

Deep learning engineering basics with PyTorch

BECM33MLE — Machine Learning Engineering

Dr. Tomáš Báča

Multi-Robot Systems group, Faculty of Electrical Engineering
Czech Technical University in Prague



FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE



MULTI-ROBOT
SYSTEMS
GROUP



DATAMOLE



DATAMOLE

- always recommend to work in a **Virtual Environment**
- it stores copies of the particular dependencies locally
- one solution to a **dependency hell**

Dependencies (requirements.txt)

```
1 numpy==2.3.3
2 matplotlib==3.10.6
3 scikit-learn==1.7.2
4 pandas==2.3.2
5 jupyter
6 jupyterlab-vim
7 wandb
8 transformers # huggingface
9 drawdata # drawing datasets
```

- the version is fixed for some of the packages to make the code within the examples work in the future

Setting up python environment (Ubuntu 24.04)

```
1 # install the python3's virtual environment
2 sudo apt get install python3-venv
3
4 # create the virtual environment
5 python3 -m venv python-env
6
7 # activate the environment
8 source ./python-env/bin/activate
9
10 # install dependencies manually
11 pip install numpy ...
```

or install dependencies in bulk

```
1 # install deps from requirements.txt
2 python3 -m pip install -r requirements.txt
```

Lecture 3: Deep learning engineering basics with PyTorch

Python Environment

Python Virtual Environment

- always recommend to work in a **Virtual Environment**
- it stores copies of the particular dependencies locally
- one solution to a **dependency hell**

Dependencies (requirements.txt)

```
numpy==2.3
matplotlib==3.10.6
scikit-learn==1.7.2
pandas==2.3.2
jupyter
jupyterlab-vim
wandb
transformers # huggingface
drawdata # drawing datasets
```

the version is fixed for some of the packages to make the code within the examples work in the future

Setting up python environment (Ubuntu 24.04)

```
# install the python3's virtual environment
sudo apt get install python3-venv

# create the virtual environment
python3 -m venv python-env

# activate the environment
source ./python-env/bin/activate

# install dependencies manually
pip install numpy ...
```

or install dependencies in bulk

```
python3 -m pip install -r requirements.txt
```

- there are other options to make sure you dependencies are met and are not colliding with other projects:
 - Conda package manager,
 - Docker,
 - Singularity / Apptainer container system,
 - Nix package manager.

Running Jupyter Notebook

```
1 # 1. activate your python env
2 source ./python-env/bin/activate
3
4 # 2. run
5 jupyter notebook
```

3. check your browser (localhost:8888)

Running Jupyter Lab

```
1 # 1. activate your python env
2 source ./python-env/bin/activate
3
4 # 2. run
5 jupyter-lab
```

3. check your browser (localhost:8888)

- Python Environment

- Jupyter Notebook & Jupyter Lab

- You can, of course, run your python scripts manually. However, this makes development and tweaking of the algorithms tedious.
- In practice, you can design the first part of the architecture in Jupyter and then move to more traditional program architecture.

Running Jupyter Notebook

```
1 # 1. activate your python env
2 source ~/.python-env/bin/activate
3
4 # 2. run
5 jupyter notebook
6
7 # check your browser (localhost:8888)
```

Running Jupyter Lab

```
1 # 1. activate your python env
2 source ./python-env/bin/activate
3
4 # 2. run
5 jupyter-lab
```

Jupyter Notebook & Jupyter Lab

Lecture 3:
Deep
learning
engineer-
ing basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

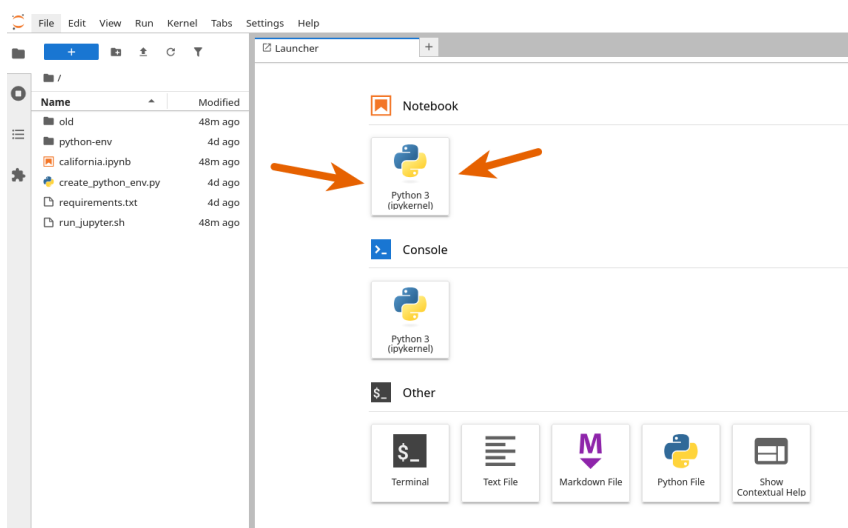
Simple ANN

Complex
XOR

Monitoring

Hyperopt

Hugging-
face



Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

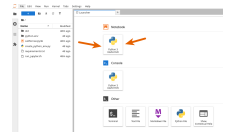
4 / 42

Lecture 3: Deep learning engineering basics with PyTorch

Python Environment

Jupyter Notebook & Jupyter Lab

Jupyter Notebook & Jupyter Lab



2025-10-11

Jupyter Notebook & Jupyter Lab

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

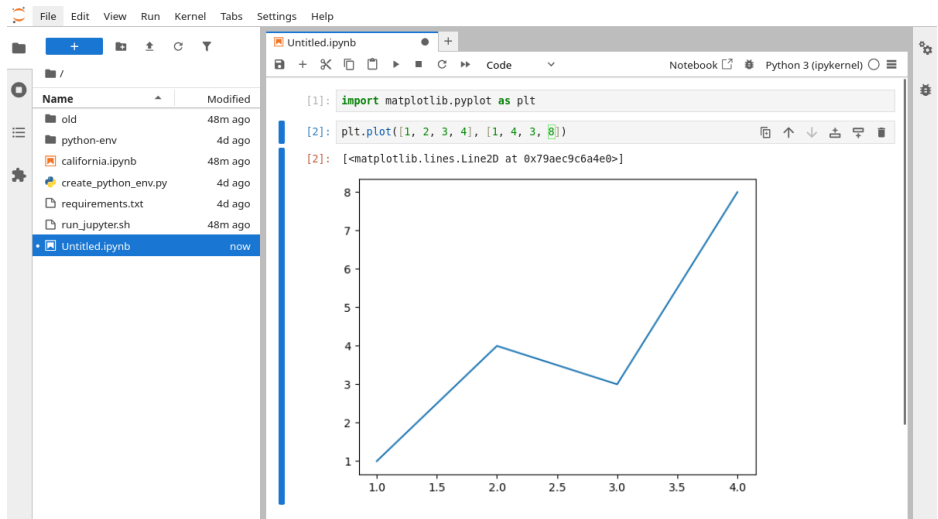
Complex

XOR

Monitoring

Hyperopt

Hugging-
face



Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

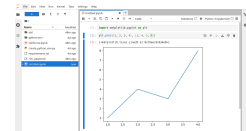
5 / 42

Lecture 3: Deep learning engineering basics with PyTorch

Python Environment

Jupyter Notebook & Jupyter Lab

Jupyter Notebook & Jupyter Lab



- use <tab> for code completion
- use shift+<tab> for explanation of objects and their properties
- use ctrl+enter to evaluate current cell

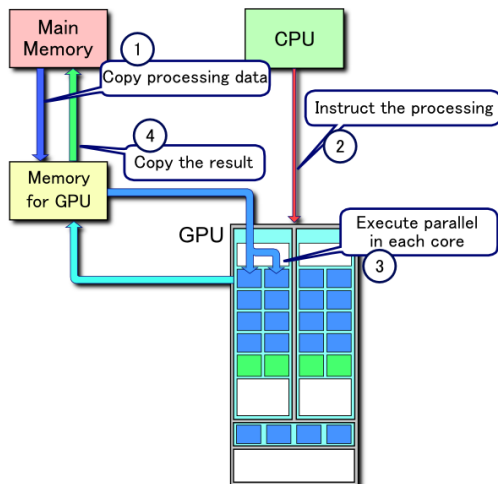
2025-10-11

CUDA

- Parallel computing in general
- vendor locked-in: Nvidia is dominating the market
- For ML: The more GPU memory you have, the better

Alternatives

- **OpenCL**
 - AMD, Intel and NVidia
 - lower performance
 - worse toolbox support
- **Vulcan**
 - mode focused on running than training
 - supports all HW platforms

