Introduction to Deep Learning (CS474)

Lecture 2

Rupam

Outline

- Pytorch Basics.
  - **Tensor**
    - Example
    - Numeric type
    - Operations
    - NumPy bridge

PyTorch Basics.

- PyTorch gives us a data type, the **Tensor**, to hold numbers, vectors, matrices, or arrays in general.
- In addition, it provides functions for operating on them.
- We can program with them incrementally and, if we want, interactively, just like we are used to from Python. If you know NumPy, this will be very familiar.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Tensor

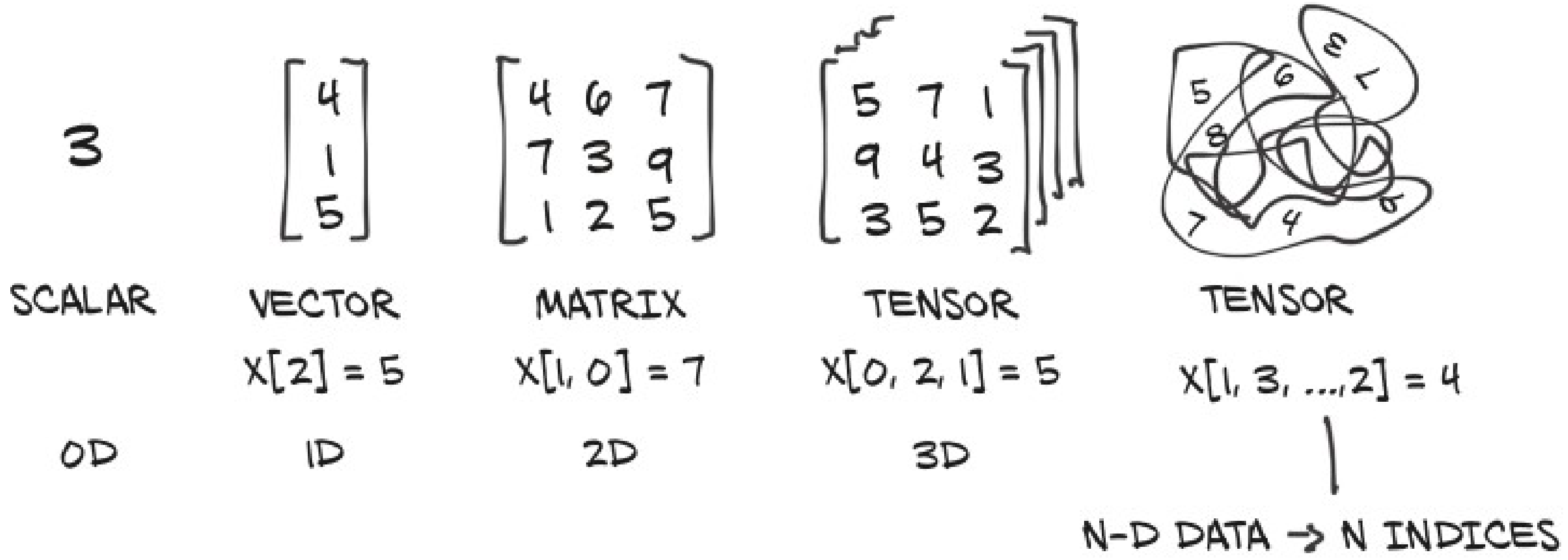

Figure 1: Tensors are the building blocks for representing data in PyTorch

Image credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Tensor

- PyTorch is not the only library that deals with multidimensional arrays.
- Compared to NumPy arrays, PyTorch tensors have a few superpowers.
- PyTorch tensors have the ability to perform very fast operations on graphical processing units (GPUs), distribute operations on multiple devices or machines, and keep track of the graph of computations that created them.
- These are all important features when implementing a modern deep learning library.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Tensor: Example

- We would like to use to represent a geometrical object.
- Let us consider a 2D triangle with vertices at coordinates (4, 1), (5, 3), and (2, 1).
- The example is not particularly pertinent to deep learning, but it's easy to follow.
- Instead of having coordinates as numbers in a Python list, as we did earlier, we can use a one-dimensional tensor by storing Xs in the even indices and Ys in the odd indices.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Tensor: Example (contd.)

TensorBasics.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Rename
Rename notebook

+ Code + Text

```
[1] import torch

    points=torch.zeros(6)
    print(points)
```

```
tensor([0., 0., 0., 0., 0., 0.])
```

```
[4] points[0] = 4.0
    points[1] = 1.0
    points[2] = 5.0
    points[3] = 3.0
    points[4] = 2.0
    points[5] = 1.0

    float(points[0]), float(points[1])
```

```
(4.0, 1.0)
```

```
#use of 2D tensor

pointsNew = torch.tensor([[4.1, 1.1], [5.1, 3.1], [2.1, 1.1]])
pointsNew
```

```
tensor([[4.1000, 1.1000],
        [5.1000, 3.1000],
        [2.1000, 1.1000]])
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Tensor: Example (contd.)

