PyTorch
-Introduction, basics

Dr. Huiping Cao

Outline

- Why PyTorch?
- What is PyTorch?
- Installation
- Working with PyTorch's Dataset and DataLoader to build input pipelines and enable efficient model training
- Working with PyTorch to write optimized machine learning code
- Using the torch.nn module to implement common deep learning architectures conveniently
- Choosing activation functions for artificial NNs

Multilayer Perceptron (MLP) & DNN

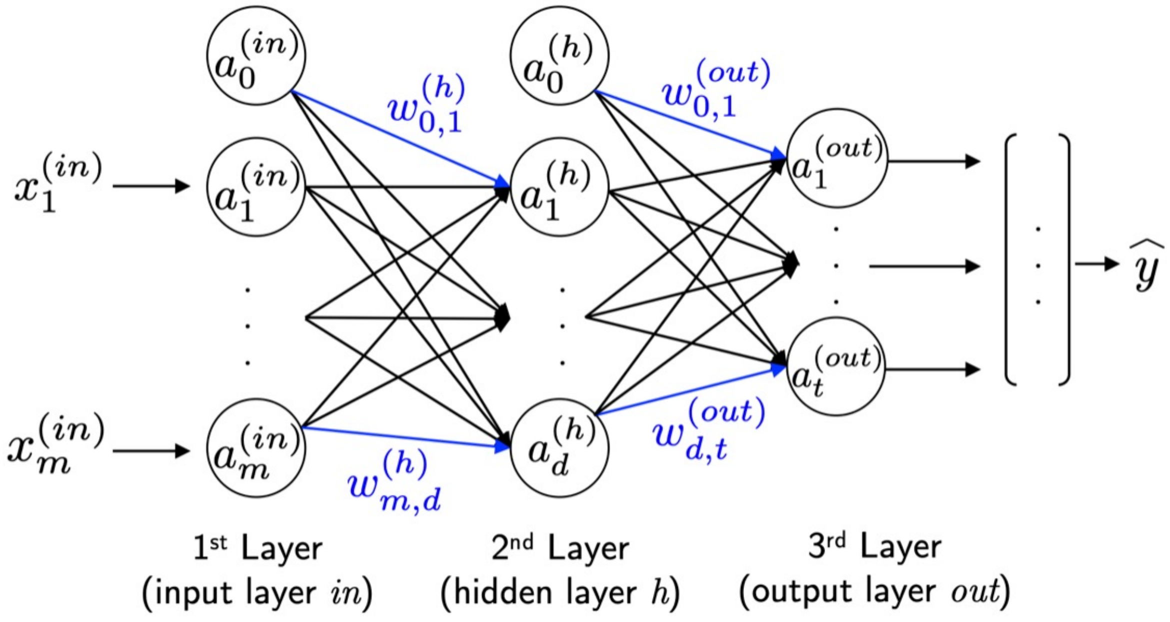
- A **Multilayer Perceptron (MLP)** consists of one **input** layer, some **hidden** layers, and one **output** layer. The units in the hidden layer are ***fully connected*** to the input layer, and the output layer is fully connected to the hidden layer.
- If an MLP has more than one hidden layer, it is called a **deep artificial neural network** (DNN).
  - Deep Neural Network (DNN) architectures are particularly well-suited for image and text analysis.

Deep learning

- **Deep learning**: a set of algorithms that are developed to train artificial neural networks with many layers most efficiently.

Why using PyTorch – many parameters to learn

- Training of a deep neural network needs to optimize at least ~10K of weight parameters.
- In a basic neural network with 100 hidden units for MNIST images (28*28). The **number of weight parameters** that we need to learn
  - (784*100 + 100)+(100*10+10) = 79510



Why using PyTorch - performance challenge

- PyTorch can speed up our machine learning tasks significantly.
- Scikit-learn libraries can parallelize model learning. However, this parallelization depends on the number of CPU cores, which is not a huge number.
  - Most advanced desktop hardware rarely comes with more than 8 or 16 such cores. s
- **Graphics Processing Units (GPUs)** have much more cores (e.g., several thousands) than CPUs (generally, at the level of 10s).
  - Writing code directly targeting GPUs is not user-friendly.

Why using PyTorch? – Using GPUs

- The challenge is that writing code to **target GPUs** is not as simple as executing Python code in our interpreter.
- There are special packages, such as **CUDA** and **OpenCL**, that allow us to target the GPU.
- The good news is that this is what PyTorch was developed for!

