RBatchGenerator()

Uproot+Awkward

What are the problems?

- required more memory to test
  (maximum try 50 files for 15 GB RAM)

Next step to try!

- save each file to numpy file
- try to use torch dataloader and load from dir.
- compare speed of training and memory using in each approach.

Load with torch loader

```python
def load_dataset(data_path='/home/northnpk/Downloads/JetClass_dataset/',
                 tree_name = "tree",
                 batch_size = 128,
                 max_num_particles=128,
                 chunk_size = 5_000,
                 vec_columns=['part_px', 'part_py', 'part_pz', 'part_energy'],
                 columns=['jet_pt', 'jet_eta', 'jet_phi', 'jet_energy'],
                 targets = ['label_QCD', 'label_Hbb', 'label_Hcc', 'label_Hgg', 'label_H4q',
                            'label_Hqql', 'label_Zqq', 'label_Wqq', 'label_Tbqq', 'label_Tbl']) -> DataLoader:
    # Passing vector columns
    max_vec_sizes = dict(zip(vec_columns, [max_num_particles]*len(vec_columns)))

    # get files path
    train_path = [data_path + 'train/' + p for p in os.listdir(data_path + 'train')]

    # load RDataFrame from path
    r_df = RDataFrame(tree_name, train_path[:10])
    # Getting RBatchGenerator from tmva RBatchGenerator
    gen_train = ROOT.TMVA.Experimental.CreatePyTorchGenerators(r_df,
                                                               batch_size,
                                                               chunk_size,
                                                               columns=columns+vec_columns+targets,
                                                               max_vec_sizes=max_vec_sizes,
                                                               target=targets,
                                                               validation_split=0,
                                                               drop_remainder=False)

    # load Generator to torch dataloader
    dataset = RBatchDataset(gen_train)
    return DataLoader(dataset=dataset, collate_fn=dataset.collate_fn, batch_size=None)
```

Load with torch loader

```
# Define load_dataset method to loading TMVA RBatchGenerator into Torch Dataloader
def load_dataset(data_path='/home/northnpk/Downloads/JetClass_dataset/',
                 tree_name = "tree",
                 batch_size = 128,
                 max_num_particles=128,
                 chunk_size = 5_000,
                 vec_columns=['part_px', 'part_py', 'part_pz', 'part_energy'],
                 columns=['jet_pt', 'jet_eta', 'jet_phi', 'jet_energy'],
                 targets = ['label_QCD', 'label_Hbb', 'label_Hcc', 'label_Hgg', 'label_H4q',
                            'label_Hqql', 'label_Zqq', 'label_Wqq', 'label_Tbqq', 'label_Tbl']) -> DataLoader:
```

Load with torch loader

```
# Passing vector columns
    max_vec_sizes = dict(zip(vec_columns, [max_num_particles]*len(vec_columns)))

    # get files path
    train_path = [data_path + 'train/' + p for p in os.listdir(data_path + 'train')]

    # load RDataFrame from path
    r_df = RDataFrame(tree_name, train_path[:10])
```

Load with torch loader

```
# Getting RBatchGenerator from tmva RBatchGenerator
 gen_train = ROOT.TMVA.Experimental.CreatePyTorchGenerators(r_df,
                                                            batch_size,
                                                            chunk_size,
                                                            columns=columns+vec_columns+targets,
                                                            max_vec_sizes=max_vec_sizes,
                                                            target=targets,
                                                            validation_split=0,
                                                            drop_remainder=False)

 # load Generator to torch dataloader
 dataset = RBatchDataset(gen_train)
 return DataLoader(dataset=dataset, collate_fn=dataset.collate_fn, batch_size=None)
```

RBatchDataset(Iterabledataset)

```python
# Define Torch IterableDataset for TMVA RBatchGenerator
class RBatchDataset(IterableDataset):
    def __init__(self, generator, vector_prep_fn=None):
        self.generator = generator
        self.batch_size = self.generator.base_generator.batch_size
        self.vector_prep_fn = vector_prep_fn
        self.apply_vector_prep_fn = False
        if self.vector_prep_fn != None:
            self.apply_vector_prep_fn = True
        # print(f'Batch size: {self.batch_size}')
        if self.generator.last_batch_no_of_rows != 0 and self.generator.number_of_batches > 1:
            self.length = ((self.generator.number_of_batches-1) * self.batch_size) + self.generator.last_batch_no_of_rows
        elif self.generator.number_of_batches == 1:
            self.length = self.generator.last_batch_no_of_rows
        else :
            self.length = (self.generator.number_of_batches * self.batch_size)

        self.vec_columns = [c not in self.generator.base_generator.given_columns for c in self.generator.base_generator.train_columns]
        self._columns = [c in self.generator.base_generator.given_columns for c in self.generator.base_generator.train_columns]
        self.label_columns = self.generator.base_generator.target_columns
        self.vec_names = []
        for n in self.generator.base_generator.given_columns:
            if n not in self.label_columns and n not in self.generator.base_generator.train_columns:
                self.vec_names.append(n)

    def collate_fn(self, data):
        tensors, targets = data
        if self.apply_vector_prep_fn :
            return self.vector_prep_fn(tensors[:, self.vec_columns].reshape(len(tensors), 4, 128), self.vec_names), tensors[:, self._columns], targets

        else :
            return tensors[:, self.vec_columns].reshape(len(tensors), 4, 128), tensors[:, self._columns], targets

    def __iter__(self):
        return self.generator.__iter__()

    def __len__(self):
        return self.generator.number_of_batches