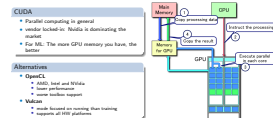


Lecture 3: Deep learning engineering basics with PyTorch

Python Environment

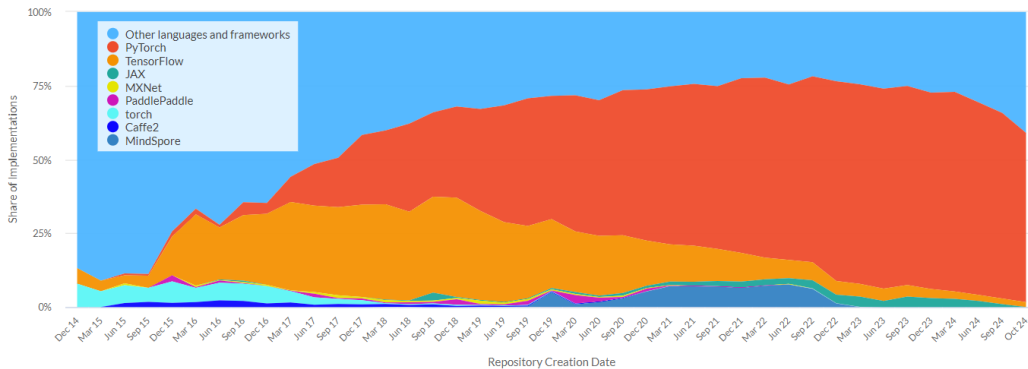
CUDA — Accelerating machine learning with GPUs

CUDA — Accelerating machine learning with GPUs



- CUDA is useful for parallel processing in general:
 - signal processing (Fourier transform)
 - image / video encoding
 - compression
 - cryptography
 - physics simulations (particle simulations)
- Installing CUDA is platform specific, always follow steps dedicated to your hardware and operating system

Share of deep learning libraries

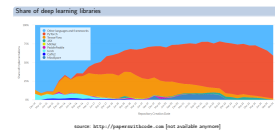


source: <http://paperswithcode.com> [not available anymore]

Lecture 3: Deep learning engineering basics with PyTorch

Deep learning libraries

Deep learning libraries



- In recent years: PyTorch is dominating the publicly released models and code.

PyTorch



- developed by Meta
- favored amongst researchers
- great for prototyping
- limited deployment capabilities
- Python-only API
- Dominance within “casuals”

Tensorflow



- developed by Google
- aimed at production
- models for end-user devices
- graph-based paradigm
- APIs:
 - Python, C++, JavaScript, Java (?)

JAX



- developed by Google
- rising star for research
- functional programming
- more *focused* on HPC
- Python-only API (Numpy-like)
- Steeper learning curve than PyTorch

Lecture 3: Deep learning engineering basics with PyTorch

Deep learning libraries

Deep learning libraries — comparison

PyTorch	Tensorflow	JAX
 • developed by Meta • favored amongst researchers • great for prototyping • limited deployment capabilities • Python-only API • Dominance within “casuals”	 • developed by Google • aimed at production • models for end-user devices • graph-based paradigm • APIs: • Python, C++, JavaScript, Java (?) • TensorFlow	 • developed by Google • rising star for research • functional programming • more focused on HPC • Python-only API (Numpy-like) • Steeper learning curve than PyTorch

Installing PyTorch

Lecture 3:
Deep
learning
engineer-
ing basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

Monitoring

Hyperopt

Hugging-
face

- generate installation command at <https://pytorch.org/get-started/> locally/

Install command generator for *my machine*

PyTorch Build	Stable (2.8.0)	Preview (Nightly)
Your OS	Linux	Mac
Package	Pip	LibTorch
Language	Python	C++ / Java
Compute Platform	CUDA 12.6	CUDA 12.8
		ROCm 7.0
		CPU
Run this Command:	pip3 install --pre torch torchvision --index-url https://download.pytorch.org/whl/nightly/cu130	

pip installation for CPU

```
1 source ./python-env/bin/activate
2
3 pip3 install --pre torch torchvision --
  index-url https://download.pytorch.org/
  whl/nightly/cu130
```

pip installation for CUDA 13.0

```
1 source ./python-env/bin/activate
2
3 pip3 install --pre torch torchvision --
  index-url https://download.pytorch.org/
  whl/nightly/cu130
```

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

9 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Installing PyTorch

Installing PyTorch

• generate installation command at <https://pytorch.org/get-started/> locally/

Install command generator for my machine

pip installation for CPU

```
source ./python-env/bin/activate

pip3 install --pre torch torchvision --
index-url https://download.pytorch.org/
whl/nightly/cu130
```

pip installation for CUDA 13.0

```
source ./python-env/bin/activate

pip3 install --pre torch torchvision --
index-url https://download.pytorch.org/
whl/nightly/cu130
```

- there are other options to make sure you dependencies are met and are not colliding with other projects:
 - Conda package manager,
 - Docker,
 - Singularity / Apptainer container system,
 - Nix package manager.

2025-10-11

Representing multi-dimensional data

- tensors are generalization of n-dimensional matrices (scalars, vectors, matrices, n-dimensional tensor, ...)
- propagation of information in deep learning (DL) linear layers is modeled by tensor operations:

$$\mathbf{y} = \mathbf{Ax} + \mathbf{b},$$

where $x \in \mathbb{R}^n$ is the layer's input tensor (vector),
 $y \in \mathbb{R}^m$ is the layer's output tensor (vector),
 $\mathbf{A} \in \mathbb{R}^{n \times m}$ is the tensor (matrix) of weights and
 $\mathbf{b} \in \mathbb{R}^m$ is the tensor (vector) of biases.

- hardware is capable of efficient evaluation of tensor operations
 - GPUs (CUDA)
 - TPUs (tensor processing units, ASICs)

Tensor dimensionality

- 0D tensor (scalar): $a = 3.14$
 - loss value, a single weight
- 1D tensor (vector): $\mathbf{a} = [3, 2, 1]^T$
 - feature vector, embedding vector
- 2D tensor (matrix): $\mathbf{A} = \begin{bmatrix} 2, 1, \\ 1, -3 \end{bmatrix}$
 - weight matrix, images (greyscale)
- 3D tensor: $\mathbb{A} = \left[\begin{bmatrix} 2, 1, \\ 1, -3 \end{bmatrix}, \begin{bmatrix} -1, 2, \\ 0, 1 \end{bmatrix} \right]$
 - batch of images (grayscale), color image, video (grayscale)
- 4D tensor:
 - video (colored), batch of color images

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Tensors

Representing multi-dimensional data

- tensors are generalization of n-dimensional matrices (scalars, vectors, matrices, n-dimensional tensor, ...)
- propagation of information in deep learning (DL) linear layers is modeled by tensor operations:

$$\mathbf{y} = \mathbf{Ax} + \mathbf{b},$$
 where $x \in \mathbb{R}^n$ is the layer's input tensor (vector),
 $y \in \mathbb{R}^m$ is the layer's output tensor (vector),
 $\mathbf{A} \in \mathbb{R}^{n \times m}$ is the tensor (matrix) of weights and
 $\mathbf{b} \in \mathbb{R}^m$ is the tensor (vector) of biases.
- hardware is capable of efficient evaluation of tensor operations
 - GPUs (CUDA)
 - TPUs (tensor processing units, ASICs)

Tensor dimensionality

- 0D tensor (scalar): $a = 3.14$
 - loss value, a single weight
- 1D tensor (vector): $\mathbf{a} = [3, 2, 1]^T$
 - feature vector, embedding vector
- 2D tensor (matrix): $\mathbf{A} = \begin{bmatrix} 2, 1, \\ 1, -3 \end{bmatrix}$
 - weight matrix, images (greyscale)
- 3D tensor: $\mathbb{A} = \left[\begin{bmatrix} 2, 1, \\ 1, -3 \end{bmatrix}, \begin{bmatrix} -1, 2, \\ 0, 1 \end{bmatrix} \right]$
 - batch of images (grayscale), color image, video (grayscale)
- 4D tensor:
 - video (colored), batch of color images

Tensors in PyTorch

Lecture 3:
Deep
learning
engineer-
ing basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex
XOR

Monitoring
Hyperopt

Hugging-
face

Example definition of tensors

```
1 import torch
2
3 # 1D tensor (row)
4 x = torch.tensor([1, 2])
5
6 # 1D tensor (column)
7 x = torch.tensor([1, 2]).view(-1, 1)
8
9 # 2D tensor (per row)
10 x = torch.tensor([[1, 2],
11                  [3, 4]])
```

Moving tensor to the GPU device

```
1 import torch
2
3 if torch.cuda.is_available():
4     device = torch.cuda.device(
5         current_device().type
6     )
7 else:
8     device = "cpu"
9
10 # 2D tensor (per row)
11 x = torch.tensor([[1, 2],
12                  [3, 4]]).to(device)
```

Operations with tensors

- reshaping: view, reshape
- math: element-wise operations, tensor-wise operations (broadcasting)
- automatic tracking of gradients