What is PyTorch?

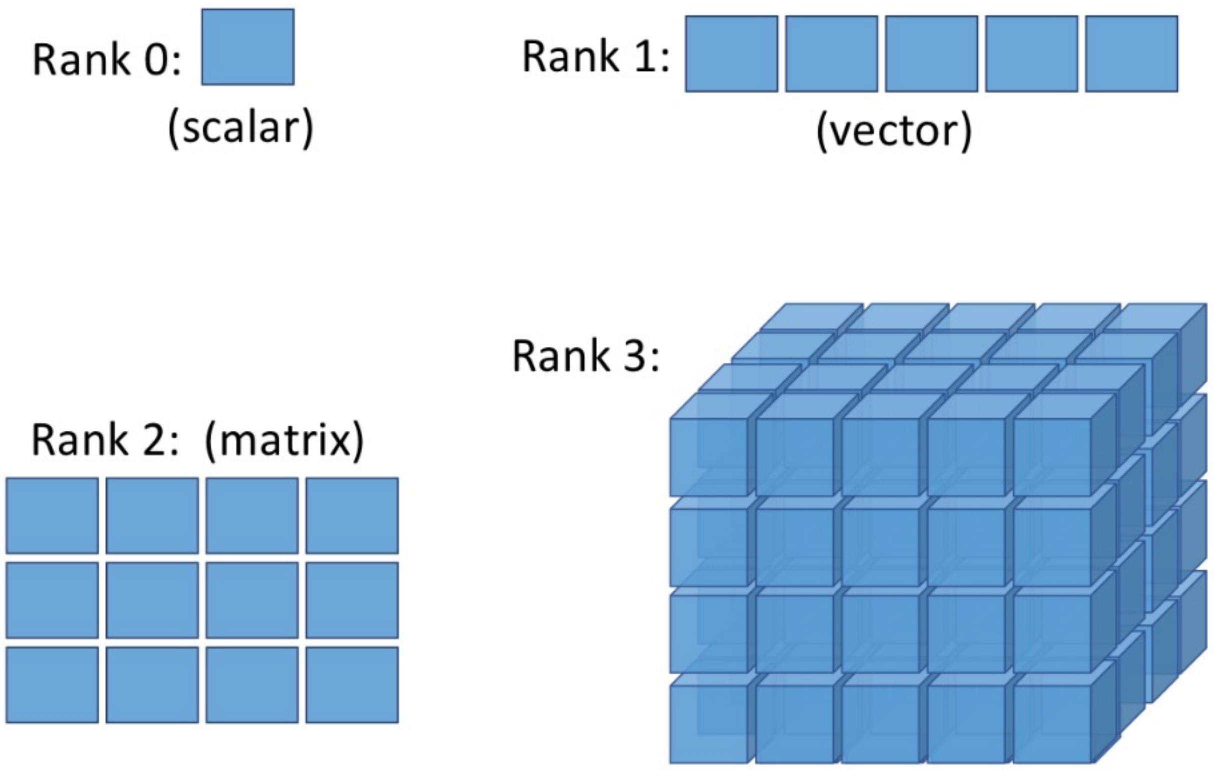
- PyTorch is one of the most popular deep learning libraries currently available, and it lets us implement **neural networks** (**NNs**) much more efficiently than any of our previous NumPy implementations.
- Scalable and multiplatform programming **interface** for implementing and running machine learning algorithms.
- PyTorch was primarily developed by the researchers and engineers from the **Facebook AI Research** (**FAIR**) lab
- PyTorch was initially released in **September 2016** and is free and open source under the modified **BSD license**.
- Many machine learning researchers and practitioners from academia and industry have adapted PyTorch to develop **deep learning solutions**, such as Tesla Autopilot, Uber’s Pyro, and Hugging Face’s Transformers (https://pytorch. org/ecosystem/).

What is PyTorch?

- PyTorch allows execution on CPUs, GPUs, and XLA devices such as TPUs
- PyTorch supports CUDA-enabled and ROCm GPUs officially. PyTorch's development is based on the Torch library (www.torch.ch).
- PyTorch is built around a **computation graph** composed of a set of **nodes**.
    - Each node represents an **operation** that may have zero or more **inputs** or **outputs.**

Tensors

- Tensors can be understood as a generalization of scalars, vectors, matrices



Tensors

- Tensors in PyTorch are similar to NumPy's arrays
  - Except that tensors are optimized for automatic differentiation and can run on GPUs.

