IMPLEMENTING AND TRAINING NEURAL NETWORKS(CONT.2)

STFANO MELACCI

UNVERSITY OF SIENA

PYTORCH, TENSORFLOW, …

- There exists several software libraries that can be used to easily implement Neural Networks and train them using gradient-based optimization (frequently - non always - with Python support)
  - PyTorch https://pytorch.org/
  - TensorFlow https://www.tensorflow.org/
  - scikit-learn https://scikit-learn.org/stable/
  - Apache MXNet https://mxnet.apache.org/
  - MATLAB (Deep Learning Toolbox) https://www.mathworks.com/discovery/neural-network.html
  - …
- In turn, they usually exploit other lower level libraries for linear algebra or multi-threading
  - BLAS (OpenBLAS), LAPACK, Intel MKL, ATLAS, …
  - Notice that this is also the case of NumPy

PYTORCH, TENSORFLOW, …

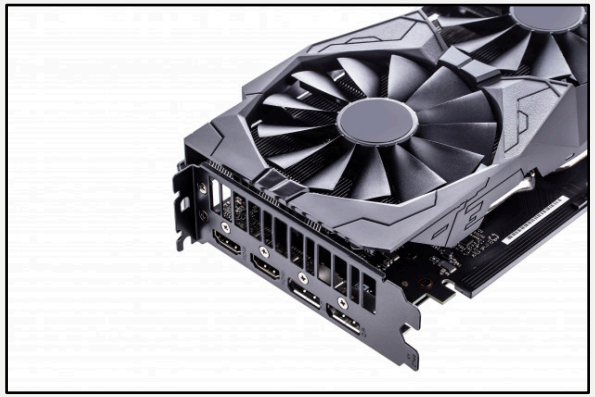
- We already discussed some NumPy-based code
  - Using NumPy we have access to common linear algebra operation and to tensor manipulation tools
- Some common questions:
  - How are these software libraries related to NumPy?
  - What do they offer that NumPy doesn't offer?
  - Which one should I use?
  - Are they so different one from each other?
  - …

PYTORCH, TENSORFLOW VS. NUMPY

- Let's consider PyTorch or TensorFlow as running examples
  - We will mostly consider PyTorch at a later stage
- These libraries are not so different from NumPy
  - They define their own Tensor format…
  - …that is either fully compatible with NumPy tensors or that can be easily converted to NumPy
    - *They keep a strong link with NumPy data format, providing widely used conversion tools*
  - They implement fast linear algebra facilities and tensor manipulation functions
    - *That are pretty similar to those of NumPy (sometimes we have very-very similar function names)*
    - Parameters might have different names; with a little bit of tolerance it is easy to pass from NumPy to these libraries and vice-versa

PYTORCH, TENSORFLOW VS. NUMPY

- However, they have some features that NumPy doesn't have!
  - The tensor data can be moved to the **GPU** memory and operations can be performed on GPUs as well
  - This is obtained with a few lines of code that specify that we are going to use a different device (we will see more details about it)
  - GPU = Fast computation on "large" scale data
    - Up to what the GPU memory can handle
  - How does one usually program GPUs?
    - *NVIDIA* cards and **CUDA** vs. rest of the world



