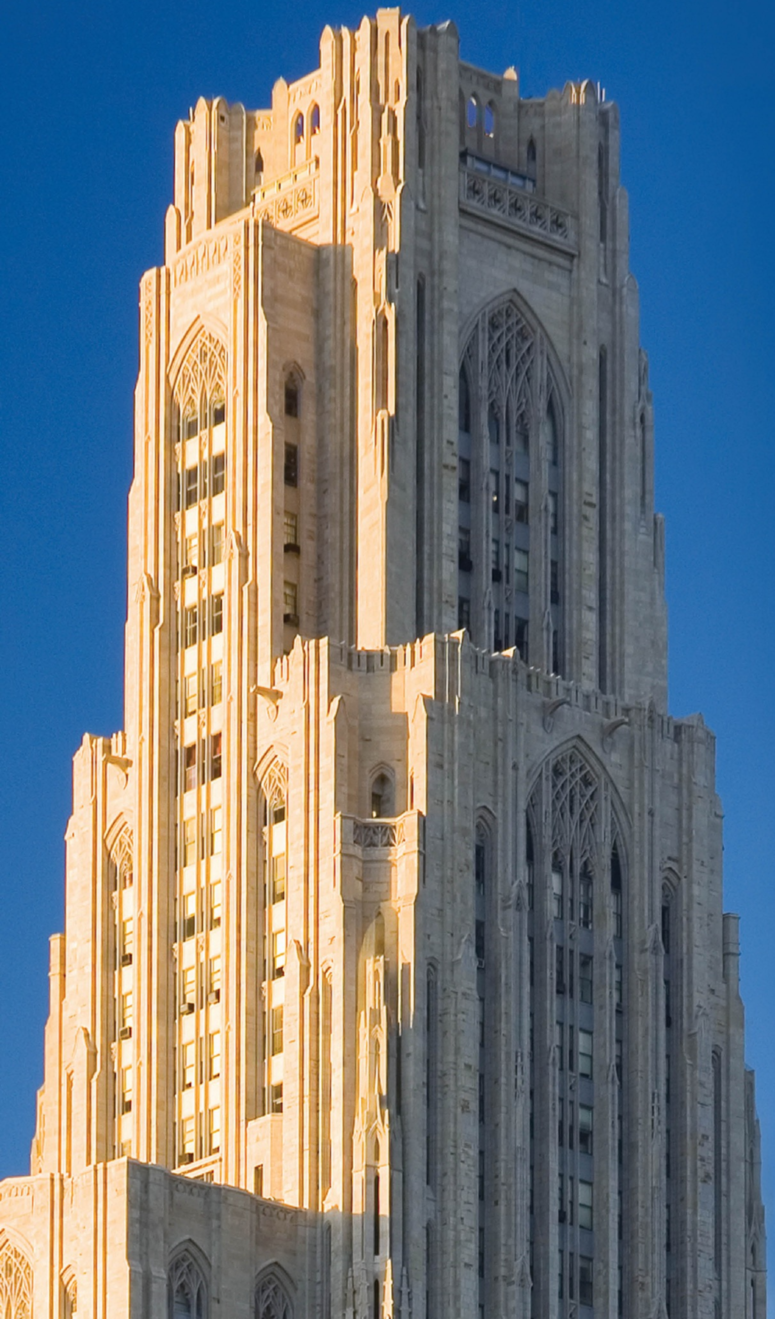


Pitt HexAI Mini Summer Camp 2023

Introduction to PyTorch

Instructors:

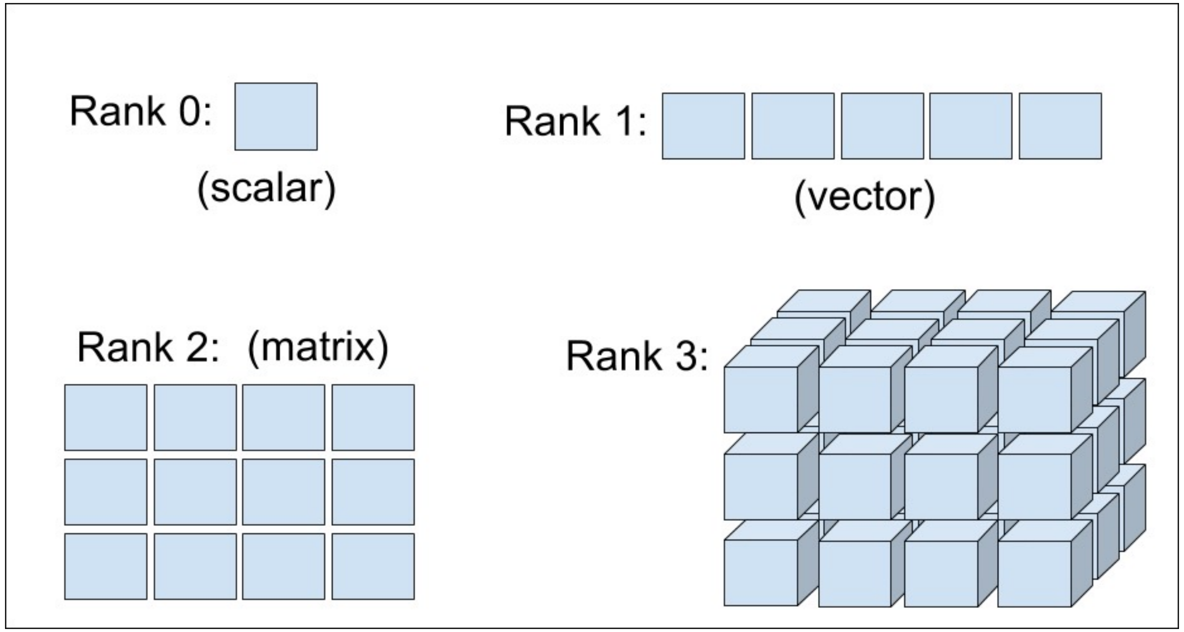
- Ahmad P. Tafti, PhD, FAMIA
- Nickolas Littlefield (PhD Student at Pitt)
- Kyle Buettner (PhD Student at Pitt)

