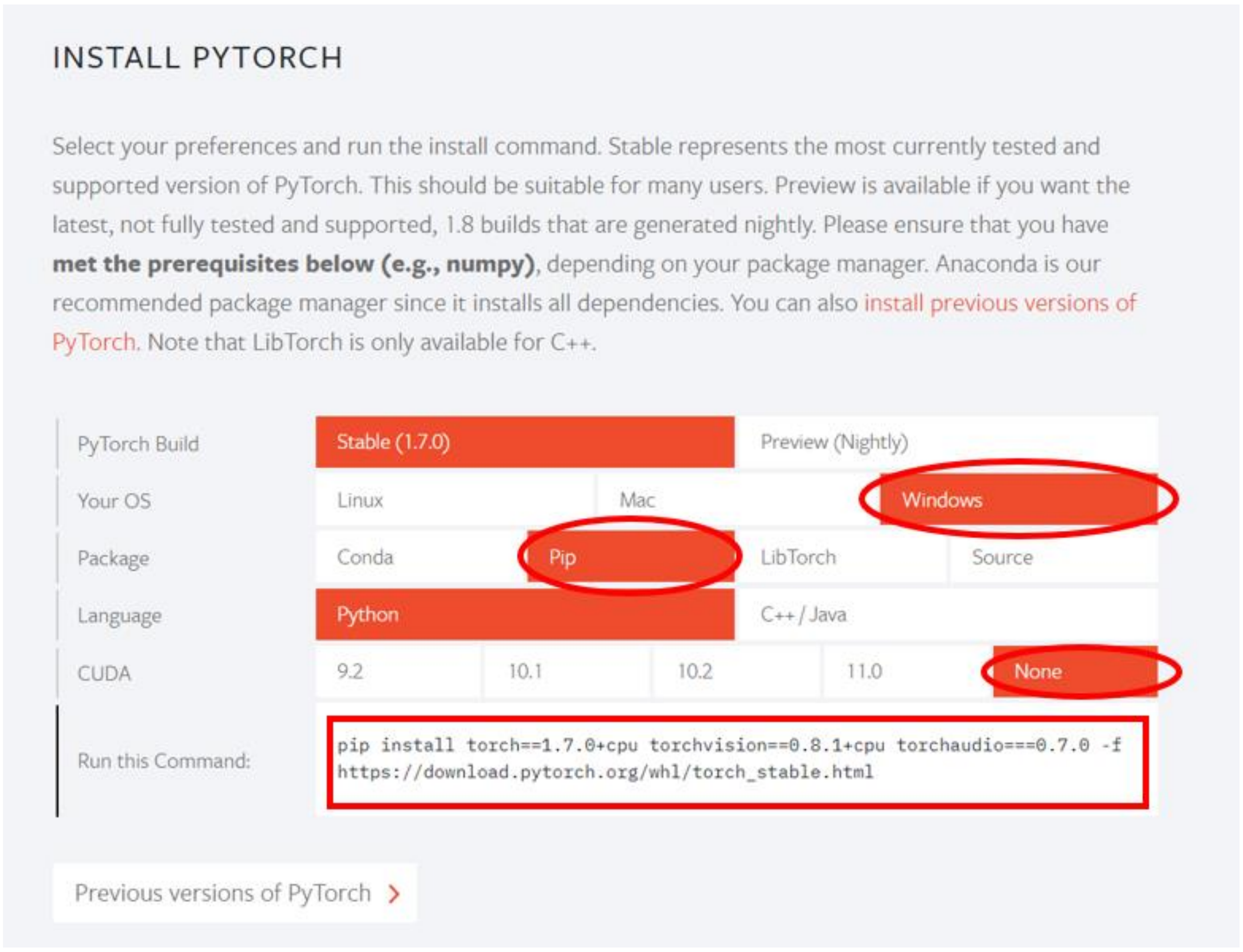
PyTorch

PyTorch is a machine learning library used to get the datasets prepared for the neural net algorithm.

Install

PyTorch has a lot of configurations, the version you need depends on the machine and the environment. The best way to install PyTorch for your machine is by running the command generated by their website after you specify some criteria.

Go to the following website for a complete guide of installation
https://pytorch.org/



INSTALL PYTORCH

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, 1.8 builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

PyTorch Build	[✓ Stable (1.7.0)]	Preview (Nightly)			
Your OS	Linux	Mac	[✓ Windows]		
Package	Conda	[✓ Pip]	LibTorch	Source	
Language	[✓ Python]	C++ / Java			
CUDA	9.2	10.1	10.2	11.0	[✓ None]

Run this Command:

```
pip install torch==1.7.0+cpu torchvision==0.8.1+cpu torchaudio===0.7.0 -f
https://download.pytorch.org/whl/torch_stable.html
```

Previous versions of PyTorch >

Figure 1: An example of generating a PyTorch install command for Python on windows operating system

Warning: If using with Python, use Python version 3.7 or lower. PyTorch uses a component of python called "Wheels" which has deprecated at python 3.8. Therefore, any version later than Python 3.7 will generate an error while installing.