PYTORCH, TENSORFLOW VS. NUMPY

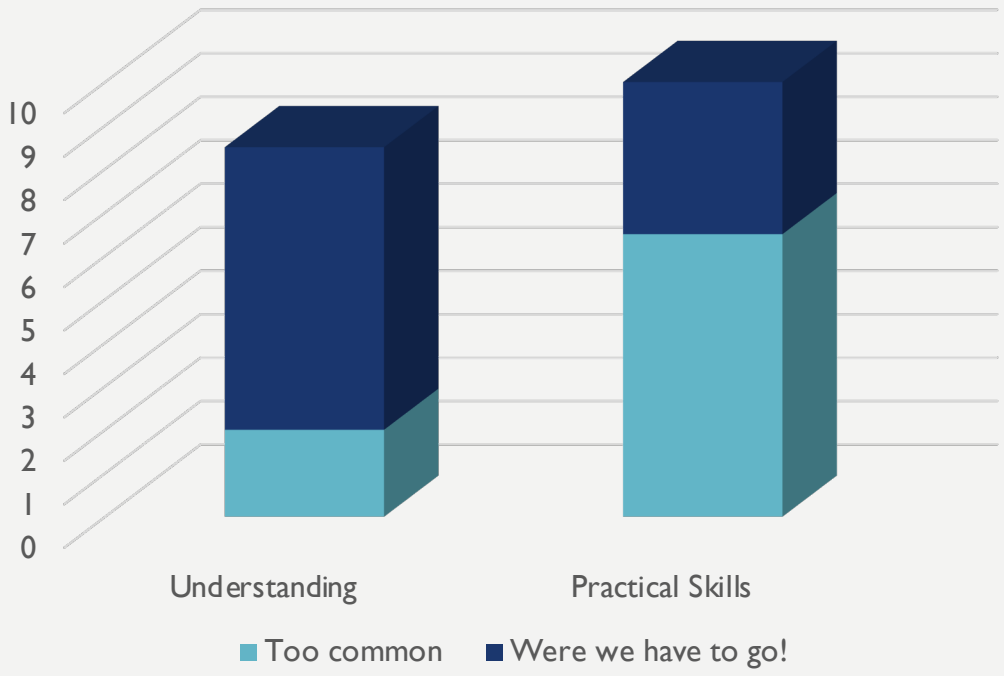
- However, they have some features that NumPy doesn't have! (cont.)
  - They perform automatic differentiation (**AutoGrad**)
    - No need to implement the chain rule ☺
    - Getting the gradients of a loss function w.r.t. to some tensors is trivial
  - How can they do that?
    - They build a **computational graph** on the fly, without showing it to the user/programmer
    - When the programmer asks for a gradient computation, they apply the chain rule traversing the computational graph in the opposite direction (as we manually did in the case of NumPy)
    - Each of their lower level operations and functions has the derivatives w.r.t. its arguments hard-coded
    - They keep track of the computed gradients, and, sometimes, they perform some optimization for faster computations (it is not always the case – the computational graph does introduce a small overhead)
    - *Side note: we have to talk about TensorFlow 1.x vs 2.x…*

PYTORCH, TENSORFLOW VS. NUMPY

- However, they have some features that NumPy doesn't have! (cont. cont.)
  - They include higher level structures (classes) that can be used to easily create neural networks
  - The provide an easy-to-use interface to train a neural model, to evaluate it, hiding the math that is behind the neural-network-related computations
    - **Keras**, "*an API designed for human beings, not machines*", has been fully integrated into *TensorFlow* 2.x https://keras.io/
    - *PyTorch* has its own interface
  - This speeds up the development of those software solutions that exploits neural network
  - Some details might get lost (major warning for newbies!)

DEEP UNDERSTANDING VS. PRACTICAL SKILLS

- A very old story…
- Short version:
  - Gain understanding to improve your practical skills as well
    - New ideas, more energy!
  - Avoid just cloning code of other people, running it, and declaring that you are done!
    - Sometimes, too high-level software interfaces lead to this (bad) ending…

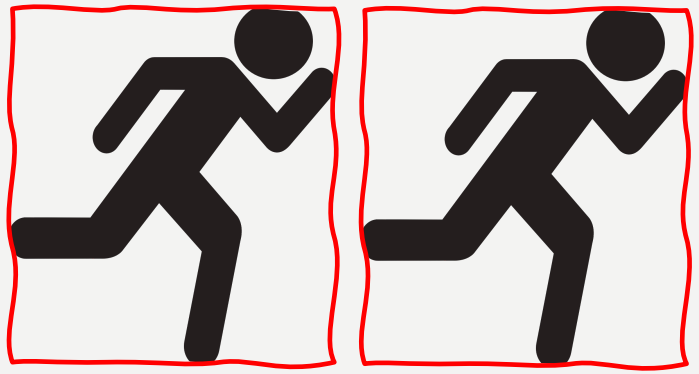


CODE BREAK: SLOWLY MOVING TO PYTORCH AND TENSORFLOW

- Let's have a look at the code!
  - PyTorch-based implementation of
    - Feed-forward neural networks
    - Gradient-based weight updates
  - TensorFlow (>= 2.0) implementation of
    - Feed-forward neural networks
    - Gradient-based weight updates

UNLEASH THE POWER OF GPUS!

