National Technical
University of Ukraine
"Igor Sikorsky
Kyiv Polytechnic Institute"

Institute of
Physics and
Technology

Intellectual Data Analysis

Practice 5: Fully Connected Neural Nets

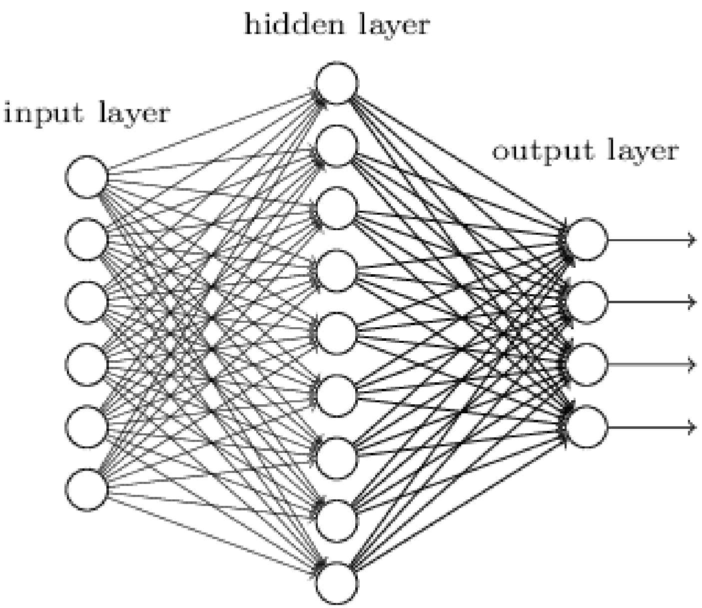
Dr. Nataliya K. Sakhnenko

Please review Lectures 6-7 before this practical

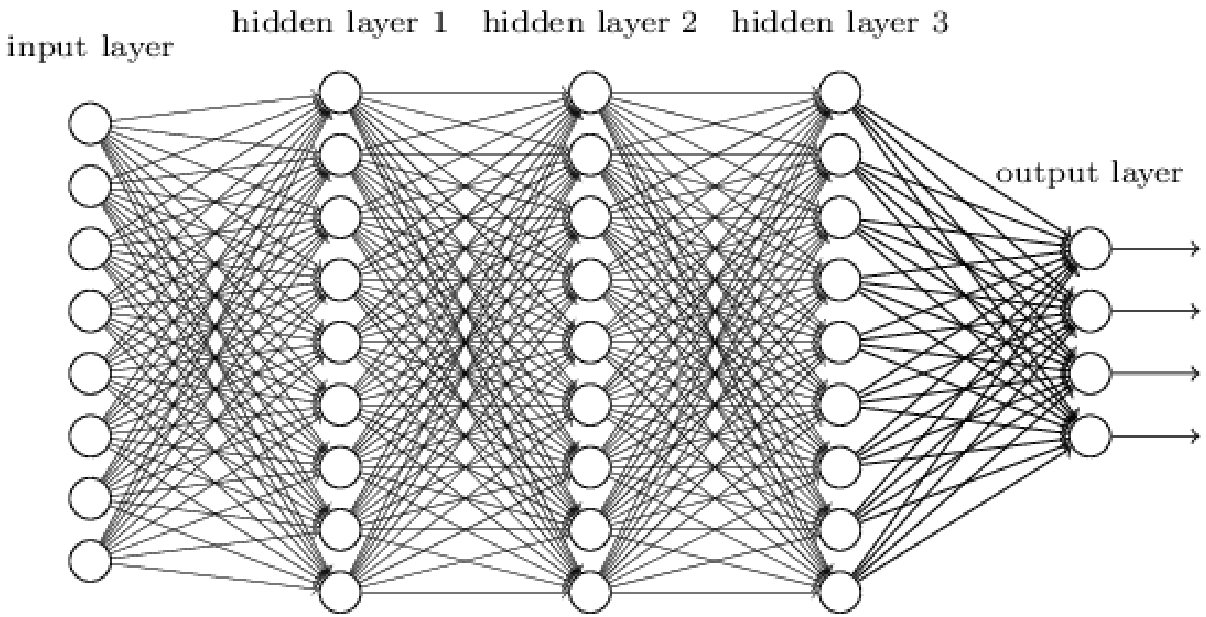
Multilayer Fully Connected (FC) ANNs

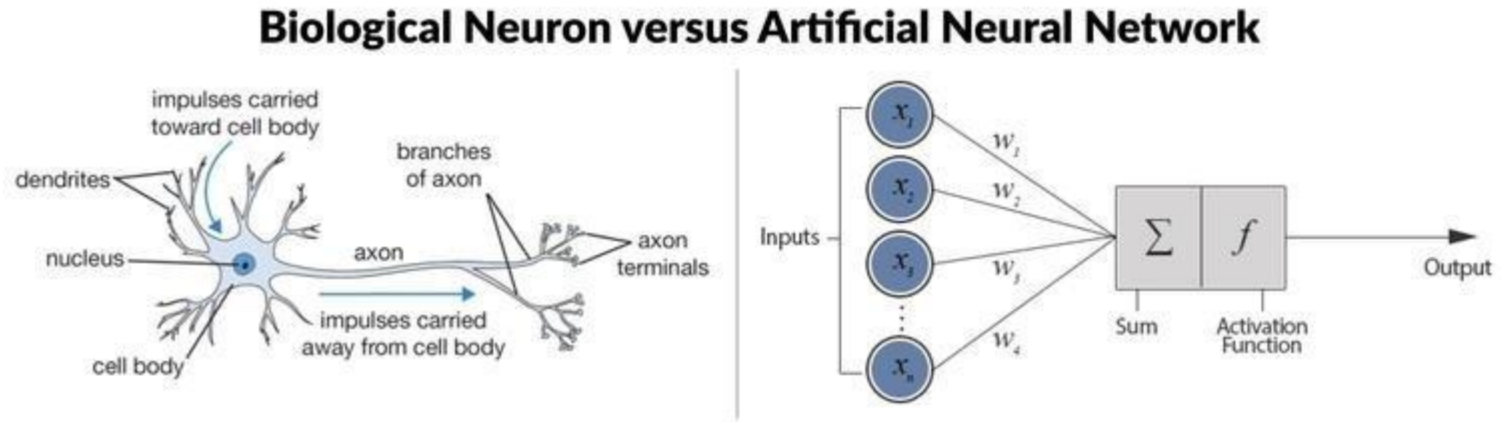
Multilayer Perceptron
(MLP)

"Non-deep" feedforward
neural network



Deep neural network





https://stats.stackexchange.com/questions/182734/what-is-the-difference-between-a-neural-network-and-a-deep-neural-network-and-w

Why PyTorch

- **Dynamic Computation Graph:** Unlike TensorFlow’s static graphs, PyTorch builds the computation graph on the fly, making it more flexible and easier to debug.
- **Pythonic:** PyTorch feels like native Python, making it more intuitive for Python developers.
- **Strong Community Support:** PyTorch has a growing and active community with extensive documentation and resources.
- **Seamless Integration:** It works well with popular Python libraries like NumPy and Pandas.

https://docs.pytorch.org/tutorials/beginner/basics/intro.html

1. Tensors
2. Datasets and DataLoaders
4. Build Model
5. Automatic Differentiation
6. Optimization Loop

Build Models with torch.nn

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        …..

