

Python Workshop: Introduction to PyTorch

Léo Laugier



*This presentation is greatly inspired by the 2017 PyTorch Workshop (<https://github.com/mlberkeley/PyTorch-Workshop>) and the 2018 Introduction to Deep Learning (<https://github.com/mlberkeley/intro-dl-workshop>) from Machine Learning at Berkeley.

March 11, 2020

Contents

- 1 Who this workshop is for
- 2 Theoretical background on artificial neural networks and deep learning
- 3 PyTorch: a Python package for training (deep) artificial neural networks
- 4 Automatic differentiation with the `torch.autograd` package
- 5 Training a CNN on MNIST Digit Classification with PyTorch
- 6 Conclusion: PyTorch is convenient for deep learning research

- 1 Who this workshop is for
- 2 Theoretical background on artificial neural networks and deep learning
- 3 PyTorch: a Python package for training (deep) artificial neural networks
- 4 Automatic differentiation with the `torch.autograd` package
- 5 Training a CNN on MNIST Digit Classification with PyTorch
- 6 Conclusion: PyTorch is convenient for deep learning research

Who this workshop is for

- People who know the general ideas of machine / deep learning and machine / deep learning frameworks.
- People who want a ground up introduction to PyTorch

This workshop is not

- A theoretical lecture about deep learning
- An advanced tutorial on PyTorch for PyTorch “connoisseurs”

Contents

- 1 Who this workshop is for
- 2 Theoretical background on artificial neural networks and deep learning
- 3 PyTorch: a Python package for training (deep) artificial neural networks
- 4 Automatic differentiation with the `torch.autograd` package
- 5 Training a CNN on MNIST Digit Classification with PyTorch
- 6 Conclusion: PyTorch is convenient for deep learning research

What's the Difference Between Artificial Intelligence, Machine Learning and Deep Learning?

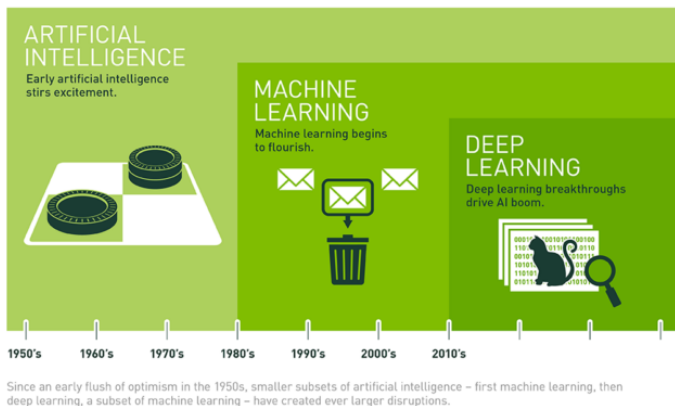


Figure: The Difference Between AI, Machine Learning, and Deep Learning. Figure from <https://blogs.nvidia.com/>

Definition of Deep Learning and Artificial Neural Networks

Definition of Deep Learning (mix between LeCun *et al.* [1], 2015 and Goodfellow *et al.* [2], 2016)

Deep learning is an approach to solve AI problems by allowing computational **models** that are composed of multiple processing layers to learn - from experience - **representations** of data with multiple levels of abstraction and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts. The hierarchy of concepts enables the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is **deep**.

Definition of Artificial Neural Networks (ANNs)

These models, often “vaguely” (Wikipedia) inspired by the structure and function of biological neural networks that constitute animal brains, are called **artificial neural networks**.

Biological neural networks: the Hubel and Wiesel Cat Experiment (1959)



Universal approximation theorem: ANNs can approximate continuous functions

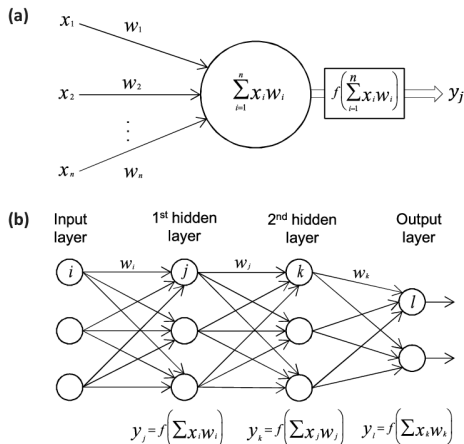


Figure: General architecture of Artificial Neurons (a) and ANNs (b). Figures from Vieira *et al.* [3], 2017

ANNs are trained by optimizing an objective function on a training dataset

Training a neural network

Let f_w be the ANN model with parameters (weights) w .

