Module 35 - Introduction to Tensorflow and Pytorch

Course by: *Team DSS (Amanda Ma, Kevin Miao, Sandeep Sainath, Youngli Hong)*

Pre-Requisites

Before we get started here, it is important to establish the strict prerequisites of this module. Neural networks are on an extreme learning curve and in order for you to get the most out of this course, it is beyond essential that you have taken the following or equivalent courses:

1. EE16A
2. EE16B (Concurrently)
3. CS61A & B or equivalent programming experience
4. Module 21 - Backpropagation
5. Module 23 - Hyperparameter tuning
6. Module 34 - CNNs
7. Module 35 - Transfer Learning

Module Objectives

The goals of this class are that after finishing the module, you will be able to:

1. get an intuition of the differences between the two libraries.
2. transition between programming, math and theory, specifically, the different optimizations and gradients.
3. load and augment data.
4. independently use any of the libraries to implement neural models of choice.
5. be able to effectively debug and read through documentation.
6. get some sense of residual nets are implemented on a basic level.

Set-Up

We will accomplish this through 5 level assignments:

1. **A/B: Introductory PyTorch and TensorFlow** (1-3 hr total, each)
2. **A/B: Having fun with Toy Models in PyTorch and TF** (2-4 hr, each)
3. **Capstone Project: Classifying Fractures** (5-7 hr total)

Level 1 assignments are hand-holding, light exercises to make you comfortable with the material and the syntax used in a certain language. We are aware that everyone is used to different libraries and think it's important to understand why certain syntactical decisions have been made by the respective PyTorch and TensorFlow developers.

Level 2 assignments are a little bit more challenging. They are showing you around through one cycle of the **data science lifecycle**. One round of loading, cleaning, classifying and evaluating. These notebooks will mainly focus on allowing you to become independent through forcing you to debug, read documentation and to connect what you have learned before this class together with the modules and notes here.

History of Deep Learning and Libraries

Both breakthroughs in computing technologies and increase in large datasets contributed to advances in machine and deep learning. The number of algorithms and problem solving strategies improved over time, along with their implementations through library frameworks.

Current Frameworks

Many software tools have been developed for machine and deep learning applications over the last quarter decade. No single library is suitable for every AI problem, so often a project requires the use of a combination of frameworks. Popular frameworks for deep learning with GPU include Tensorflow, CNTK, Caffe, Torch/Pytorch, and Theano. Keras is considered a wrapper library built on top of Tensorflow. Many of these open source libraries are available on GitHub.

Tensorflow

Tensorflow, first released by Google Brain in 2015, is an open source library for implementing, deploying, and maintaining deep learning algorithms for various commercial applications -- translation, computer vision, search, speech recognition, etc. In 2019, Google migrated from Tensorflow version 1.x to version 2.x. In the second version, Tensorflow uses the simpler Keras API for building neural networks.

In Tensorflow, **tensors** or multidimensional arrays are the standard way to represent data. A dataflow/computation **graph** defines a series of computations that interact with one another, where **nodes** represent mathematical operations and **edges** represent data/tensors flowing through or around the system. This design interface is conducive to machine and deep learning modeling.

Tensorflow provides these functionalities through the Python programming language (and many other languages!) Python is convenient to work with because it meshes everything together through high-level abstractions. However, C++ is really what's under the hood for mathematical operations.

One development convenience that Tensorflow offers is the **eager execution** mode, which allows evaluation and modification of individual graph operations separately, rather than a single evaluation of an entire graph.

Tensor Objects:

- have 3 main properties: a unique name, shape, and dtype (data type)

Module 35 - Introduction to Tensorflow and Pytorch

- can have one type of data at a time

Scalar: a rank-0 tensor that contains a single value
Vector: a rank-1 tensor that is a list of values
Matrix: a rank-2 tensor with two “axes”

Constant: a tensor that cannot be modified
Variable: a tensor that can be modified by various graph operations

Key Terms
Tensors
Graph (nodes, edges)
Eager Execution
Scalar, Vector, Matrix
Constant, Variable

⇒ *Complete Assignment 1B*

Module 35 - Introduction to Tensorflow and Pytorch

PyTorch

PyTorch
PyTorch was created in 2016 mainly by Facebook AI in an effort to create a low-level yet extremely intuitive, pythonian solution to Deep Learning applications. Theano use had been declining significantly and TensorFlow use had increased, but Facebook AI was not satisfied with the extremely declarative nature of it. PyTorch was born. PyTorch is designed to be more pythonian and dynamic. It is also developed with the objective to make it more up-to-date with the standard of data parallelism. Even though TensorFlow is used for high-scale employment, yet, many researchers, Tesla's autopilot and UBER's pyro swear by PyTorch.

Tensors and NumPy
PyTorch tensors are extremely similar to NumPy, more than TensorFlow's tensors. They have the same functionality, they have a similar structure and similar functions are used to initialize them. You can perform basic arithmetic operations on it. Mutative indexing is also similar between the two. Some differences are that Tensor contains more attributes that support gradients. Additionally, Tensors have some other optimizations that make it faster than NumPy arrays. You will see more in assignment 1b. It is beyond the scope to think about why this is the case.

