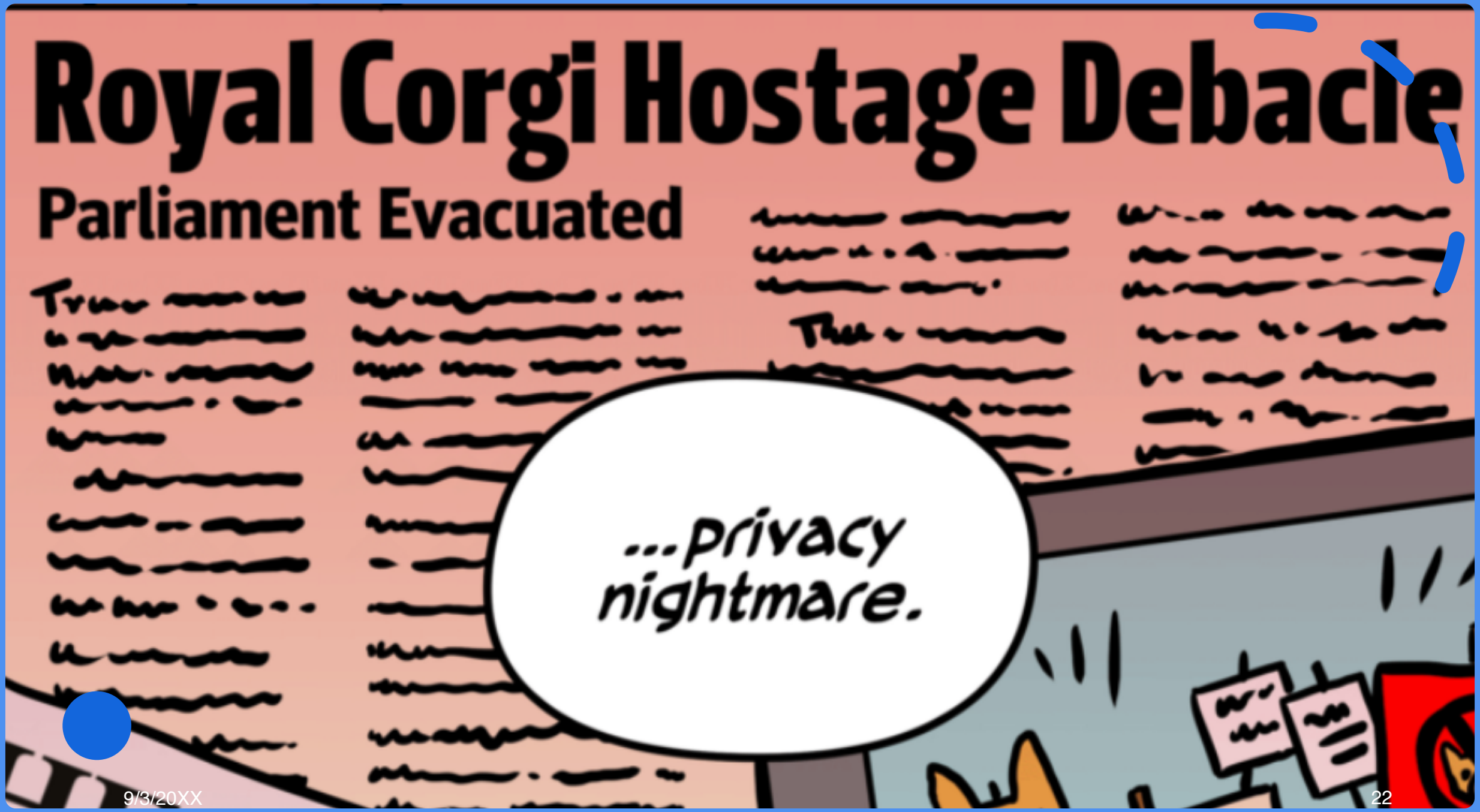


Walt Disney

The way to get started i
quit talking and begin
doing.

Timeline

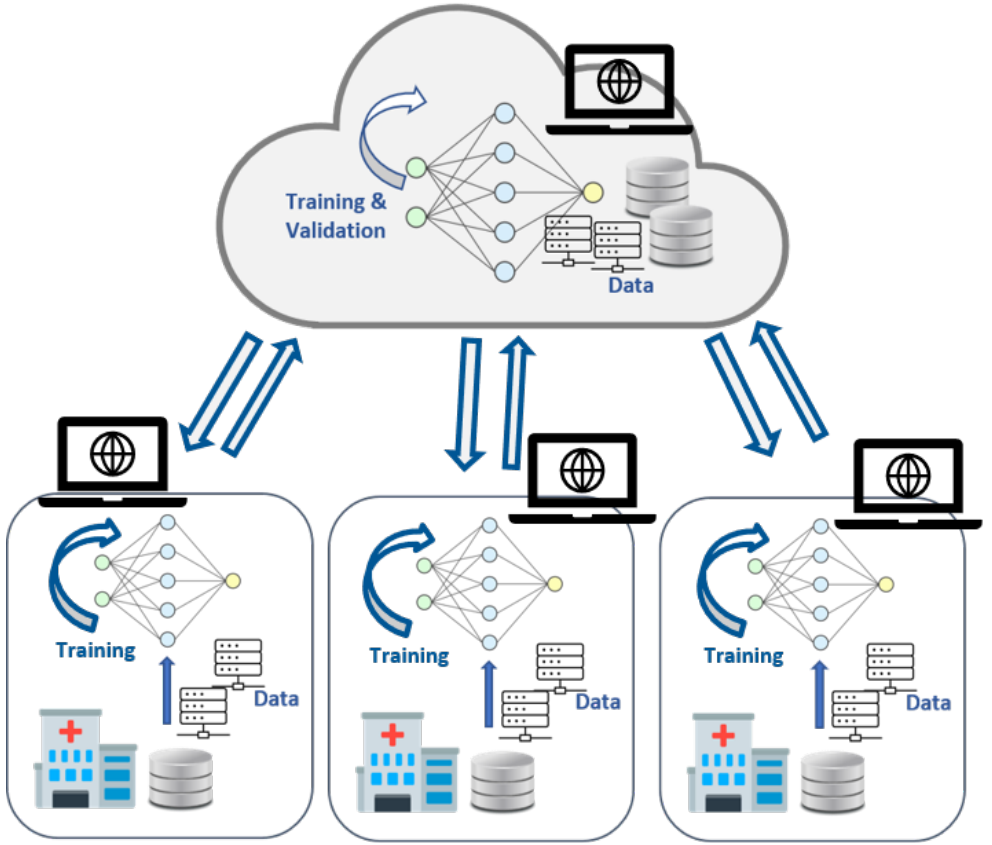
1. Start with downloading the data and get familiarity
2. Set up the proper preprocessing if needed
3. Designing your neural net
4. Training and testing
5. Federated learning: explaining a simple notebook to get an idea


Royal Corgi Hostage Debacle
Parliament Evacuated
...privacy nightmare.

An extended concept: Federated Learning

How to train your model by with other silos without sharing the data?

We can do it with PyTorch!



Thank you!

Francesca Manni

Francescang.manni@gmail.com