Moving tensor back to CPU

```
1 import torch
2
3 x_cpu = x.cpu()
```

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

11 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Tensors in PyTorch

2025-10-11

Tensors in PyTorch

Example definition of tensors

```
1 import torch
2
3 # 1D tensor (row)
4 x = torch.tensor([1, 2])
5
6 # 1D tensor (column)
7 x = torch.tensor([1, 2]).view(-1, 1)
8
9 # 2D tensor (per row)
10 x = torch.tensor([[1, 2],
11                  [3, 4]])
```

Operations with tensors

- reshaping: view, reshape
- math: element-wise operations, tensor-wise operations (broadcasting)
- automatic tracking of gradients

Moving tensor to the GPU device

```
1 import torch
2
3 if torch.cuda.is_available():
4     device = torch.cuda.device(
5         current_device().type
6     )
7 else:
8     device = "cpu"
9
10 # 2D tensor (per row)
11 x = torch.tensor([[1, 2],
12                  [3, 4]]).to(device)
```

Moving tensor back to CPU

```
1 import torch
2
3 x_cpu = x.cpu()
```

Simple MLP for simple XOR

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex
XOR

Monitoring
Hyperopt

Hugging-
face

XOR classification task

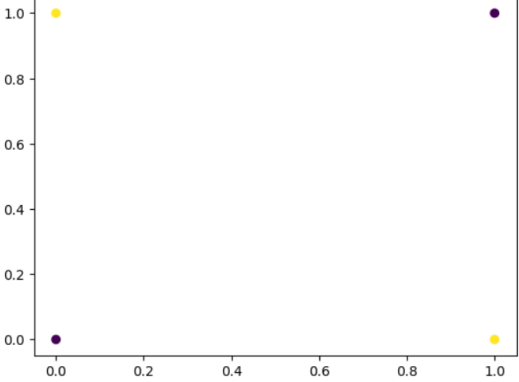
- linearly non-separable classes
- can not be solve by purely linear classification (SVM, perceptron)
- solutions within classical ML:
 - lifting to higher dimensions (+ x_1x_2)
 - decision tree
 - kernel SVM

Jupyter Notebook

The following part of the lecture is accompanied by

`01_pytorch_xor_simple.ipynb`

XOR classification



Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Simple ANN

Simple MLP for simple XOR

2025-10-11

Simple MLP for simple XOR

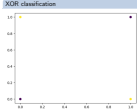
XOR classification task

- linearly non-separable classes
- can not be solve by purely linear classification (SVM, perceptron)
- solutions within classical ML:
 - lifting to higher dimensions (+ x_1x_2)
 - decision tree
 - kernel SVM

Jupyter Notebook

The following part of the lecture is accompanied by

`01_pytorch_xor_simple.ipynb`

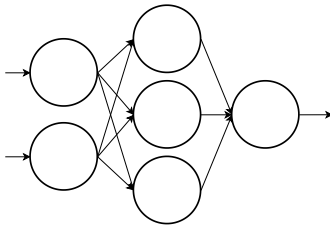


Solving XOR using PyTorch and MultiLayer Perceptron (MLP)

1. Imports

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4
5 import torch
6 from torch import nn
7 from torch import optim
8 from torch.autograd import Variable
```

- will be the same in all following examples



2. NN structure

```
1 class NeuralNetwork(nn.Module):
2     def __init__(self):
3
4         super().__init__()
5
6         self.linear1 = nn.Linear(2, 3)
7         self.sig1 = nn.Sigmoid()
8         self.linear2 = nn.Linear(3, 1)
9
10    def forward(self, x):
11
12        x = self.linear1(x)
13        x = self.sig1(x)
14        x = self.linear2(x)
15
16    return x
```

- 1 input layer, hidden layer, 1 output layer
- sigmoid activation before the hidden layer

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Simple ANN

Solving XOR using PyTorch and MultiLayer Perceptron (MLP)

1. Imports

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import torch
from torch import nn
from torch import optim
from torch.autograd import Variable
```

* will be the same in all following examples

2. NN structure

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = nn.Linear(2, 3)
        self.sig1 = nn.Sigmoid()
        self.linear2 = nn.Linear(3, 1)

    def forward(self, x):
        x = self.linear1(x)
        x = self.sig1(x)
        x = self.linear2(x)
        return x
```

- 1 input layer, hidden layer, 1 output layer
- sigmoid activation before the hidden layer

Solving XOR using PyTorch

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

Monitoring

Hyperopt

Hugging-
face

3. prepare the model

```
1 # instantiate the model
2 model = NeuralNetwork()
3
4 # randomize weights before learning
5 for m in model.modules():
6     if isinstance(m, nn.Linear):
7         m.weight.data.normal_(0, 1)
```

4. prepare optimization task

```
1 # prepare loss function
2 loss_func = nn.MSELoss()
3
4 # prepare optimizer
5 optimizer = optim.SGD(
6     model.parameters(),
7     lr=0.02
8 )
```

5. create a XOR training set

```
1 X = torch.Tensor(
2     [[0, 0],
3      [0, 1],
4      [1, 0],
5      [1, 1]])
6
7 # two classes
8 y = torch.Tensor(
9     [0,
10     1,
11     1,
12     0]
13 ).view(-1, 1)
```

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

14 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Simple ANN

Solving XOR using PyTorch

2025-10-11

Solving XOR using PyTorch

3. prepare the model

```
1 # instantiate the model
2 model = NeuralNetwork()
3
4 # randomize weights before learning
5 for m in model.modules():
6     if isinstance(m, nn.Linear):
7         m.weight.data.normal_(0, 1)
```

4. prepare optimization task

```
1 # prepare loss function
2 loss_func = nn.MSELoss()
3
4 # prepare optimizer
5 optimizer = optim.SGD(
6     model.parameters(),
7     lr=0.02
```

5. create a XOR training set

```
1 X = torch.Tensor(
2     [[0, 0],
3      [0, 1],
4      [1, 0],
5      [1, 1]])
6
7 # two classes
8 y = torch.Tensor(
9     [0,
10     1,
11     1,
12     0]
13 ).view(-1, 1)
```

- The **loss function** defines how the "output error" of the neural net is computed.
 - different loss functions are suitable for different tasks
 - **MSELoss** stands for *Mean Square Error Loss*, is suitable for regression (and simple classification)
- the SGD optimizer performs *Stochastic Gradient Descent* to find a local optimum of a function
- the `lr` argument defines the step size of the gradient descent-based optimization methods

6. learning loop

```
1 for i in range(1000):
2
3     # forward pass
4     pred = model(X)
5
6     # reset gradients
7     optimizer.zero_grad()
8
9     # calculate the loss
10    loss = loss_func.forward(pred, y)
11
12    # calculate gradients
13    loss.backward()
14
15    # update the weights
16    optimizer.step()
17
18    if i % 100 == 0:
19        print("epoch {} loss {}".format(i, loss))
```

7. learning

```
1 epoch 0 loss 0.477
2 epoch 100 loss 0.228
3 epoch 200 loss 0.202
4 epoch 300 loss 0.172
5 epoch 400 loss 0.135
6 epoch 500 loss 0.084
7 epoch 600 loss 0.033
8 epoch 700 loss 0.007
9 epoch 800 loss 0.001
10 epoch 900 loss 0.000
```

Lecture 3: Deep learning engineering basics with PyTorch

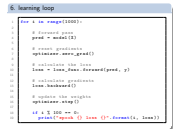
PyTorch

Simple ANN

Solving XOR using PyTorch

2025-10-11

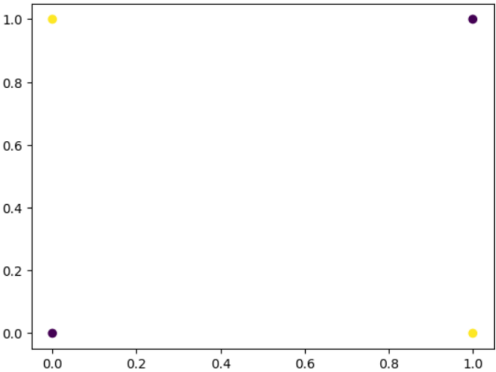
Solving XOR using PyTorch



8. testing

```
1 X = torch.Tensor(  
2     [[0, 0],  
3      [0, 1],  
4      [1, 0],  
5      [1, 1]])  
6  
7 output = model(X)
```

XOR classification output — it works!

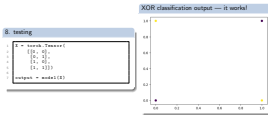


Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Simple ANN

Solving XOR using PyTorch



Solving XOR using PyTorch

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

Monitoring

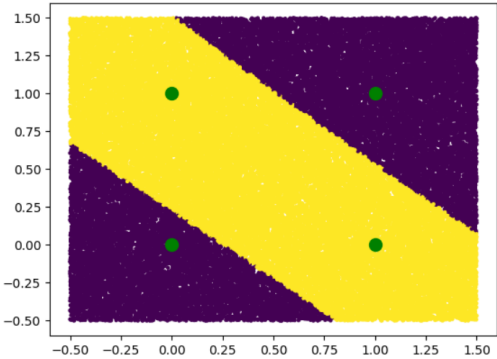
Hyperopt

Hugging-
face

Decision boundary?