```

RBatchDataset(Iterabledataset)

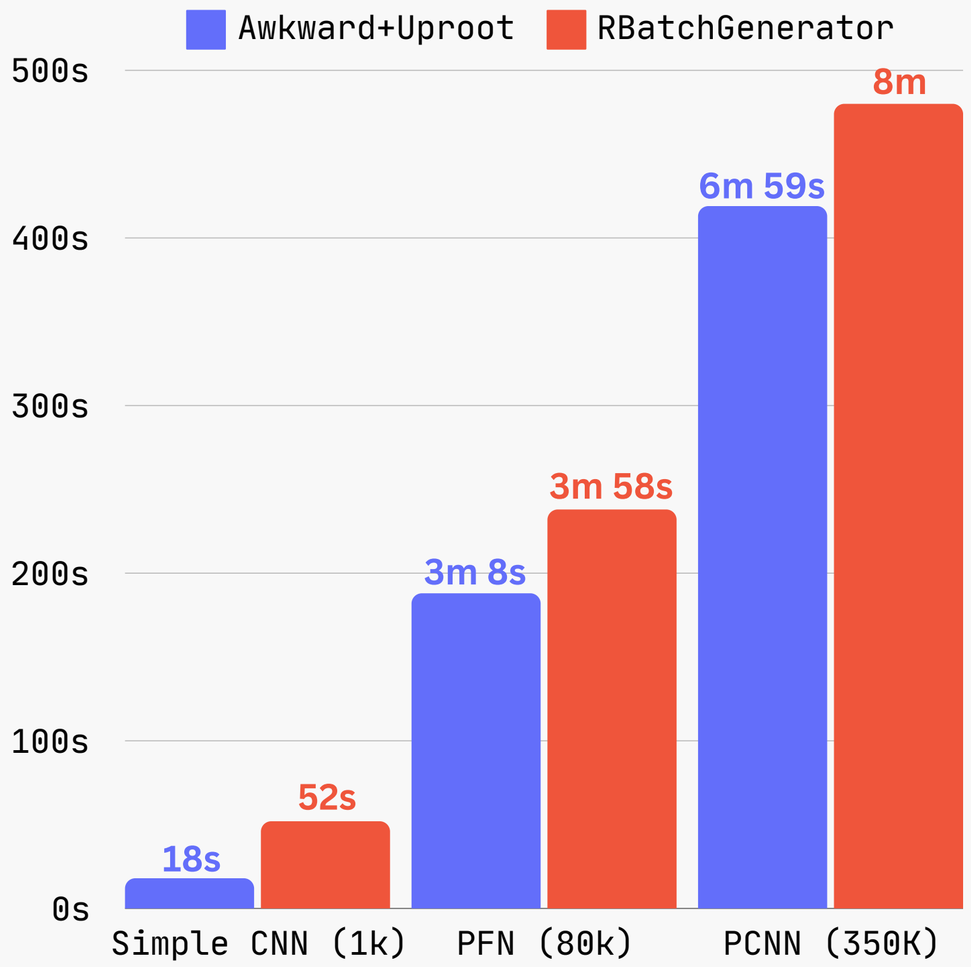
```
def collate_fn(self, data):
        tensors, targets = data
        if self.apply_vector_prep_fn :
            return self.vector_prep_fn(tensors[:, self.vec_columns].reshape(len(tensors), 4, 128), self.vec_names), tensors[:, self._columns], targets

        else :
            return tensors[:, self.vec_columns].reshape(len(tensors), 4, 128), tensors[:, self._columns], targets
```

RBatchGenerator()

training time with Pytorch
CPU only for 1 epoch
(Lower : Better)

- 10 files
- batch_size : 128
- dataset : JetClass



RBatchGenerator()

Next step to try!

- profiling with perf, memray to see What, Where, When, and How the memory use.
- Training Particle Transformers with RBatchGenerator

Thank you