PyTorch: What and why?

- Developed by Facebook’s AI Research Group
- An easy-to-use, yet powerful Python deep learning library used for applications in computer vision and natural language processing.
- Highly flexible, efficient and scalable, designed to minimize the number of computations required, and be compatible with different varieties of hardware architectures.

Tensors: The Data Structure of Deep Learning

- Tensors are fundamental data structures that efficiently perform mathematical operations on large sets of data
- Can be **n-dimensional**
- Three key components:
  - **Shape**: the size of the tensor
  - **Data type**: type of data stored in the tensor
  - **Device:** the device where the tensor is stored (CPU or GPU)



Datasets and DataLoader

- Provides functionality for loading training and test data efficiently
- **Dataset:** Provides a uniform interface to access training/testing data
  - __getitem__: returns the i-th data record in the dataset
  - __len__: returns the size of the dataset
  - Usually defined by us, unless we are using a predefined dataset
- **Data Loader:** Efficiently loads and stacks data from the dataset into batches. We define the parameters
  - batch_size: Number of samples
  - shuffle: Shuffle the dataset into random order

nn Module: Neural Networks

- Provides functionality for creating neural networks
- Contains collections of different layers, activation functions, and loss functions
- We define both the structure of the network and the forward() function to define the way the computation is done

```python
class SimpleClassifier(nn.Module):

    def __init__(self, num_inputs, num_hidden, num_outputs):
        super().__init__()
        # Initialize the modules we need to build the network
        self.linear1 = nn.Linear(num_inputs, num_hidden)
        self.act_fn = nn.Tanh()
        self.linear2 = nn.Linear(num_hidden, num_outputs)

    def forward(self, x):
        # Perform the calculation of the model to determine the prediction
        x = self.linear1(x)
        x = self.act_fn(x)
        x = self.linear2(x)
        return x
```

```python
model = SimpleClassifier(num_inputs=2, num_hidden=4, num_outputs=1)
# Printing a module shows all its submodules
print(model)
```

```
SimpleClassifier(
  (linear1): Linear(in_features=2, out_features=4, bias=True)
  (act_fn): Tanh()
  (linear2): Linear(in_features=4, out_features=1, bias=True)
)
```

torchvision

- Includes a variety of popular pretrained neural networks architectures for computer vision tasks such as classification, segmentation, and object detection

Object Detection

The following object detection models are available, with or without pre-trained weights:

- Faster R-CNN
- FCOS
- RetinaNet
- SSD
- SSDlite

Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

Training a Model

- After creating our model (custom or torchvision) we train and optimize it
- Need to define a loss function and an optimizer:
  - Common Loss Functions: BCELoss/BCELossWithLogits (Binary output), CrossEntropyLoss (Multiple classes)
  - Common Optimizers: SGD (Stochastic Gradient Descent), Adam
- Steps:
  1. Get training batch from the data loader
  2. Obtain predictions from the model for the batch
  3. Calculate the loss between the predictions and the actual labels
  4. Backpropagation
  5. Update the model parameters
  6. Evaluate on validation (if available)

GPUs

- Training neural networks require a lot of memory
- Graphics Processing Units (GPUs) can speed up computations and provide the memory needed
- PyTorch provides ways to transfer data from the CPU to a GPU.
- Steps:
  - Determine whether a GPU is available and get the device name: torch.cuda.is_available()
  - Transfer model to GPU
  - During training transfer the data to the GPU

```
# Set device for training
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
# Move model to device
model = model.to(device)
```

Testing a Model

- After training, we evaluate the model on unseen data
- Steps:
  1. Get a batch of data
  2. Get the predictions from the model
  3. Calculate the evaluation metric (accuracy, F1-Score, IoU)

Thank you!

Ahmad P. Tafti: tafti.ahmad@pitt.edu
Nickolas Littlefield: ngl18@pitt.edu
Kyle Buettner: buettnerk@pitt.edu