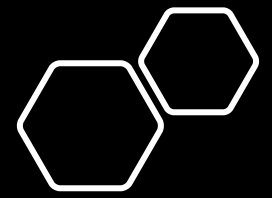
AIMS | African Institute for Mathematical Sciences RWANDA

AMMI Review sessions

Deep Learning Labs (2)

Data Loading in Pytorch



10 Feb 2020

torch.utils.data.Dataset

- Abstract class representing a dataset. Your custom dataset should inherit Dataset and override the following methods

(1) torch.utils.data.Dataset

- Abstract class representing a dataset. Your custom dataset should inherit Dataset and override the following methods

override the following methods:

- __len__ so that len(dataset) returns the size of the dataset.
- __getitem__ to support the indexing such that dataset[i] can be used to get ii\ th sample

(1) torch.utils.data.Dataset

- Abstract class representing a dataset. Your custom dataset should inherit Dataset and override the following methods

override the following methods:

- __len__ so that len(dataset) returns the size of the dataset.
- __getitem__ to support the indexing such that dataset[i] can be used to get ii\ th sample

- Usually we read the data in __init__ but leave the reading of the images to ___getitem___
- This is memory efficient because all the images are not stored in the memory at once but read as required

(1) torch.utils.data.Dataset

- In __getitem___ you choose the way you want to form your training sample, i.e it can be of any convenient shape or data structure

- Our dataset will take an optional argument transform so that any required processing can be applied on the sample

- datasetObject[i], will call the __getitem__(i) method on that object

(2) torch.utils.data.TensorDataset

- torch.utils.data.TensorDataset(**tensors*)
- Dataset wrapping tensors.
- Each sample will be retrieved by indexing tensors along the first dimension.

e.g.

train_dataset = TensorDataset(torch.from_numpy(X_train).float(), torch.from_numpy(y_train).long())

(3) torchvision.datasets

- Return Dataset object for popular vision datasets

- torchvision.datasets.ImageFolder(root) A generic data loader where the images are arranged in this way:

root/dog/xxx.png
root/dog/xxy.png
root/dog/xxz.png

root/cat/123.png
root/cat/nsdf3.png
root/cat/asd932_.png

Transofrms

- In many cases we need to apply transformations to our training set e.g normalizations.

- transforms.Compose([]) allows composing series of transforms from these custom transforms.

- torchvision package provides some common transforms as well
  such as transforms.Normalize((mean), (std)), and transforms.ToTensor()
  They basically do the same function as custom transforms but they **expect your ___getitem__()**
  to return only tensor, i.e won't work for other custom formats

torch.utils.data.DataLoader

- Combines a dataset and a sampler, and provides an iterable over the given dataset, supports:

- Batching the data
- Shuffling the data
- Load the data in parallel using multiprocessing workers.

- dataloader = DataLoader(transformed_dataset, batch_size=4, shuffle=True, num_workers=4)

Data Loading Order and **Sampler**

- torch.utils.data.Sampler classes are used to specify the sequence of indices/keys used in data loading.
- A sequential or shuffled sampler will be automatically constructed based on the shuffle argument to a DataLoader
- Users may use the sampler argument to specify a custom Sampler object that at each time yields the next index/key to fetch.

Single- and Multi-process Data Loading

- PyTorch provides an easy switch to perform multi-process data loading by simply setting the argument num_workers to a positive integer.
- Having more workers will increase the memory and IO usage and that's the most serious overhead.
- you would experiment and launch approximately as many as are needed to saturate the training.
- It depends on the batch size
- In practice you can use the formula

num_worker = 4 * num_GPU .

Single- and Multi-process Data Loading

- The more data you put into the GPU memory, the less memory is available for the model.
- Single-process data loading data fetching is done in the same process a DataLoader is initialized, Therefore, data loading may block computing.
- However, this mode may be preferred when the entire dataset is small and can be loaded entirely in memory, often shows more readable error traces and thus is useful for debugging.
- the main process generates the indices using sampler and sends them to the workers. So any shuffle randomization is done in the main process which guides loading by assigning indices to load.
- **Memory Pinning**: Host to GPU copies are much faster when they originate from pinned (page-locked) memory.

Limited memory, huge tensors constrain

- If your data can’t fit into your cpu → you can’t load it all it once
  - Implement the loading logic in the ___getitem__() method instead of __init__
  - Load at least the batch size
  - Use running statistics in data preprocessing e.g moving average
  - Keep track of the elements in memory

References

- https://pytorch.org/docs/stable/torchvision/datasets.html
- https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader
- https://discuss.pytorch.org/t/guidelines-for-assigning-num-workers-to-dataloader/813