Let J be the objective (loss, cost) function (e.g. Mean Squared Error for regression, Cross Entropy for classification, ...).

Let $\mathcal{D}_{train} = (x^i, y^i)_{i \in [1, n]}$ be the training dataset.

Optimization problem: $\hat{w} = \arg \min_w \mathbb{E}_{(x, y) \sim \mathcal{D}_{train}} [J(f_w(x), y)]$

Then, classic parametric Machine Learning approach:

- Evaluate on validation set (hyperparameter tuning)
- Test on test set...

The objective function is optimized with Stochastic Gradient Descent

Optimization problem

Minimizing the loss function: $\hat{w} = \arg \min_w \mathbb{E}_{(x,y) \sim \mathcal{D}_{train}} [J(f_w(x), y)]$

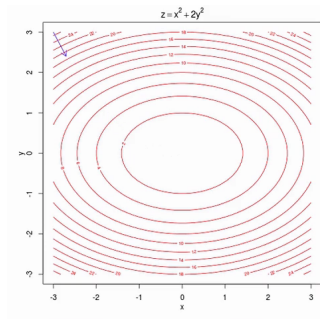
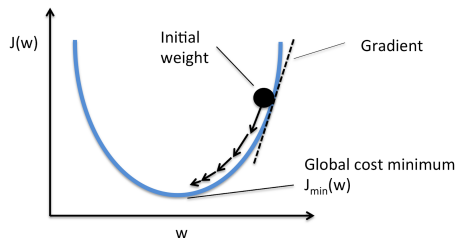
Optimization algorithm

Gradient descent: $w \leftarrow w - \epsilon \frac{\partial J}{\partial w}$

Approximation of the Optimization algorithm

Stochastic Gradient Descent (SGD): replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a **randomly selected subset** of the data).

Gradient descent finds a local minimum of a differentiable function



SGD is computed with backpropagation

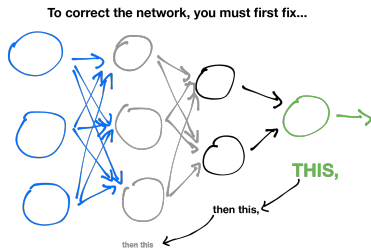


Figure: Illustration of the chain rule

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad (\text{BP1})$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad (\text{BP2})$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (\text{BP3})$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (\text{BP4})$$

Figure: from <https://neuralnetworksanddeeplearning.com>

Contents

- 1 Who this workshop is for
- 2 Theoretical background on artificial neural networks and deep learning
- 3 PyTorch: a Python package for training (deep) artificial neural networks**
- 4 Automatic differentiation with the `torch.autograd` package
- 5 Training a CNN on MNIST Digit Classification with PyTorch
- 6 Conclusion: PyTorch is convenient for deep learning research

We want a framework suitable for deep learning research

We want:

- to be able to define a general computation structure,
- a framework able to **differentiate automatically** and back-propagate gradients efficiently by leveraging deep learning dedicated hardware (**GPUs**, TPUs, model / data parallelism) without worrying about low level computations,
- to quickly **build neural network models**, initialize weights and play with pre-made (CNNs, RNNs, Attentions, ...) and custom architectures (MyToyNN),
- **flexibility** for research purpose, easy testing and debugging,
- very little boilerplate for **common tasks** (mathematical operations, classic architectures, loss functions, optimizers, data pre-processing, dataset loading, ...).

We want a computational graph handling feedforward computations and gradient backpropagation for us

Definition of a computational graph

A **computational graph** is a directed graph where the nodes correspond to **operations** or **variables**. Variables can feed their value into operations, and **operations can feed their output into other operations**. This way, every node in the graph defines a function of the variables.

