UNIVERSITÀ DI PISA

Master Degree in Computer Science

Machine Learning Course

Building and training machine learning models in Python with keras and torch

Alessandro Dipalma

alessandro.dipalma@phd.unipi.it

Outline

1. Arrays and tensors
2. Overview of TensorFlow and Keras
3. PyTorch
   - Sequential APIs
   - Autograd mechanism
   - The pytorch computational graph
4. Wrap up and advices

Hands on session!

shorturl.at/QR57g

The need for frameworks

ML Frameworks simplify the process of:

- Building complex machine learning model.
  (Not just Neural Networks!)
- Training efficiently on GPUs/TPUs.
- Managing deployment pipelines.

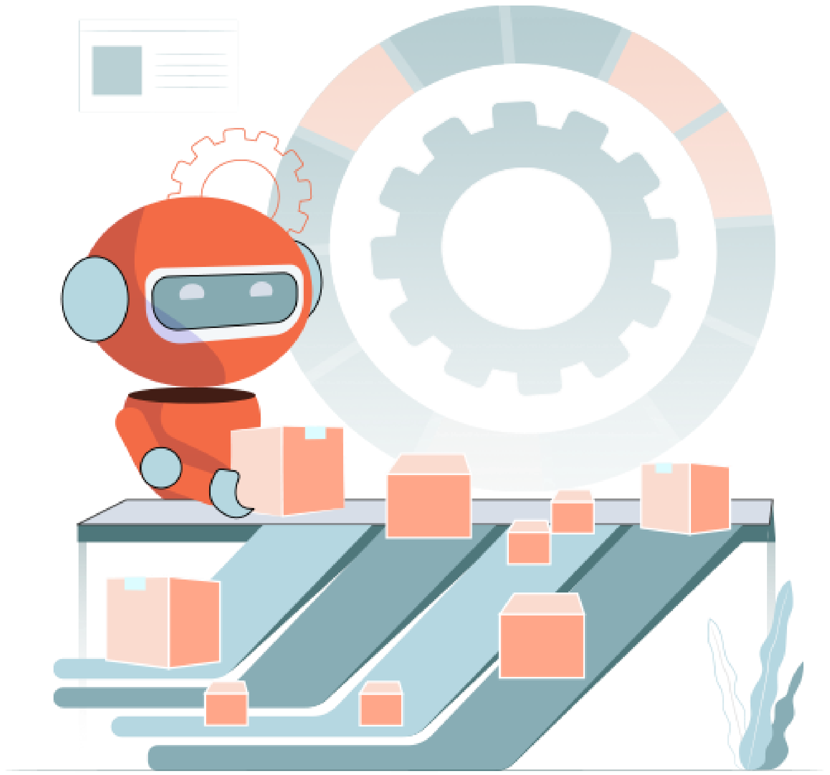
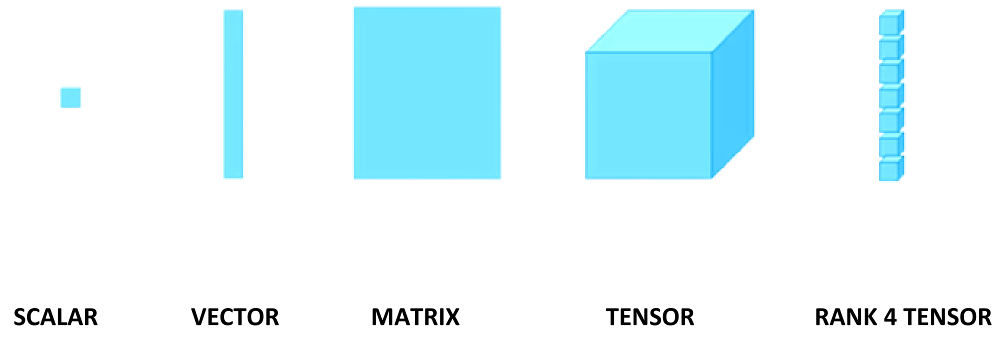


Image credits https://www.neoito.com/blog/top-machine-learning-frameworks/

Tensors

At the core of ML frameworks lies the Tensor data structure.

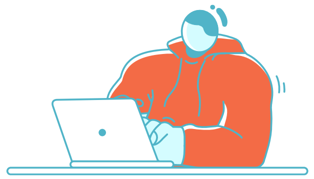
- Similar to NumPy multidimensional arrays, from which they inherit semantics.
- Optimized to run on hardware accelerators (GPUs, TPUs)
- Optimized for automatic differentiation: use them only if you need the structure, otherwise operations on numpy array are generally faster.



Tensor Broadcasting

Broadcasting is the mechanism that allows Tensors with different shapes to be processed together (e.g., added, multiplied).