- Small data does not allow us to see the real benefits of using GPUs to speedup computations
- They are designed to strongly parallelize larger-scale operations
  - Thousands of **parallel** threads!
  - Watch out for the way data are stored in **memory**
  - **Time** to go back and froth to/from the main (CPU) memory of the machine



Two CPU cores vs. several GPU threads

CODE BREAK: CPU VS. GPU

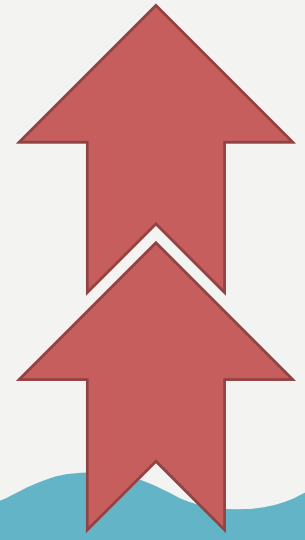
- Let's have a look at the code!
  - Matrix-by-matrix multiplication
  - CPU vs. GPU
    - NumPy, PyTorch, TensorFlow (>= 2.0)

NEURAL NETWORKS: PYTORCH

- Basic operations on ***torch.Tensor*** objects
  - torch.*op*(…)
    - https://pytorch.org/docs/stable/torch.html
  - torch.nn.functional.*op*(…) is a neural net-related operation
    - https://pytorch.org/docs/stable/nn.functional.html
- Operations as members of the ***torch.Tensor*** class (example: let's say we have a tensor X)
  - Usually shortcuts to basic operations
  - X.*op*(…) is sometimes equivalent to torch.*op*(X, …)
  - X.*op*_(…) is an **in-place** operation
    - https://pytorch.org/docs/stable/tensors.html

NEURAL NETWORKS: PYTORCH

- Using these operations, we can implement neural networks and all we need to do with them
  - Training, inference, …
  - That's what we saw in the previously discussed examples!
- However, there exists **higher-level classes** that are **fully-based on these operations**, and that help in
  - *hiding the math (that's good and bad)*
  - *plugging operations together*
  - *creating more easily manageable computational blocks*
  - *handling different network layouts*
  - …



If you look at their source code, you will see that they simply call the operations we described in the previous slide!

NEURAL NETWORKS: PYTORCH

- Most common (base) class: ***torch.nn.Module***
  - Example: a network layer is a subclass of a *torch.nn.Module*
  - Example: an MLP neural network is a set of layers, so it is a *torch.nn.Module*
- Basically, a *torch.nn.Module* collects some tensors that are ***learnable variables***, and it implements a ***forward*** method that passes the signal through the module
  - Variables are instances of *torch.nn.parameter.Parameter*
  - They are sub-classes of *torch.Tensor*, so basically a "more decorated" *torch.Tensor*
- Modules can be collected into **containers**, that implement network layouts and pass the signal through all the collected modules (actually this is not much stressed in PyTorch as it was in Torch)
  - Containers are *torch.nn.Module* as well

NEURAL NETWORKS: PYTORCH

- If your net is implemented as a (combination of) *torch.nn.Module*(s), then it will be easy to
  - Get a quick reference to all the network learnable parameters
  - Forward the signal trough the whole network: a *torch.nn.Module* is callable, and, when called, it invokes the ***forward*** method of the module
  - Save and load the network
  - Use and optimizer to update the network parameters
    - *Stochastic Gradient Descent (SGD), Adam, …*
  - …

CODE BREAK: HIDING THE MATH (PYTORCH)

- Let's have a look at the code!
  - Higher-level PyTorch-based implementation of
    - Feed-forward neural networks
    - No math involved at all (in the code)
  - (A) Using custom 'Net' class
  - (B) Using a sequential container

NEURAL NETWORKS: TENSORFLOW

- What about TensorFlow?
  - It follows a similar organization, very similar from the lower-level perspective
  - Basic operations on tensors (as in PyTorch)
  - Higher-level modules (as torch.nn.Module) that are based on ***Keras***
  - From this point of view, it is extremely similar to what we saw about PyTorch
  - *One of the main differences is that Keras allows us to exploit an even higher-level abstraction of the network, optimizer, accuracy measures etc… (we will not cover this at all – the focus of this course is on PyTorch)*

CODE BREAK: HIDING THE MATH (TENSORFLOW)

- Let's have a look at the code!
  - Higher-level TensorFlow (>= 2.0)-based implementation of
    - Feed-forward neural networks
    - No math involved at all (in the code)
  - (A) Using custom 'Net' class
  - (B) Using a sequential container

  Notice: this code could have been written using the Keras facilities, making it shorter and more compact – I kept a more verbose organization to match the PyTorch version