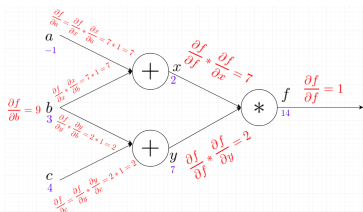


Figure: Backpropagating and finding the gradient of f w.r.t all the variables in the graph by the application of chain rule. Blue elements represent the outputs of the nodes whereas red element represents the gradients that were calculated during backpropagation. Figure from <https://medium.com/spidernitt/>

TensorFlow and PyTorch: 2 packages with distinct approaches

2 main Python packages for deep learning: **TensorFlow** (Google, 2015) and **PyTorch** (Facebook, 2016).

- TensorFlow 1.0 (2017): the computational graph is built “**before compilation**” and run later in a Session (for speed purpose). Note that TensorFlow 2.0 (2019) claims to be more “Pythonesque”.
- PyTorch 1.0 (2018): the computational graph is built **on the fly**. The advantage to define by run is more flexibility for dynamic inputs.

Summary:

- TensorFlow: **Define-then-run**
- PyTorch: **Define-by-run**, installation pytorch.org/

- 1 Who this workshop is for
- 2 Theoretical background on artificial neural networks and deep learning
- 3 PyTorch: a Python package for training (deep) artificial neural networks
- 4 Automatic differentiation with the `torch.autograd` package**
- 5 Training a CNN on MNIST Digit Classification with PyTorch
- 6 Conclusion: PyTorch is convenient for deep learning research

Automatic differentiation: the torch.autograd package

The torch.autograd package (https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html)

The autograd package provides **automatic differentiation** for all **operations** on Tensors. It is a **define-by-run framework**, which means that your backprop is defined by how your code is run, and that every single iteration can be different.

The torch.Tensor class is a multi-dimensional matrix containing elements of a single data type

PyTorch Tensors are similar to NumPy Arrays.

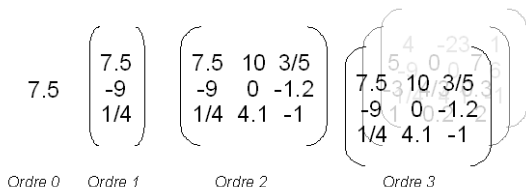


Figure: from [fr.wikipedia](https://fr.wikipedia.org/wiki/Tenseur)

2 kinds of tensors: **constants** (e.g. inputs, targets. No gradient tracking) and **variables** (e.g. ANN's weights to be optimized. **Gradient tracking**)

PyTorch < 0.4.0: **constants** = torch.Tensor, **variables** = torch.Variable
PyTorch ≥ 0.4.0:

constants = torch.tensor(*args, **kwargs, **requires_grad = False**),
variables = torch.tensor(*args, **kwargs, **requires_grad = True**)

Graphics Processing Units (GPUs) speed up feedforward computations and gradient backpropagation

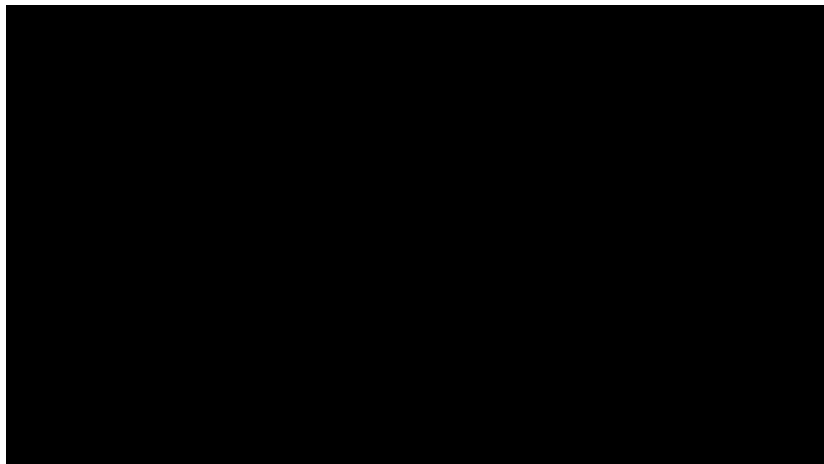
Historically: specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images (see [Real-Time Deformation and Fracture in a Game Environment](#)).