- The smaller Tensor is "virtually" copied along the missing dimensions to match the larger one.
- It avoids creating unnecessary copies of data in memory, though it introduces a somewhat magic semantic that usually is the cause of a lot of bugs



TensorFlow

History: Released by Google Brain in 2015.

Key Features:

- High-level APIs like Keras.
- Comes with an entire ecosystem to ease:
- *Visualization*
- *Support for distributed training and TPU acceleration.*
- *Distribution*

Keras

Born as high level interface for TensorFlow, but currently supports multiple backends: Tensorflow, Torch and JAX.

Provides multiple ways to build NN architecture:

- *Sequential APIs*
- *Functional APIs*
- *Subclassing (Torch-like)*

Keras

Sequential vs Functional APIs

SEQUENTIAL: A linear stack of layers. One input tensor, one output tensor.

Pros:

- Extremely simple, concise, and readable.
- Great for standard architectures (e.g., VGG-style CNNs, simple MLPs).

Cons: Inflexible. Cannot handle:

- Multiple inputs/outputs (e.g., multimodal learning).
- Residual connections (ResNet) or skipping layers.
- Shared layers (siamese networks).

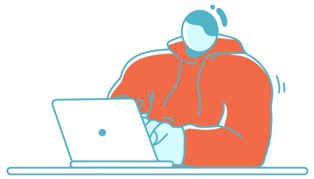
FUNCTIONAL: Treats layers as functions that map tensors to tensors. Builds a directed acyclic graph (DAG).

Pros:

- Handles complex topologies (non-linear).
- Essential for modern architectures like ResNet, Inception, or Transformers.

Cons:

- Slightly more verbose.



PyTorch

Released by Facebook in 2016.

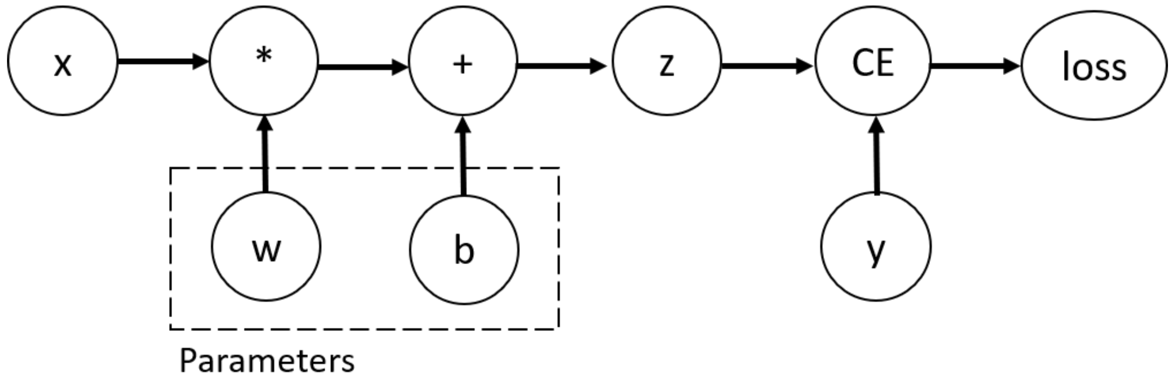
Key Features:

- Dynamic computation graphs.
- Strong debugging capabilities.
- Modules like torchvision for vision tasks.

PyTorch

Automatic Differentiation

Autograd is PyTorch's automatic differentiation engine that powers neural network training. It allows to compute gradients automatically, which is essential for optimizing model parameters using gradient-based methods.



Picture from pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html

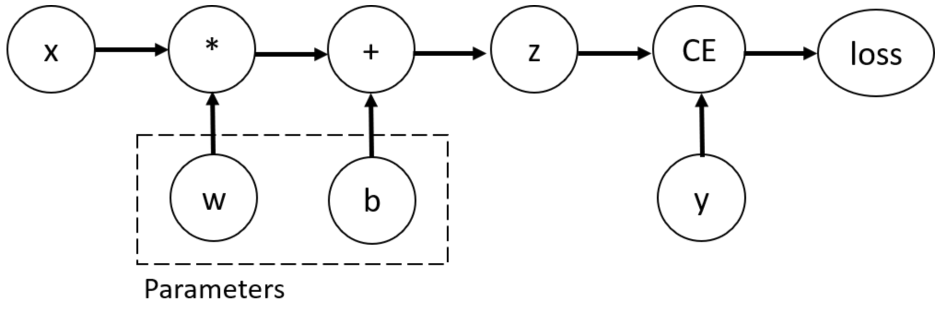
PyTorch

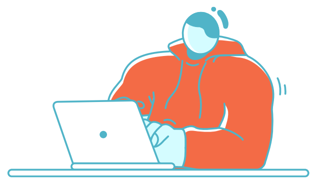
Automatic Differentiation

1. **Forward Pass**
   Autograd records operations in the computation graph.
   Nodes represent tensors; edges represent functions applied to those tensors.