Gradients and Optimizers
As mentioned in the previous paragraph, Tensors support (partial) derivatives! How? Well through initialization of the argument `required_grad=True`. Basically, what happens is that tensors will keep track of the gradients. If you perform any calculations with it, the output tensors will keep track of the arithmetic functions you have placed in there. This allows for easy derivative recovery. `.backward()` calculates the derivative with respect to what you are differentiating it with. You can recover the gradient through `tensor.grad`. In order to remove the gradient, because, for instance, you want to access the data you can simply call `tensor.detach()`. However, if you are training a whole network, you don't want to deal with individual tensors. No worries, there is a `torch.optim`, an optimizer, pre-defined for you. The optimizer optimizes the backprop algorithm with different smart tricks. The simple thing you need to do is to initialize the optimizer with a tensor of coefficients, learning rates and a momentum factor. A momentum factor is the degree of how much you want to use previous gradients in defining what direction to go for minimizing the loss function. Define a loss function and call backward on it, and you are done with one iteration of backpropagation. Easy as that.

Functions
This paragraph is not in the slides. However, we just wanted to let you know that many of the functions and layers that you will be using are already defined for you. For more information, please go through the documentation for `torch.nn`: https://pytorch.org/docs/stable/nn.html.

Module 35 - Introduction to Tensorflow and Pytorch

Building Neural Networks

Since you have done previous modules where you are required to build CNNs, you are an expert now in ANN! Building Neural Networks in PyTorch is really intuitive within the OOP paradigm. Neural networks are wrapped in class wrappers. Each of the neural networks inherits from `nn.Module`. They also need to include an initiator, a `forward` function and perhaps some auxiliary functions. The initializer just needs to initialize the super class and it also initializes the layers that you will be using. The `forward` function basically is equivalent to `__call__`. When you call the net on an input, then you want to return what the `forward` function returns. Simply, one forward pass. Auxiliary functions that are useful are those that flatten the number of features that way you don't have to hard code anything.

Data Utilities

Datasets and data loaders like neural networks are defined within classes. Datasets hold the data, data loaders are used for training the actual model. Datasets contain an initializer that has the `data`, `labels` and `transforms`. Transforms here is a transformation functions that you will use to perform data augmentation or any conversion (toTensors() for instance). Other functions that you need are `len` and also `__get__`. Get takes the index number and the dataset returns a tuple, dictionary (or what is the most suitable for that situation) located at index x. Data Loaders can be constructed from datasets and are a simple one line using torch.data.dataloader, where you only need to define the number of works and batch_size.

Data Augmentation

Data augmentation is extremely important as it can help us prevent overfitting but it can also create 9 pictures out of one picture, basically increasing our data size. Transforms are many functions that are predefined for you and you are encouraged to read about them in their documentation.

CUDA/GPUs

CUDA is a parallel computing platform that allows the use of GPUs for general processing. To use a GPU in Pytorch you need to send it back and forth. Basically, you need to define a device first, then you send your data and model to the device through `model/tensor.to(device)`. If you want to get it back, then you can just append `.data`

ResNets

Residual Nets are a smart augmentation of CNN where we have a residual block as shown below. This residual block basically makes it such that your input also has an effect on layers further than one away. This development came together when neural networks became deeper and deeper. For the purpose of the class you don't need to know how they work exactly, but also

it is still being studied heavily. Nonetheless, they come in different flavors and in PyTorch you don't have to worry about writing one yourself. You can just download one.
⇒ *Complete assignment 2b*

Project Overview

You will be using Stanford MURA data for the classification of bone fractures from image. This project will contain little to no starter code and will be conducted in the library of choice. It is intimidating at first, however, if you completed the first 4 assignments successfully and on your own, you should be more than prepared to tackle this project. Follow the structure of the previous assignments and even copy/paste some pieces of code. Additionally, loading **pickle** data has been explained in a tutorial link in the assignment.

⇒ *Complete the project*

Ethics

Now you have completed a bone fracture classification model, it is time we start thinking about the ethical implications about this. In general, we often forget to pay attention to the ethical sides which can have disastrous consequences and ramification for people's lives. One reason for that being that we all are biased. The moral machine test shows us that every person is influenced by the culture they grow up in. People in China value the lives of elderly more than people in western countries, for instance. These biases shape the technologies we create as well. But also, many more unconscious biases are embedded into these systems as well. For example, in the COMPAS recidivism algorithm, African American criminals are predicted to be more recidivist than their white peers who have exactly the same background where only their race differs. This has a huge impact on those people's lives. Another instance is where women are more likely to see job listings that are associated with traditionally feminine jobs, perpetuating the stereotype. Machine learning, again, is not a bad or good tool. It is essential that you as a machine learning scientist understand the limitations and extent of the consequences. Machine bias, the bias embedded in computer systems, are more far reaching than any human being due to its easy proliferation and large-scale employment. We will live it off as an exercise for you to start thinking about how we can make more responsible algorithms. For your project, you can receive bonus points if you implement 'meta analysis', 'transparency' or evaluate your model with multiple different 'scores'.

⇒ *Take the quiz*

Conclusion

You should now feel extremely familiar with building neural networks and using deep learning frameworks in your endeavors to conquer the world. To conclude, as you have seen, TensorFlow

Module 35 - Introduction to Tensorflow and Pytorch

and PyTorch are extremely diverse libraries and frameworks that have a lot in common but are also optimizing in different ways. Hopefully, it will not scare you away from trying out different libraries when working on a specific project. TensorFlow, as you have seen, is way more applicable for use in large-scale employment applications, whereas PyTorch is extremely nice to have if you need to set-up something quick. Nonetheless, both frameworks are used interchangeably in industry. Many Data Scientists use TF as a quick go-to and PyTorch is embedded in Tesla Autopilot for example. Additionally, we hope that you enjoyed this module but also keep thinking about the ethical implications of implementing Machine Learning.

Congratulations, you have now achieved the **learning objectives** stated in the beginning.