Graphics Processing Units (**GPU**) have **highly parallel structure** that make them more efficient than general-purpose central processing units (CPUs) for algorithms that process **large blocks of data in parallel**: very suitable for operations on tensors!

API: **CUDA** (NVIDIA) that PyTorch uses to load data and models on GPUs: (<https://pytorch.org/docs/stable/notes/cuda.html>)

Other specialized hardware for Deep Learning: Tensor Processing Units (TPUs, Google).

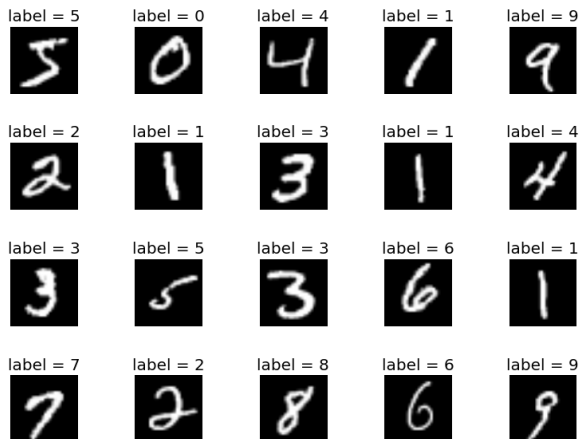
Graphics Processing Units (GPU) speed up feedforward computations and gradients backpropagation



Contents

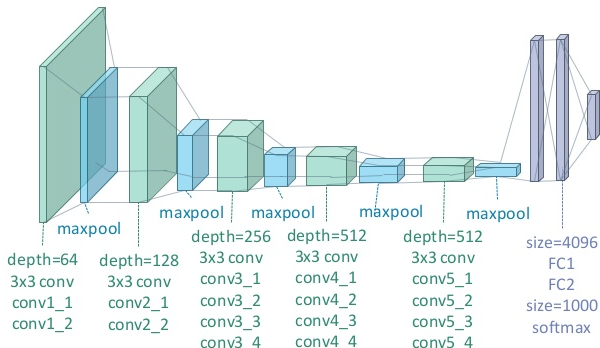
- 1 Who this workshop is for
- 2 Theoretical background on artificial neural networks and deep learning
- 3 PyTorch: a Python package for training (deep) artificial neural networks
- 4 Automatic differentiation with the `torch.autograd` package
- 5 Training a CNN on MNIST Digit Classification with PyTorch**
- 6 Conclusion: PyTorch is convenient for deep learning research

MNIST Digit Classification task



Convolutional Neural Networks are specific neural networks achieving SOTA results in image classification

VGG 19



60

Convolutions have translation invariance characteristics

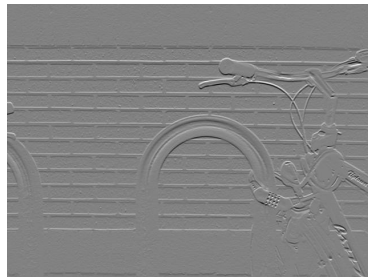
1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