PyTorch Topics

- Basics:
  - PyTorch’s programming model: creating and manipulating tensors.
  - How to load data and utilize the **torch.utils.data** module
  - Understand and use the **existing, ready-to-use datasets in the torch.utils.data.Dataset** submodule
- Advanced topics:
  - PyTorch neural network **torch.nn** module

Install PyTorch locally

- Follow steps in https://pytorch.org/get-started/locally/
- For example, for Mac OS
  - It is recommended that you use Python 3.7 or greater.
  - I already have anaconda installed.

```
$ python -V
Python 3.7.6

$ conda install pytorch torchvision -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done
…
```

Verification

- To ensure that PyTorch was installed correctly, we can verify the installation by running sample PyTorch code. Here we will construct a randomly initialized tensor.

```
$ python
Python 3.7.6 (default, Jan  8 2020, 13:42:34)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch  # this may take multiple seconds
>>> x = torch.rand(5, 3)
>>> print(x)
tensor([[0.3508, 0.2834, 0.4105],
        [0.6246, 0.4332, 0.3540],
        [0.0533, 0.8227, 0.9727],
        [0.7327, 0.7022, 0.3898],
        [0.2885, 0.0066, 0.8309]])
```

Tensors

- **Tensors** are a specialized data structure that are very similar to arrays and matrices.
- PyTorch uses **tensors** to encode **the inputs and outputs of a model**, as well as the **model's parameters**.
- Tensors are similar to **NumPy's ndarrays**, except that tensors can run on GPUs or other hardware accelerators.
- Topics:
  - Initializing a tensor
  - Tensor attributes
  - Move a tensor to GPU
  - Indexing and slicing

Tensor operations – Initializing a tensor

- Initializing a Tensor: Tensors can be initialized in various ways. Take a look at the following examples

```
# Tensors can be created directly from data.
# The data type is automatically inferred.
data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)
print(x_data)
```

Output:

```
tensor([[1, 2],
        [3, 4]])
```

Tensor operations – Initializing a tensor

```
# Tensors can be created from NumPy arrays
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
print(x_np)
```

Output:

```
tensor([[1, 2],
        [3, 4]])
```

Attributes of a tensor

- Tensor attributes describe their **shape**, **datatype**, and the **device** on which they are stored.

```
# Tensors can be created from NumPy arrays
tensor = torch.rand(3, 4)
print(f"Shape of tensor: {tensor.shape}")
print(f"Datatype of tensor: {tensor.dtype}")
print(f"Device tensor is stored on: {tensor.device}")
```

Output:

```
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

Operations on Tensors – Move to GPU

- There are more than 100 tensor operations, including arithmetic, linear algebra, matrix manipulation (transposing, indexing, slicing), sampling and more.
- Each of these operations can be run on the GPU (at typically higher speeds than on a CPU). If you're using **Colab**, allocate a GPU by going to **Runtime > Change runtime type > GPU**.
- By default, tensors are created on the CPU. We need to explicitly **move tensors to the GPU using .to method** (after checking for GPU availability).
  - Copying large tensors across devices can be expensive in terms of time and memory!

```
# We move our tensor to the GPU if available
tensor = torch.rand(3, 4)
if torch.cuda.is_available():
        tensor = tensor.to("cuda")
print(f"Device tensor is stored on: {tensor.device}")
```

Output:

```
Device tensor is stored on: cuda:0
```

Operations on Tensors – Indexing and slicing

- Standard numpy-like indexing and slicing:

```
tensor1 = torch.ones(4, 4)

print(tensor1)
print(f"First row: {tensor1[0]}")
print(f"First column: {tensor1[:, 0]}")
print(f"Last column: {tensor1[..., -1]}")
tensor1[:,1] = 0
print(tensor1)
```

Output:

```
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
First row: tensor([1., 1., 1., 1.])
First column: tensor([1., 1., 1., 1.])
Last column: tensor([1., 1., 1., 1.])
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

References

- Chapter 12: By Sebastian Raschka , Yuxi (Hayden) Liu , Vahid Mirjalili: Machine Learning with PyTorch and Scikit-Learn, Packt.
- https://pytorch.org/tutorials/
  - Most materials of this lecture are from https://pytorch.org/tutorials/beginner/basics/intro.html
- https://www.youtube.com/watch?v=c36lUUr864M&t=9613s (4.5 hours video)
- https://github.com/yunjey/pytorch-tutorial
- https://github.com/MorvanZhou/PyTorch-Tutorial