Warning: A project level virtual environment is required to install or run PyTorch.

Note: A container with dependencies installed (singularity) is provided in Compute Canada. Please see the Compute Canada section for instruction on usage.

PyTorch Library Important Files

Dataset

Path: <project folder>/venv/lib/torchvision/datasets/mnist.py
Description: Dataset is a structure with which your data will be loaded. The file above is the default dataset definition by PyTorch. However, you can write your own dataset and use it for a more customized dataset for your purpose.

The following datasets are used for Triumf's CNN framework:

1) https://github.com/WatChMaL/WatChMaL/blob/master/watchmal/dataset/cnn_mpmt/cnn_mpmt_dataset.py
2) https://github.com/WatChMaL/WatChMaL/blob/master/watchmal/dataset/h5_dataset.py

The following datasets are used for Triumf's point net framework:

1) https://github.com/WatChMaL/WatChMaL/blob/master/watchmal/dataset/cnn_mpmt/cnn_mpmt_dataset.py
2) https://github.com/WatChMaL/WatChMaL/blob/master/watchmal/dataset/h5_dataset.py

The main datasets are in the first link, but it uses another base dataset class, which can be imported from the second link.

Sampler:
Path: <project folder>/venv/lib/torch/utils/data/sampler.py
Description: The purpose of a sampler is to determine how the batches should be formed. The sampler.py comes with a few already made samplers. But it is important to note that you can write your own sampler for more custom purposes. The DataLoader takes in a sampler as one of its parameters.

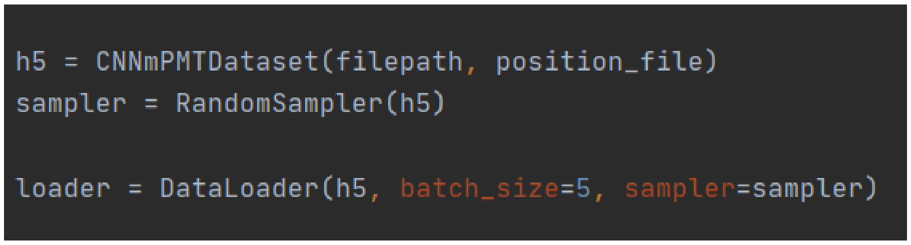
DataLoader
Path: <project folder>/venv/lib/torch/utils/data/dataloader.py
Description: A DataLoader combines Dataset and Sampler to provide an iterable over the dataset. Which the machine learning algorithm can use to fetch formatted data for training.

Hint: More information can be found about Sampler and DataLoader in the code files sampler.py and dataloader.py

DataLoader Usage

Below are the steps you can take to create a DataLoader:

Step 1: Pick a PyTorch Dataset component
Step 2: Generate an instance of the Dataset from the data file
Step 3: Write a sampler or pick one of the default ones
Step 4: Create an instance of the DataLoader (pass in the Dataset and the sampler as parameter)



```
h5 = CNNmPMTDataset(filepath, position_file)
sampler = RandomSampler(h5)

loader = DataLoader(h5, batch_size=5, sampler=sampler)
```

Figure 2: Instantiating a DataLoader

The following file shows how Triumf instantiates and uses the DataLoader: https://github.com/WatChMaL/WatChMaL/blob/master/watchmal/engine/engine_classifier.py

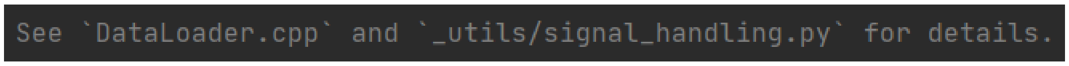
Dynamic Sampling Progress

WatChMal team already had Python code that composites two events. Our task was to find a way to dynamically composite two events for the Convolutional Neural Network.

Problem: Our main challenge was to sample 2 different events in the same sampler.

Approach 1: Our first approach to this problem was to write a custom sampler that returns two different events. This technique did not work in our favor. As it turns out samplers do not return events. It only gives the events indices based on the algorithm provided. And then the events are fetched by the index and loaded into the DataLoader.

Approach 2: By now it was clear that we had to write a custom DataLoader that calls the sampler twice to get 2 different events. Before we could write a DataLoader, we had to understand how a DataLoader works. While tracing the methods back, trying to understand the source code of the DataLoader, we arrived at a C++ file. dataloader.cpp. Given that we only had 3 more weeks left of the project and no prior knowledge of C++, it was no longer feasible for us to learn a new language and then write the DataLoader.


```
See `DataLoader.cpp` and `_utils/signal_handling.py` for details.
```
Figure 3: Code Snippet from dataloader.py

Steps if we had time

It is possible to write a new DataLoader. So, we would learn C++, understand the source code, and then modify it to call the sampler twice. And then we would have to add the composition algorithm. The DataLoader can then pick two events from the two samplers and then composite them to return it dynamically.

After that we would add a searching/filtering algorithm in our custom sampler to allow our users to pick the events based on some matching criteria.