```
1 # generate new testing data
2 X_test = torch.rand(50000, 2)*2.0 -
   0.5
3
4 # run our data through the model
5 output = model(X_test)
6
7 # plot the testing data just for
   comparison
8 plt.scatter(X_test[:, 0], X_test[:,
   1], c=output >= 0.5, s=5)
9 plt.scatter([0, 1, 1, 0], [0, 0, 1,
   1], c="green", s=100)
```

Decision boundaries between the two classes



Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

17 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Simple ANN

Solving XOR using PyTorch

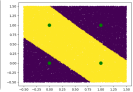
2025-10-11

Solving XOR using PyTorch

Decision boundary?

```
1 # generate new testing data
2 X_test = torch.rand(50000, 2)*2.0 -
   0.5
3
4 # run our data through the model
   output = model(X_test)
5
6 # plot the testing data just for
   comparison
7 plt.scatter(X_test[:, 0], X_test[:,
   1], c=output >= 0.5, s=5)
8 plt.scatter([0, 1, 1, 0], [0, 0, 1,
   1], c="green", s=100)
```

Decision boundaries between the two classes



What did we do?

Lecture 3:
Deep
learning
engineer-
ing basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

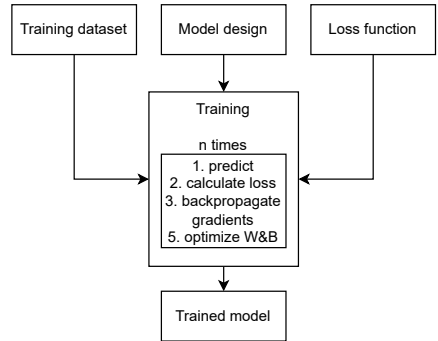
Simple ANN

Complex
XOR

Monitoring
Hyperopt

Hugging-
face

- We created a training dataset including labels.
- We Designed out *deep learning model* (quite shallow now).
- We set up the loss function and an optimization method.
- We executed the learning step 1000x times.
- We used the model to predict a class for new data.



Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

18 / 42

Lecture 3: Deep learning engineering basics with PyTorch

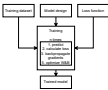
PyTorch

Simple ANN

What did we do?

What did we do?

- We created a training dataset including labels.
- We Designed out *deep learning model* (quite shallow now).
- We set up the loss function and an optimization method.
- We executed the learning step 1000x times.
- We used the model to predict a class for new data.



2025-10-11

More complex XOR task

Lecture 3:
Deep
learning
engineer-
ing basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex
XOR

Monitoring

Hyperopt

Hugging-
face

deeper XOR classification task

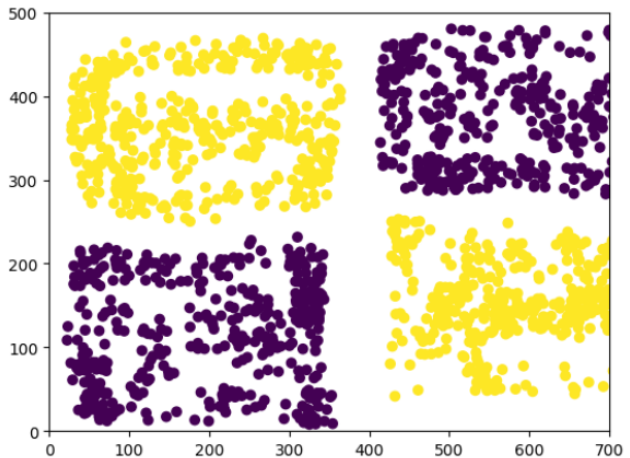
- non-singular classes
- dataset needs scaling
- decision boundaries not linear
- more training data
 - might need splitting to batches
- might benefit from running using CUDA

Jupyter Notebook

The following part of the lecture is accompanied by

`02_pytorch_xor_medium.ipynb`

XOR classification



Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

19 / 42

Lecture 3: Deep learning engineering basics with PyTorch

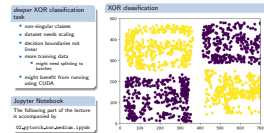
PyTorch

Complex XOR

More complex XOR task

2025-10-11

More complex XOR task



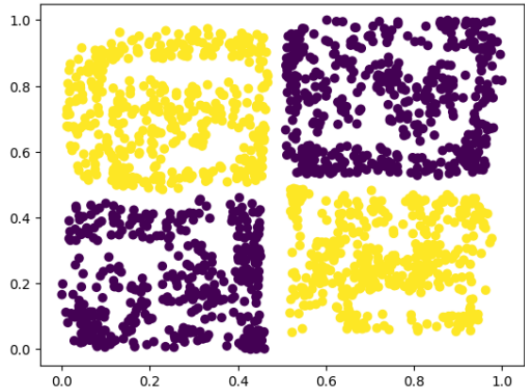
Dataset preprocessing

- PyTorch does not contain complex Pipeline with preprocessing like Sklearn
- We are supposed to get the preprocessing done ourselves
- Let's use Sklearn again
 - `QuantileTransformer()`

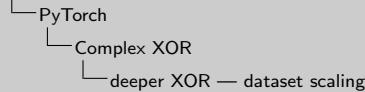
Code

```
1 scaler =
2   QuantileTransformer(
3       n_quantiles=10)
4 scaler.fit(X)
5 X_transformed =
6   scaler.transform(X)
```

XOR classification



Lecture 3: Deep learning engineering basics with PyTorch



Dataset preprocessing

- PyTorch does not contain complex Pipeline with preprocessing like Sklearn
- We are supposed to get the preprocessing done ourselves
- Let's use Sklearn again
 - `QuantileTransformer()`

Code

```
scaler =
  QuantileTransformer(
      n_quantiles=10)
scaler.fit(X)
X_transformed =
  scaler.transform(X)
```

XOR classification

Batches

- proper batch size is important:
- too large:
 - data might not fit into memory
 - the model might generalize poorly
- too small:
 - many learning steps with an epoch

Code

```
1 # convert the transformed dataset to tensors
2 X_tens = torch.tensor(X_transformed).to(torch.float32)
3 y_tens = torch.tensor(y).to(torch.float32).view(-1, 1)
4
5 # split dataset into batches
6 from torch.utils.data import DataLoader, TensorDataset
7
8 dataset = TensorDataset(X_tens, y_tens)
9 dataloader = DataLoader(
10     dataset,
11     batch_size=32,
12     shuffle=True
13 )
```

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Complex XOR

deeper XOR — learning batches

2025-10-11

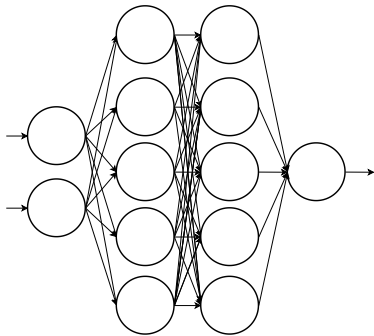
deeper XOR — learning batches

Batches

- proper batch size is important
- too large
 - data might not fit into memory
 - the model might generalize poorly
- too small
 - many learning steps with an epoch

Code

```
1 # convert the transformed dataset to tensors
2 X_tens = torch.tensor(X_transformed).to(torch.float32)
3 y_tens = torch.tensor(y).to(torch.float32).view(-1, 1)
4
5 # split dataset into batches
6 from torch.utils.data import DataLoader, TensorDataset
7
8 dataset = TensorDataset(X_tens, y_tens)
9 dataloader = DataLoader(
10     dataset,
11     batch_size=32,
12     shuffle=True
13 )
```



2. NN structure

```
1 class NeuralNetwork(nn.Module):
2
3     def __init__(self):
4         super().__init__()
5
6         self.linear1 = nn.Linear(2, 5)
7         self.sigml = nn.Sigmoid()
8         self.linear2 = nn.Linear(5, 5)
9         self.sig2 = nn.Sigmoid()
10        self.linear3 = nn.Linear(5, 1)
11
12    def forward(self, x):
13        x = self.linear1(x)
14        x = self.sigml(x)
15        x = self.linear2(x)
16        x = self.sig2(x)
17        x = self.linear3(x)
18        return x
```

- 1 input layer, 2 hidden layers, 1 output layer
- sigmoid activation before the hidden layers

Lecture 3: Deep learning engineering basics with PyTorch

- PyTorch
 - Complex XOR
 - deeper XOR — deeper model

2025-10-11

deeper XOR — deeper model



2. NN structure