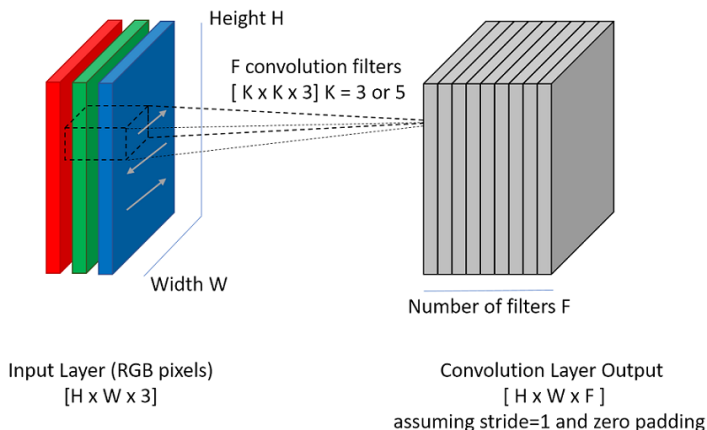
4		

Convolved
Feature

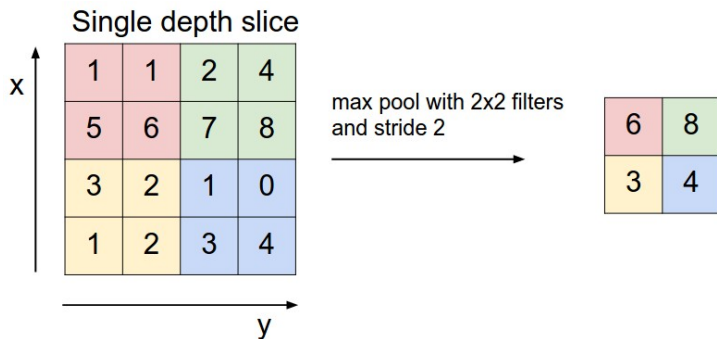
Convolution can be applied for edge detection



Convolutional layers



Pooling layers replace the output of the net at a certain location with a summary statistic of the nearby outputs



Pooling helps to make the representation approximately invariant to small translations of the input.

Training a CNN on MNIST Digit Classification with PyTorch (7/7): Demonstration

Demo (Google Colab Notebook) at bit.ly/3cVPyEy

Contents

- 1 Who this workshop is for
- 2 Theoretical background on artificial neural networks and deep learning
- 3 PyTorch: a Python package for training (deep) artificial neural networks
- 4 Automatic differentiation with the `torch.autograd` package
- 5 Training a CNN on MNIST Digit Classification with PyTorch
- 6 Conclusion: PyTorch is convenient for deep learning research

Conclusion: PyTorch is convenient for deep learning research




- Because of its flexibility, PyTorch is very used in **research**
- PyTorch is **Pythonesque** and **“NumPy-esque”**
- TensorFlow is more optimized for production and **faster at run time** because the computational graph is built before running: you pay(ed -in TF 1.0 at least-) the price during development
- PyTorch being younger than TensorFlow, comparatively little support, fewer StackOverflow questions, etc. You may be one of the first to get any particular error
- Easier to feed layers with variable-length input tensors without necessarily padding...
- Other historic Python packages for deep learning: Caffe (merged in PyTorch), Theano (2007).

Deep Learning and ANNs

- The Deep Learning Book (Goodfellow *et al.* [2], 2016))
- <https://neuralnetworksanddeeplearning.com> (Michael Nielsen)
- Stanford CS230 Deep Learning (Andrew Ng)
- UC Berkeley CS294-129 Designing, Visualizing and Understanding Deep Neural Networks (John Canny) and CS294-131 Special Topics in Deep Learning (Trevor Darrell)
- Coursera Deep Learning Specialization (deeplearning.ai)
- Université Paris-Saclay MVA Master courses on Deep Learning (Vincent Lepetit, Stéphane Mallat, Guillaume Charpiat, Sébastien Gerchinovitz)

Parallel computing, CUDA, GPUs

- UC Berkeley CS267 Applications of Parallel Computers (Jim Demmel)

-  Yann LeCun, Yoshua Bengio, and Geoffrey Hinton.
Deep learning.
Nature, 521(7553):436–444, 2015.
-  Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning.
MIT Press, 2016.
<http://www.deeplearningbook.org>.
-  Sandra Vieira, Walter Pinaya, and Andrea Mechelli.
Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications.
Neuroscience Biobehavioral Reviews, 74, 01 2017.

Python Workshop: Introduction to PyTorch

Léo Laugier



*This presentation is greatly inspired by the 2017 PyTorch Workshop (<https://github.com/mlberkeley/PyTorch-Workshop>) and the 2018 Introduction to Deep Learning (<https://github.com/mlberkeley/intro-dl-workshop>) from Machine Learning at Berkeley.

March 11, 2020