```
[10] # We can ask the tensor about its shape!
     # This informs us about the size of the tensor along each dimension

     pointsNew.shape
```

```
torch.Size([3, 2])
```

```
[12] # Now we can access an individual element in the tensor using two indices:
     pointsNew[0, 1]
     #Above line help us to get the Y-coordinate of the zeroth point
```

```
tensor(1.1000)
```

```
# We can also access the first element in the tensor.
pointsNew[0]
```

```
tensor([4.1000, 1.1000])
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Tensor: Numeric types

- So far, we have covered the basics of how tensors work, but we have not yet touched on **what kinds of numeric types** we can **store in** a **Tensor**.
- The **dtype** argument to tensor constructors (that is, functions like tensor , zeros , and ones) specifies the numerical data (d) type that will be contained in the tensor.
- possible values for the `dtype` argument:
  - `torch.float32` or `torch.float`: 32-bit floating-point
  - `torch.float64` or `torch.double`: 64-bit, double-precision floating-point
  - `torch.float16` or `torch.half`: 16-bit, half-precision floating-point
  - `torch.int8`: signed 8-bit integers
  - `torch.uint8`: unsigned 8-bit integers
  - `torch.int16` or `torch.short`: signed 16-bit integers
  - `torch.int32` or `torch.int`: signed 32-bit integers
  - `torch.int64` or `torch.long`: signed 64-bit integers
  - `torch.bool`: Boolean

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Tensor: Numeric types

- Computations happening in neural networks are typically executed with 32-bit floating-point precision.

- Higher precision, like 64-bit, will not buy improvements in the accuracy of a model and will require more memory and computing time.
-

```python
import torch

#In order to allocate a tensor of the right numeric type,
#we can specify the proper dtype as an argument to the constructor.

double_points=torch.ones(10, 2, dtype=torch.double)
short_points=torch.tensor([[1, 2], [22, 33]], dtype=torch.short)

print(double_points)
print(short_points)
```

```
tensor([[1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.]], dtype=torch.float64)
tensor([[ 1,  2],
        [22, 33]], dtype=torch.int16)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Tensor: Operations (Indexing tensors)

- Reminder!

All elements in the list

```
# In[53]:
some_list = list(range(6))
some_list[:]
some_list[1:4]
some_list[1:]
some_list[:4]
some_list[:-1]
some_list[1:4:2]
```

From element 1 inclusive to element 4 exclusive

From element 1 inclusive to the end of the list

From the start of the list to element 4 exclusive

From the start of the list to one before the last element

From element 1 inclusive to element 4 exclusive, in steps of 2

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Tensor: Operations (Indexing tensors)

- Hope you remember our previous *Notebook* example with **pointsNew**!

```
[14] # All rows after the first; implicitly all columns.

     pointsNew[1:]

     tensor([[5.1000, 3.1000],
             [2.1000, 1.1000]])
```

```
#All rows after the first; all columns
pointsNew[1:, :]

tensor([[5.1000, 3.1000],
        [2.1000, 1.1000]])
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Tensor: Operations (transpose)

- You can create a *Notebook* in Google colab for experimenting with the tensor API.
- Vast majority of operations on and between tensors are available in the **torch module**.

```
# Checking transpose.

a=torch.ones(3, 2)
a_transpose=torch.transpose(a, 0, 1)

print(a.shape)
print(a_transpose.shape)
```

```
torch.Size([3, 2])
torch.Size([2, 3])
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Tensor: Operations (addition)

```
import torch

# Initialize

x = torch.ones(2, 3)
y = torch.rand(2, 3)    # Initialize with random values

# Operations

z1 = x + y
z2 = torch.add(x, y)

print(z2)
print(z1)
```

```
tensor([[1.2826, 1.3097, 1.0306],
        [1.6098, 1.7936, 1.0860]])
tensor([[1.2826, 1.3097, 1.0306],
        [1.6098, 1.7936, 1.0860]])
```

Tensor: Operations (In-place operations)

- Certain operations exist only as methods of the Tensor object.

- They are recognizable from a trailing underscore in their name, like zero_ , which indicates that the method operates in place by modifying the input instead of creating a new output tensor and returning it.

- For instance, the zero_ method zeros out all the elements of the input.

-

```python
# checking In-place operation.
new = torch.ones(3, 2)
new.zero_()
print(new)
```

```
tensor([[0., 0.],
        [0., 0.],
        [0., 0.]])
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Tensor: NumPy bridge

- Converting a Torch Tensor to a NumPy array and vice versa is a breeze.
- The Torch Tensor and NumPy array will share their underlying memory locations (if the Torch Tensor is on CPU), and changing one will change the other.

  - Converting a Torch Tensor to a NumPy Array
  - Converting NumPy Array to Torch Tensor

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Tensor: NumPy bridge

- Converting a Torch Tensor to a NumPy Array

```
[2] import torch
    import numpy
    # checking bridge!
    a = torch.ones(5)
    print(a)
    b = a.numpy()
    print(b)
```

```
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
```

```
#Now we will see how the numpy array changed in value.
a.add_(1)
print(a)
print(b)
```

```
tensor([2., 2., 2., 2., 2.])
[2. 2. 2. 2. 2.]
```

Slide credit: https://pytorch.org

Rupam

Tensor: NumPy bridge

- Converting NumPy Array to Torch Tensor

```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)
```

Out:

```
[2. 2. 2. 2. 2.]
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
```

Slide credit: https://pytorch.org

References

- All the contents present in the slides are taken from various online resources. Due credit is given in the respective slides.These slides are used for *academic* purposes only.