```
1 class NeuralNetwork(nn.Module):
2
3     def __init__(self):
4         super().__init__()
5
6         self.linear1 = nn.Linear(2, 5)
7         self.sigml = nn.Sigmoid()
8         self.linear2 = nn.Linear(5, 5)
9         self.sig2 = nn.Sigmoid()
10        self.linear3 = nn.Linear(5, 1)
11
12    def forward(self, x):
13        x = self.linear1(x)
14        x = self.sigml(x)
15        x = self.linear2(x)
16        x = self.sig2(x)
17        x = self.linear3(x)
18        return x
```

• 1 input layer, 2 hidden layers, 1 output layer
• sigmoid activation before the hidden layers

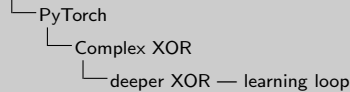
Learning loop

```
1 for i in range(1000):
2
3     for batch_X, batch_y in dataloader:
4
5         X_var = Variable(batch_X, requires_grad=
6             False).to(device)
7         y_var = Variable(batch_y, requires_grad=
8             False).to(device)
9
10        output = model(X_var)
11
12        optimizer.zero_grad()
13
14        loss = loss_func.forward(output, y_var)
15
16        loss.backward()
17
18        optimizer.step()
19
20    if i % 100 == 0:
21        print("epoch {} loss {}".format(i, loss))
```

Learning

1	epoch 0	loss	0.304
2	epoch 100	loss	0.060
3	epoch 200	loss	0.091
4	epoch 300	loss	0.005
5	epoch 400	loss	0.040
6	epoch 500	loss	0.022
7	epoch 600	loss	0.003
8	epoch 700	loss	0.028
9	epoch 800	loss	0.006
10	epoch 900	loss	0.013

Lecture 3: Deep learning engineering basics with PyTorch



deeper XOR — learning loop

```
Learning loop
1 for i in range(1000):
2     for batch_X, batch_y in dataloader:
3         X_var = Variable(batch_X, requires_grad=
4             False).to(device)
5         y_var = Variable(batch_y, requires_grad=
6             False).to(device)
7         output = model(X_var)
8         optimizer.zero_grad()
9         loss = loss_func.forward(output, y_var)
10        loss.backward()
11
12        optimizer.step()
13
14    if i % 100 == 0:
15        print("epoch {} loss {}".format(i, loss))
```

Learning
epoch 0 loss 0.304
epoch 100 loss 0.060
epoch 200 loss 0.091
epoch 300 loss 0.005
epoch 400 loss 0.040
epoch 500 loss 0.022
epoch 600 loss 0.003
epoch 700 loss 0.028
epoch 800 loss 0.006
epoch 900 loss 0.013

More complex XOR task

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex
XOR

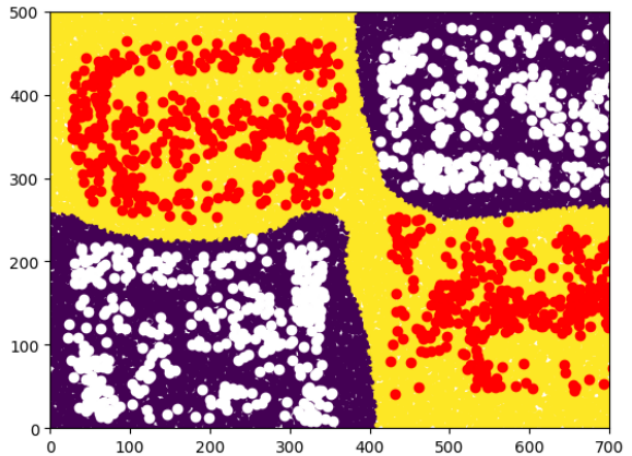
Monitoring
Hyperopt

Hugging-
face

Observations

- the original training data (red and white) almost correctly classified
- the decision boundary (between yellow and blue) generalizes quite well, but
- the decision boundaries are not robust (but we implemented nothing in to process to make it robust)

Decision boundary



Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

24 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

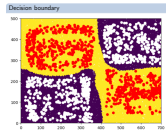
Complex XOR

More complex XOR task

2025-10-11

More complex XOR task

- Observations
- the original training data (red and white) almost correctly classified
 - the decision boundary (between yellow and blue) generalizes quite well, but
 - the decision boundaries are not robust (but we implemented nothing in to process to make it robust)



Monitoring of the learning process

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex
XOR

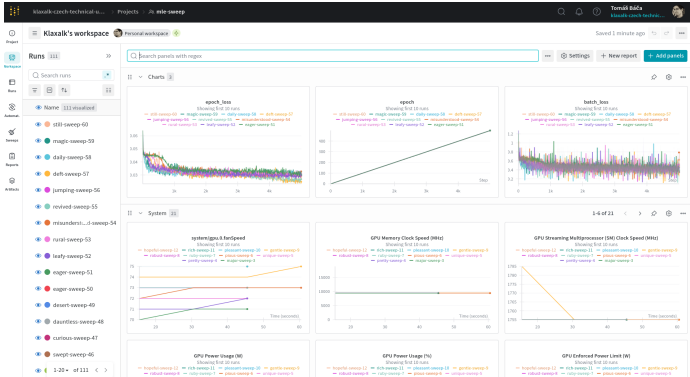
Monitoring

Hyperopt

Hugging-
face

Weights & Biases

- <http://wandb.ai>
- Online system for Monitoring
- Python API that integrates with your learning process
- Can gather arbitrary learning data, metrics
- Can gather resulting models and their weights (and biases)
- Facilitates parameter sweeps



Jupyter Notebook

The following part of the lecture is accompanied by `03_pytorch_separability_wandb.ipynb`

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

25 / 42

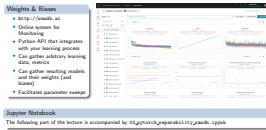
Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Monitoring

Monitoring of the learning process

Monitoring of the learning process



2025-10-11

Installation

```
1 pip3 install wandb
```

First steps

- 1 register
- 2 get access token
- 3 login:

```
1 wandb login <your access token>
```

Preparation

```
1 import wandb
2
3 # define reusable params:
4 epochs=1000
5 momentum=0.9
6 learning_rate=0.2
7
8 run = wandb.init(
9     entity="<your account>",
10    project="mle",
11    # tracked values
12    config={
13        "learning_rate": learning_rate,
14        "momentum": momentum,
15        "architecture": "MLP",
16        "epochs": epochs,
17    },
18 )
```

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Monitoring

How to start with WandB

2025-10-11

How to start with WandB

Installation

```
pip3 install wandb
```

First steps

- 1 register
- 2 get access token
- 3 login

```
wandb login <your access token>
```

Preparation

```
import wandb

# define reusable params:
epochs=1000
momentum=0.9
learning_rate=0.2

run = wandb.init(
    entity="<your account>",
    project="mle",
    # tracked values
    config={
        "learning_rate": learning_rate,
        "momentum": momentum,
        "architecture": "MLP",
        "epochs": epochs,
    },
)
```

Tracking variables during the learning process

Learning loop

```
1 for i in range(epochs):
2
3     batch_n = 0
4     avg_epoch_loss=0
5
6     for batch_X, batch_y in dataloader:
7
8         <the learning 'core' from the previous case>
9
10        wandb.log({"batch_loss": loss.mean(), "epoch": i})
11
12        avg_epoch_loss = avg_epoch_loss + loss.mean()
13
14        batch_n = batch_n + 1
15
16    avg_epoch_loss = avg_epoch_loss / batch_size
17
18    wandb.log({"epoch_loss": avg_epoch_loss, "epoch": i})
19
20    if i % 100 == 0:
21        print("epoch {} loss {}".format(i, loss.cpu().data.
            numpy()))
```

Learning

```
1 epoch 0 loss 0.63
2 epoch 100 loss 0.52
3 epoch 200 loss 0.39
4 epoch 300 loss 0.54
5 epoch 400 loss 0.39
6 epoch 500 loss 0.45
7 epoch 600 loss 0.33
8 epoch 700 loss 0.36
9 epoch 800 loss 0.26
10 epoch 900 loss 0.47
```

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

27 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Monitoring

Tracking variables during the learning process

Tracking variables during the learning process

```
Learning loop
1 for i in range(epochs):
2     batch_n = 0
3     avg_epoch_loss=0
4
5     for batch_X, batch_y in dataloader:
6
7         <the learning 'core' from the previous case>
8
9         wandb.log({"batch_loss": loss.mean(), "epoch": i})
10
11         avg_epoch_loss = avg_epoch_loss + loss.mean()
12
13         batch_n = batch_n + 1
14
15    avg_epoch_loss = avg_epoch_loss / batch_size
16
17    wandb.log({"epoch_loss": avg_epoch_loss, "epoch": i})
18
19    if i % 100 == 0:
20        print("epoch {} loss {}".format(i, loss.cpu().data.
            numpy()))
```