    def forward(self, x):
         …
         return …
```

```
model =
NeuralNetwork().to(device)
```

✓ **Loss Function**

nn.MSELoss for regression tasks

nn.CrossEntropyLoss for classification

✓ **Optimizer**

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

Optimization

Inside the training loop, optimization happens in three steps:

•Call optimizer.zero_grad() to reset the gradients of model parameters. Gradients by default add up; to prevent double-counting, we explicitly zero them at each iteration.

•Backpropagate the prediction loss with a call to loss.backward().

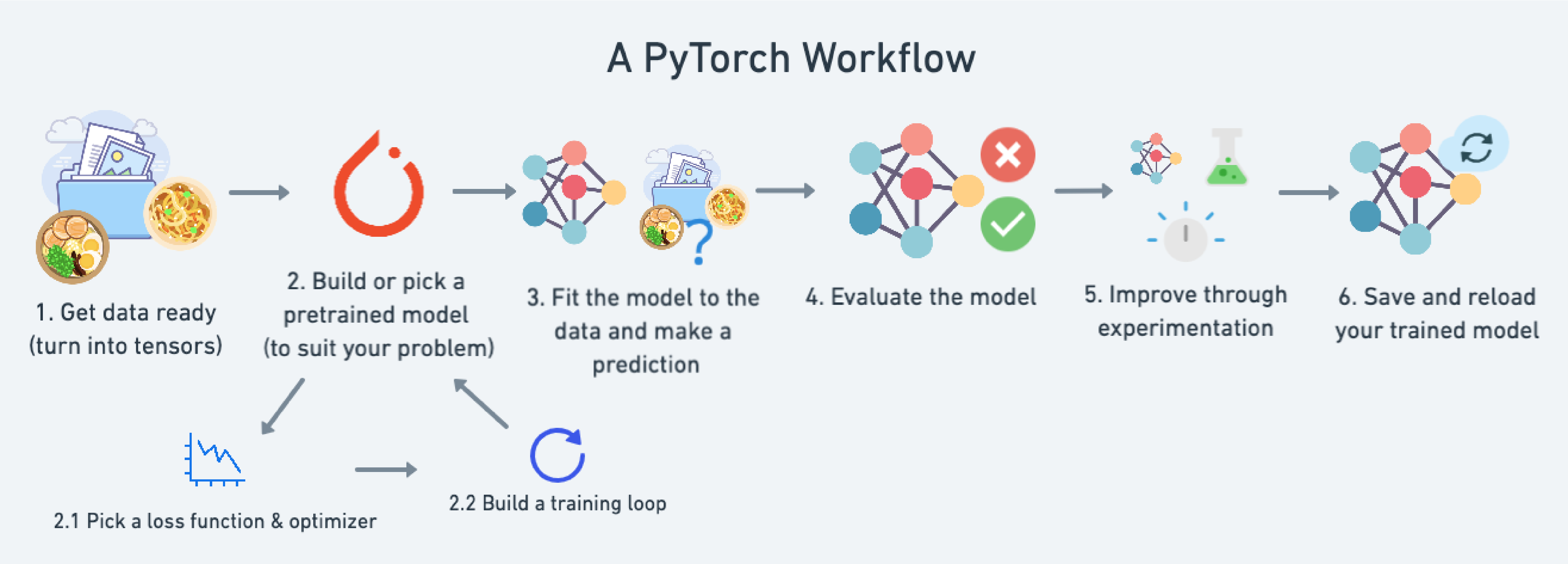
•Once we have our gradients, we call optimizer.step() to adjust the parameters by the gradients collected in the backward pass.

```
def train_loop(dataloader, model, loss_fn,
        optimizer):

    model.train()# added for best practices
    for batch, (X, y) in enumerate(dataloader):
        optimizer.zero_grad()
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)
        # Backpropagation
        loss.backward()
        optimizer.step()
```

```
def test_loop(dataloader, model, loss_fn):

    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y)
            correct += …
```


A PyTorch Workflow
1. Get data ready (turn into tensors)
2. Build or pick a pretrained model (to suit your problem)
3. Fit the model to the data and make a prediction
4. Evaluate the model
5. Improve through experimentation
6. Save and reload your trained model
2.1 Pick a loss function & optimizer
2.2 Build a training loop