2. **Backward Pass**
   Starts with a scalar loss tensor calling backward().
   Gradients are propagated backward along the graph, using the chain rule.

3. **Key Methods**
   `backward()`: Computes gradients of a scalar loss.
   `tensor.grad()`: attribute: Holds the computed gradient for each tensor.
   `torch.no_grad()`: Context manager for disabling gradient computation.





Picture from pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html

PyTorch

Neural Networks

- Architectural building blocks:
- *Neural layers. Can define custom ones by extending* `nn.Module` *class*
- *Transformation layers: padding, pooling, normalization…*
- *Activation functions:* `nn.ReLu, nn.Tanh, nn.Sigmoid`*…*
- Loss functions: `nn.L1Loss, nn.MSELoss, nn.CrossEntropyLoss`…
- Optimizers in `torch.optim`: `SGD, Adam, RMSProp`…

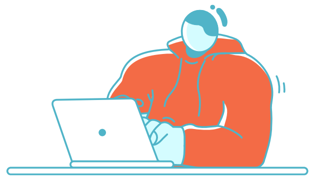
PyTorch

Training Loop

What is cool about otrch is that we have complete flexibility about how we want to train our models.

Key components:

- Dataloaders (and Datasets)
- Training step
- Optimization
- Evaluation
- Callbacks



PyTorch

“Callbacks”

Differently from Keras, where you just pass the callbacks=[] parameter to model builder you have to manage «by hand» stuff like checkpointing and early stopping.

```
patience = counter = 0
for epoch in range(epochs):
    # ... training and validation ...
    if vloss < best_vloss:
        best_vloss = vloss
        counter = 0
    else:
        counter += 1
    if counter >= patience:
        print("Early Stopping!")
        break
```

```
best_vloss = float('inf’)
for epoch in range(epochs):
    train_one_epoch()
    vloss = validate()
    # Logica di Checkpointing
    if vloss < best_vloss:
        best_vloss = vloss
        # Save only weights
        torch.save(model.state_dict(), 'best_model.pth')
```

PyTorch

Schedulers

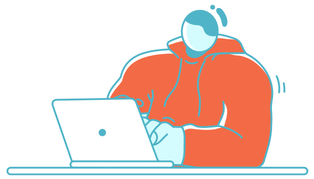
- A fixed Learning Rate (LR) is often inefficient: too high and it oscillates; too low and it converges slowly.
- Schedulers adjust the LR during training, usually decreasing it to allow fine-tuning as the model approaches the global minimum.

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
# Example: Reduce LR if validation loss stops improving
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=5)

for epoch in range(epochs):
    train(...)
    val_loss = validate(...)

    # Step is called AFTER validation
    scheduler.step(val_loss)
```

Let’s classify penguins!



Concluding remarks

Wrap up: TF vs PT

	PyTorch	TensorFlow
EASE OF USE	DYNAMIC COMPUTATIONAL GRAPH, RESEARCH PURPOSE, IMPERATIVE	EAGER EXECUTION 2.0, HISTORICALLY USED FOR LARGE-SCALE DOMAINS, APIs for C++, Java
FLEXIBILITY	EASIER TO DESIGN CUSTOM ARCHITECTURES AND COMPLEX MODEL	FAST PROTOTYPIZATION OF KNOWN ARCHITECTURES AND DEFAULT TRAINING CYCLES
POPULARITY	RAPID POPULARITY GAIN ESPECIALLY IN ACADEMIC FIELDS AND RESEARCH MODELS.	LONG-STANDING REPUTATION IN INDUSTRY DOMAINS
DEPLOYMENT AND PRODUCTION	TORCH SCRIPT & TORCH SERVE	TENSORFLOW SERVING AND TF LITE

Some personal suggestions

- **Train yourself before the models!** Spend your time in reading the library documentation: popular frameworks come with plenty of examples and tutorials that cover most base use cases. You'll be glad to have read them before searching stackoverflow or asking generative friends for any random looking error that WILL come out during the project development.
- Pay attention to the broadcasting sematics, it usually creates a lot of bugs.

Tools that make life easier

- A logging board (Neptune, Wandb, Tensorboard...) can help you to keep track of your experiments and produce those fancy plots you see allover the web.
- A library to effectively exploit multiprocessing in hyperparameter tuning, even just for your laptop (I like Ray Tune).
- PyTorch Lightning is a framework born to avoid writing boilerplate (and often inefficient) torch code. Provides high level classes in keras/tensorflow fashion, but for PyTorch.

References

- TensorFlow Documentation
- PyTorch Documentation

Thank you for the attention!