```
Learning
1 epoch 0 loss 0.63
2 epoch 100 loss 0.52
3 epoch 200 loss 0.39
4 epoch 300 loss 0.54
5 epoch 400 loss 0.39
6 epoch 500 loss 0.45
7 epoch 600 loss 0.33
8 epoch 700 loss 0.36
9 epoch 800 loss 0.26
10 epoch 900 loss 0.47
```

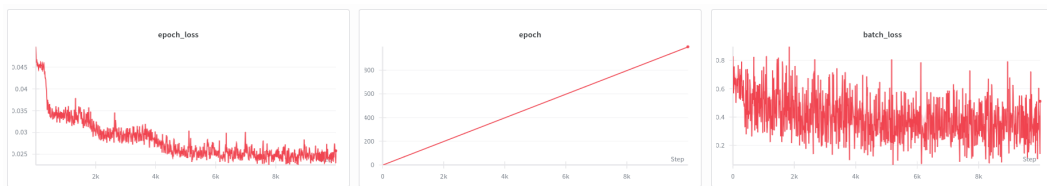
2025-10-11

Visualizing the variables in WandB

What do you get?

- history of your learning runs
- history of your parameters
- possibly history of your models

Graphs of your variables



Tomáš Bába (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

28 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Monitoring

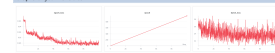
Visualizing the variables in WandB

Visualizing the variables in WandB

What do you get?

- history of your learning runs
- history of your parameters
- possibly history of your models

Graphs of your variables



2025-10-11

What problem did we solve this time?

Lecture 3:
Deep
learning
engineering basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex
XOR

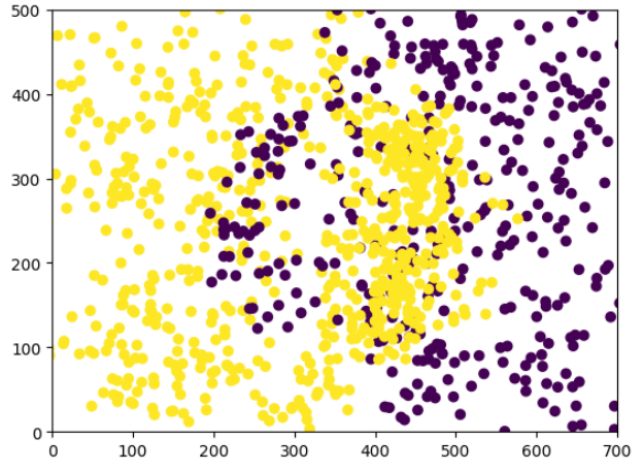
Monitoring
Hyperopt

Hugging-
face

Initial handcrafted approach

- MinMaxScaler
- hidden layers with 5 and 3 neurons
- learning rate = 0.2
- momentum = 0.9
- epochs = 1000
- batch size = 128

Classification of non-separable classes from Lecture 02



Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

29 / 42

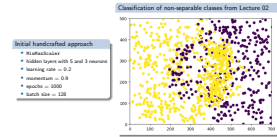
Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Monitoring

What problem did we solve this time?

What problem did we solve this time?

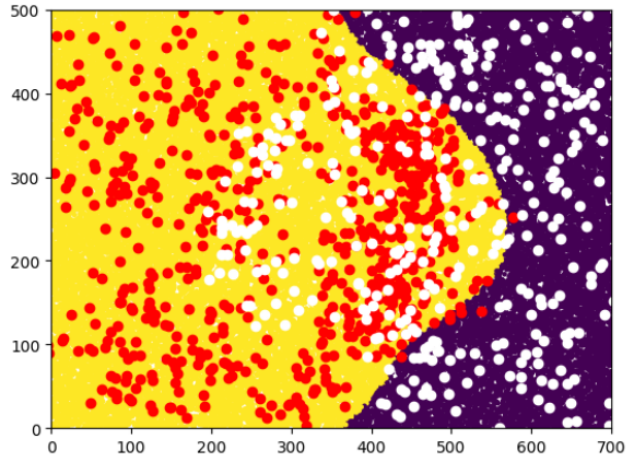


2025-10-11

Initial handcrafted approach

- loss oscillated around 0.3
- looks to be biased more towards yellow class
- **were my parameters well-chosen?**

Results



Lecture 3: Deep learning engineering basics with PyTorch

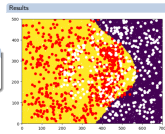
PyTorch

Monitoring

Results

2025-10-11

Results



- The parameters were chosen arbitrarily with more-or-less (less) decent expert knowledge. For most real-world settings, the parameters can not be chosen just like this, but they need to be selected by an automated routine, similarly to the Sklearn's *grid search*.

WandB sweep

- equivalent to Sklearn's grid search
- orchestrated by WandB library
- independent from your ML library
 - can vary **any** parameter
- can order by a single performance metric
- these settings will lead to 48 individual training runs

Configuration

```
1 # define parameters for hyperparameter sweep
2 entity="<your account>"
3 project="mle-sweep"
4 model_file_name="separation"
5
6 sweep_config = {
7     "method": "grid", # or "random"
8     "metric": {"name": "epoch_loss", "goal": "
9         minimize"},
10    "parameters": {
11        "num_hidden": {"values": [1, 2, 3]},
12        "hidden_dim": {"values": [2, 3, 4, 5]},
13        "learning_rate": {"values": [0.01, 0.1, 0.2,
14        0.5]},
15    }
16 }
```

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Hyperopt

Hyperparameter optimization — sweeps

2025-10-11



Parametrized neural net structure

```
1 class NeuralNetwork(nn.Module):
2
3     def __init__(self, input_dim, output_dim, hidden_dim, num_hidden):
4
5         super().__init__()
6
7         layers = [nn.Linear(input_dim, hidden_dim)]
8
9         for i in range(num_hidden):
10             layers.append(nn.Sigmoid())
11             layers.append(nn.Linear(hidden_dim, hidden_dim))
12
13         layers.append(nn.Sigmoid())
14
15         layers.append(nn.Linear(hidden_dim, output_dim))
16
17         self.model = nn.Sequential(*layers)
18
19     def forward(self, x):
20
21         return self.model(x)
```

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Hyperopt

Hyperparameter optimization — flexible neural net structure

2025-10-11

```
Parametrized neural net structure
class NeuralNetwork(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dim, num_hidden):
        super().__init__()
        layers = [nn.Linear(input_dim, hidden_dim)]
        for i in range(num_hidden):
            layers.append(nn.Sigmoid())
            layers.append(nn.Linear(hidden_dim, hidden_dim))
        layers.append(nn.Sigmoid())
        layers.append(nn.Linear(hidden_dim, output_dim))
        self.model = nn.Sequential(*layers)
    def forward(self, x):
        return self.model(x)
```


WandB sweep

- put whole training instance into train()

train() function

```
1 def train(config=None):
2
3     with wandb.init(config=config):
4
5         # here you get your configuration for this instance
6         config = wandb.config
7
8         # <init model>
9         # <init loss>
10        # <init optimizer>
11        # <train>
12
13        # trained model -> upload wandb artifact
14        model_file="models/{}.pt2".format(model_file_name)
15        torch.save(model.state_dict(), model_file)
16        artifact = wandb.Artifact('model', type='model')
17        artifact.add_file(model_file)
18        wandb.log_artifact(artifact)
```

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Hyperopt

Sweep orchestration — train()

Sweep orchestration — train()

```
def train(config=None):
    with wandb.init(config=config):
        # here you get your configuration for this instance
        config = wandb.config

        # <init model>
        # <init loss>
        # <init optimizer>
        # <train>

        # trained model -> upload wandb artifact
        model_file="models/{}.pt2".format(model_file_name)
        torch.save(model.state_dict(), model_file)
        artifact = wandb.Artifact('model', type='model')
        artifact.add_file(model_file)
        wandb.log_artifact(artifact)
```

WandB sweep

- run the sweep and wait
- observe live at wandb.ai
- ... wait
- ... wait
- check the Results
- download the best-performing model

Running the sweep

```
1 sweep_id = wandb.sweep(sweep_config, project=project)
2
3 wandb.agent(sweep_id, function=train)
```

Retrieving the best model

```
1 api = wandb.Api()
2
3 sweep = api.sweep("{} / {} / {}".format(entity, project,
4 sweep_id))
5 best_run = sweep.best_run()
6
7 # download the artifacts
8 for artifact in best_run.logged_artifacts():
9     if artifact.type == "model":
10         artifact_path = artifact.download()
11         print(artifact_path)
```

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Hyperopt

Sweep orchestration — running the sweep

2025-10-11



WandB sweep monitoring

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex
XOR

Monitoring

Hyperopt

Hugging-
face

Sweep: 7bu6vv71

Search panels with regex

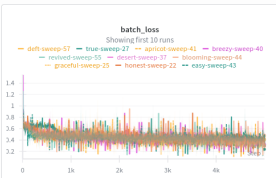
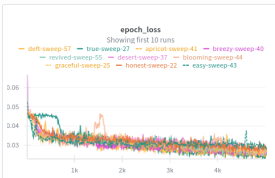
Settings + New report + Add panels

Search runs

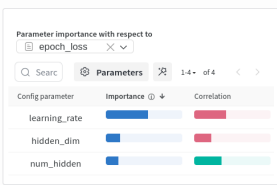
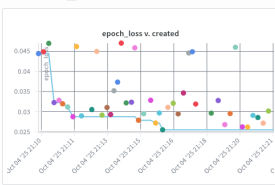
Name 60 visualized

- def-sweep-57
- true-sweep-27
- apricot-sweep-41
- breezy-sweep-40
- revived-sweep-55
- desert-sweep-37
- blooming-sweep-44
- graceful-sweep-25
- honest-sweep-22
- easy-sweep-43
- clean-sweep-8
- giddy-sweep-10
- rich-sweep-42
- desert-sweep-49
- devoted-sweep-13

Charts



Sweep



+ Add section

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

35 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Hyperopt

WandB sweep monitoring

2025-10-11

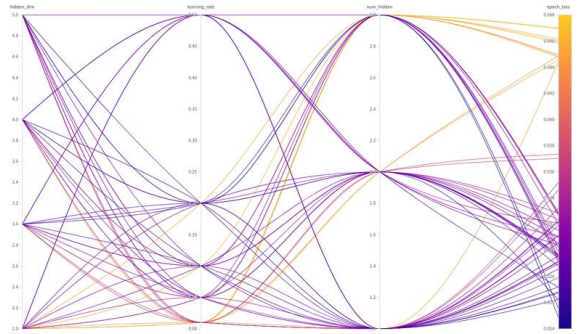
WandB sweep monitoring



Results?

- hidden dim = 5 neurons
- num hidden = 3 layers
- learning rate = 0.2
- duration 1 hour
 - 1 minute per instance
- now we might want to train with these parameters more carefully
 - more epochs
 - fine-tuning for accuracy / recall / precision
 - data augmentation for better fit

How parameters affect the loss?



Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Hyperopt

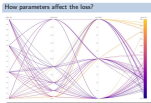
WandB sweep — results

2025-10-11

WandB sweep — results

Results?

- hidden dim = 5 neurons
- num hidden = 3 layers
- learning rate = 0.2
- duration 1 hour
 - 1 minute per instance
- now we might want to train with these parameters more carefully
 - more epochs
 - fine-tuning for accuracy / recall / precision
 - data augmentation for better fit



Results after the sweep?

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

Monitoring

Hyperopt

Hugging-
face

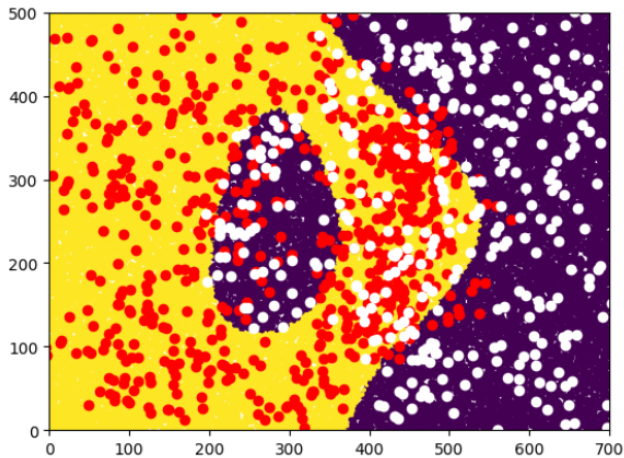
After automated sweep

- better separation
- less biased towards the yellow class
- **parameters were chosen methodically**

WandB

- helps to spot early problems during learning
- you don't have to mess with native GUI
 - often GUI is native not a viable option
- backs up your results (metaresults)

Results



Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

37 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Hyperopt

Results after the sweep?

2025-10-11

Results after the sweep?

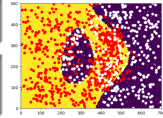
After automated sweep

- better separation
- less biased towards the yellow class
- **parameters were chosen methodically**

WandB

- helps to spot early problems during learning
- you don't have to mess with native GUI
 - often GUI is native not a viable option
- backs up your results (metaresults)

Results



PyTorch tools?

Lecture 3:
Deep
learning
engineer-
ing basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

Monitoring

Hyperopt

Hugging-
face

Common Layers

- Linear (a.k.a. fully-connected)
- Convolutional (signal processing, image processing)
- Pooling (reduce dimensionality)
- Activation (Sigmoid, ReLU, Tanh)

Advanced Layers

- Recurrent (time-series processing and modelling)
- Normalization (stable training, e.g., with ReLU)
- Dropout (random neuron dropouts for regularization)
- Embedding (categorical encoding, NLP)
- Upsampling (generative models)
- Transformer (w/ Attention, TransformerEncoder, TransformerDecoder)

Optimizers

- Stochastic Gradient Descent (SGD)
- ADAM
- LBFGS — 2nd order method, small problems
- ... tens of others

Loss functions

- Mean Square Error (MSELoss)
- Absolute Mean Error (L1)
- Cross Entropy
- Binary Cross Entropy (+ With Logits)
- Cosine Similarity
- ... tens of others

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

38 / 42

Lecture 3: Deep learning engineering basics with PyTorch

PyTorch

Hyperopt

PyTorch tools?

2025-10-11

PyTorch tools?

Common Layers

- Linear (a.k.a. fully-connected)
- Convolutional (signal processing, image processing)
- Pooling (reduce dimensionality)
- Activation (Sigmoid, ReLU, Tanh)

Advanced Layers

- Recurrent (time-series processing and modelling)
- Normalization (stable training, e.g., with ReLU)
- Dropout (random neuron dropouts for regularization)
- Embedding (categorical encoding, NLP)
- Upsampling (generative models)
- Transformer (w/ Attention, TransformerEncoder, TransformerDecoder)

Optimizers

- Stochastic Gradient Descent (SGD)
- ADAM
- LBFGS — 2nd order method, small problems
- ... tens of others

Loss Functions

- Mean Square Error (MSELoss)
- Absolute Mean Error (L1)
- Cross Entropy
- Binary Cross Entropy (+ With Logits)
- Cosine Similarity
- ... tens of others

Models

- models for variety all the common frameworks
- models from notable creators (Google, Meta, Microsoft, OpenAI)
- examples for running the models
- many distributed using Huggingface's own python library
- tracking of finetuned models
- notable models:
 - OpenAI's GPT
 - Google's OWL
 - Meta's Llama

Datasets

- all kinds of datasets for ML
- *common datasets*: mostly images, text, audio, tabular
- distributed using Huggingface's own python library

Spaces

- live demos of ML models
- whole "apps" to play with
- great for inspiration

Models • models for variety of the common frameworks • models from notable creators (Google, Meta, Microsoft, OpenAI) • examples for running the models • many distributed using Huggingface's own python library • tracking of finetuned models • notable models • OpenAI's GPT • Google's OWL • Meta's Llama	Datasets • all kinds of datasets for ML • <i>common datasets</i>: mostly images, text, audio, tabular • distributed using Huggingface's own python library	Spaces • live demos of ML models • whole "apps" to play with • great for inspiration
---	---	---

Try using a model via Huggingface

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

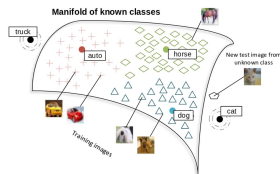
Monitoring

Hyperopt

Hugging-
face

google/owlv2-large-patch14

- zero-shot open vocabulary detector
 - can recognize objects it was not trained on
- will fit in *normal-ish* GPU
- similar mechanism as noted in **lecture 01**



Jupyter Notebook

The following part of the lecture is accompanied by

06_huggingface_owl.ipynb

```
import requests
from PIL import Image
import torch

from transformers import OwlV2Processor, OwlV2ForObjectDetection

processor = OwlV2Processor.from_pretrained("google/owlv2-large-patch14")
model = OwlV2ForObjectDetection.from_pretrained("google/owlv2-large-patch14")

url = "http://images.cocodataset.org/val2017/000000039769.jpg"
image = Image.open(requests.get(url, stream=True).raw)
texts = [{"a photo of a cat", "a photo of a dog"}]
inputs = processor(text=texts, images=image, return_tensors="pt")

with torch.no_grad():
    outputs = model(**inputs)

# Target image sizes (height, width) to rescale box predictions [batch_size, 2]
target_sizes = torch.Tensor([image.size[::-1]])
# Convert outputs (bounding boxes and class logits) to Pascal VOC Format (xmin, y
results = processor.post_process_object_detection(outputs=outputs, target_sizes=t
i = 0 # Retrieve predictions for the first image for the corresponding text que
text = texts[i]
boxes, scores, labels = results[i]["boxes"], results[i]["scores"], results[i]["l
for box, score, label in zip(boxes, scores, labels):
    box = [round(i, 2) for i in box.tolist()]
    print(f'Detected {text[label]} with confidence {round(score.item(), 3)} at l
```

<https://huggingface.co/google/owlv2-large-patch14#use-with-transformers>

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

40 / 42

Lecture 3: Deep learning engineering basics with PyTorch

└─ Hugging-face

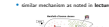
└─ Try using a model via Huggingface

2025-10-11

Try using a model via Huggingface

google/owlv2-large-patch14

- zero-shot open vocabulary detector
- can recognize objects it was not trained on
- will fit in *normal-ish* GPU
- similar mechanism as noted in **lecture 01**



Jupyter Notebook

The following part of the lecture is accompanied by

06_huggingface_owl.ipynb

```
import requests
from PIL import Image
import torch

from transformers import OwlV2Processor, OwlV2ForObjectDetection

processor = OwlV2Processor.from_pretrained("google/owlv2-large-patch14")
model = OwlV2ForObjectDetection.from_pretrained("google/owlv2-large-patch14")

url = "http://images.cocodataset.org/val2017/000000039769.jpg"
image = Image.open(requests.get(url, stream=True).raw)
texts = [{"a photo of a cat", "a photo of a dog"}]
inputs = processor(text=texts, images=image, return_tensors="pt")

with torch.no_grad():
    outputs = model(**inputs)

# Target image sizes (height, width) to rescale box predictions [batch_size, 2]
target_sizes = torch.Tensor([image.size[::-1]])
# Convert outputs (bounding boxes and class logits) to Pascal VOC Format (xmin, y
results = processor.post_process_object_detection(outputs=outputs, target_sizes=t
i = 0 # Retrieve predictions for the first image for the corresponding text que
text = texts[i]
boxes, scores, labels = results[i]["boxes"], results[i]["scores"], results[i]["l
for box, score, label in zip(boxes, scores, labels):
    box = [round(i, 2) for i in box.tolist()]
    print(f'Detected {text[label]} with confidence {round(score.item(), 3)} at l
```


Try using a model via Huggingface

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

Monitoring

Hyperopt

Hugging-
face

How to

- the code can be used as it is
- the model is automatically downloaded (approx. 2 GB)

query: ["cat"]



Plotting the results

```
1 # draw the bounding boxes over the image
2 draw_image = image.copy()
3 draw = ImageDraw.Draw(draw_image)
4
5 # Plot bounding boxes and text
6 for box, score, label in zip(boxes, scores, labels)
7 :
8
9     if score < 0.4:
10         continue
11
12     box = [round(i, 2) for i in box.tolist()]
13
14     xmin, ymin, xmax, ymax = box
15     label_text = f"{texts[0][label]}: {round(score,
16         item(), 2)}"
17     draw.rectangle([xmin, ymin, xmax, ymax],
18         outline="white", width=3)
19     draw.text((xmin, ymin), label_text, fill="white",
20         )
21
22 # Show with matplotlib
23 plt.figure(figsize=(8,8))
24 plt.imshow(draw_image)
25 plt.axis("off")
```

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

41 / 42

Lecture 3: Deep learning engineering basics with PyTorch

└─ Hugging-face

└─ Try using a model via Huggingface

Try using a model via Huggingface

How to:
• the code can be used as it is
• the model is automatically downloaded (approx. 2 GB)

query: ["cat"]



Plotting the results

```
1 # draw the bounding boxes over the image
2 draw_image = image.copy()
3 draw = ImageDraw.Draw(draw_image)
4
5 # Plot bounding boxes and text
6 for box, score, label in zip(boxes, scores, labels)
7 :
8
9     if score < 0.4:
10         continue
11
12     box = [round(i, 2) for i in box.tolist()]
13
14     xmin, ymin, xmax, ymax = box
15     label_text = f"{texts[0][label]}: {round(score,
16         item(), 2)}"
17     draw.rectangle([xmin, ymin, xmax, ymax],
18         outline="white", width=3)
19     draw.text((xmin, ymin), label_text, fill="white",
20         )
21
22 # Show with matplotlib
23 plt.figure(figsize=(8,8))
24 plt.imshow(draw_image)
25 plt.axis("off")
```

2025-10-11

Try using a model via Huggingface

Lecture 3:
Deep
learning
engineering
basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

Monitoring

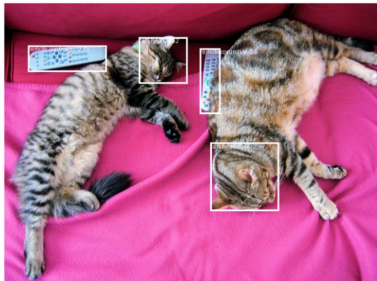
Hyperopt

Hugging-
face

How to

- the code can be used as it is
- the model is automatically downloaded (approx. 2 GB)

query: ["cat's head", "remote controller"]



Plotting the results

```
1 # draw the bounding boxes over the image
2 draw_image = image.copy()
3 draw = ImageDraw.Draw(draw_image)
4
5 # Plot bounding boxes and text
6 for box, score, label in zip(boxes, scores, labels):
7     :
8
9     if score < 0.4:
10         continue
11
12     box = [round(i, 2) for i in box.tolist()]
13
14     xmin, ymin, xmax, ymax = box
15     label_text = f"{texts[0][label]}: {round(score,
16         item(), 2)}"
17     draw.rectangle([xmin, ymin, xmax, ymax],
18         outline="white", width=3)
19     draw.text((xmin, ymin), label_text, fill="white",
20         )
21
22 # Show with matplotlib
23 plt.figure(figsize=(8,8))
24 plt.imshow(draw_image)
25 plt.axis("off")
```

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

41 / 42

Lecture 3: Deep learning engineering basics with PyTorch

Hugging-face

Try using a model via Huggingface

2025-10-11

Try using a model via Huggingface

How to:
• the code can be used as it is
• the model is automatically downloaded (approx. 2 GB)

query: ["cat's head", "remote controller"]



Plotting the results

```
1 # draw the bounding boxes over the image
2 draw_image = image.copy()
3 draw = ImageDraw.Draw(draw_image)
4
5 # Plot bounding boxes and text
6 for box, score, label in zip(boxes, scores, labels):
7     :
8
9     if score < 0.4:
10         continue
11
12     box = [round(i, 2) for i in box.tolist()]
13
14     xmin, ymin, xmax, ymax = box
15     label_text = f"{texts[0][label]}: {round(score,
16         item(), 2)}"
17     draw.rectangle([xmin, ymin, xmax, ymax],
18         outline="white", width=3)
19     draw.text((xmin, ymin), label_text, fill="white",
20         )
21
22 # Show with matplotlib
23 plt.figure(figsize=(8,8))
24 plt.imshow(draw_image)
25 plt.axis("off")
```

Try using a model via Huggingface

Lecture 3:
Deep
learning
engineering basics
with
PyTorch

Tomáš
Báča

Python
Environ-
ment

Deep
learning
libraries

PyTorch

Simple ANN

Complex

XOR

Monitoring

Hyperopt

Hugging-
face

How to

- the code can be used as it is
- the model is automatically downloaded (approx. 2 GB)

query: ["drone"]



Plotting the results

```
1 # draw the bounding boxes over the image
2 draw_image = image.copy()
3 draw = ImageDraw.Draw(draw_image)
4
5 # Plot bounding boxes and text
6 for box, score, label in zip(boxes, scores, labels)
7 :
8     if score < 0.4:
9         continue
10
11     box = [round(i, 2) for i in box.tolist()]
12
13     xmin, ymin, xmax, ymax = box
14     label_text = f"{texts[0][label]}: {round(score,
15         item(), 2)}"
16     draw.rectangle([xmin, ymin, xmax, ymax],
17         outline="white", width=3)
18     draw.text((xmin, ymin), label_text, fill="white",
19         )
20
21 # Show with matplotlib
22 plt.figure(figsize=(8,8))
23 plt.imshow(draw_image)
24 plt.axis("off")
```

Tomáš Báča (CTU in Prague)

Lecture 3: Deep learning engineering basics with PyTorch

October 6th, 2025

41 / 42

Lecture 3: Deep learning engineering basics with PyTorch

└─ Hugging-face

└─ Try using a model via Huggingface

Try using a model via Huggingface



2025-10-11

Thanks for your attention

Lecture 3: Deep learning engineering basics with PyTorch

└─ Hugging-face

└─ Conclusion

Conclusion

Thanks for your attention