smallTraining

July 24, 2023

[1]:
```
import torch
import json
import os
import random
import numpy as np
import torch
from sklearn import preprocessing
import torch.multiprocessing as mp
import torch
from torch.utils.data import DataLoader
import copy
from torch import nn
%matplotlib inline
import matplotlib.pyplot as plt
from dnaDataloader import expermentDataloader
from dnaDataloader import addData
from scipy import stats as st
from dnaModelUtil import train
device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")
cpu = torch.device("cpu")
batch_size = 25
device
```

[1]:
```
device(type='cuda', index=1)
```

[2]:
```
try:
   mp.set_start_method('spawn')
except RuntimeError:
   pass # throws error if run twice without resetting the kernal, if its␣
 ↪already set we dont care that this errors
```

0.1 Eval with small traing sets

The goal here is to eval the model while training on small sets of data. That is how well can MLP learn from 100-200 examples of single-molecule experiments. A verity of samples have been provided anf preprocessed (converted from excel to csv and had empty frames added) to the folder

[3]:
```
folders = [d[0] for d in os.walk("/home/khood/GitHub/SNN-DNA-project/
 ↪Prepocessing/sorted")][1:] # remove first one is it is "/home/khood/GitHub/
 ↪SNN-DNA-project/Prepocessing/sorted"
len(folders)
```

[3]:
```
6
```

[4]:
```
datasets = []
featIn = 0
for d in folders:
    data = expermentDataloader(
        f"{d}/index.csv",
        f"{d}",
    )
    rawData = [d for d in data]
    featIn = len(rawData[0][0])
    trainValidData = []
    testData = []
    addData(testData, trainValidData, rawData, rhsSize=300)


    np.random.shuffle(trainValidData)
    trainData = []
    validData = []
    addData(trainData, validData, trainValidData,␣
 ↪rhsSize=int(len(trainValidData)*(1/3)))

    datasets.append({"name": f"{os.path.basename(d)}",
                     "train":DataLoader(trainData, batch_size=batch_size,␣
 ↪shuffle=True) ,
                     "valid":DataLoader(validData, batch_size=batch_size,␣
 ↪shuffle=True) ,
                     "test":DataLoader(testData, batch_size=len(testData),␣
 ↪shuffle=True) ,
                     "model": {}}
                    )
```

[5]:
```
print(f"datasets:")
for d in datasets:
    print(d)
print(f"{len(datasets)}")
print(f"featIn: {featIn}")
```

```
datasets:
{'name': '1800_nM_AR_out', 'train': <torch.utils.data.dataloader.DataLoader
object at 0x7f52ee8cceb0>, 'valid': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3f4bf10>, 'test': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3f620b0>, 'model': {}}
```

```
{'name': '800_nM_AR_out', 'train': <torch.utils.data.dataloader.DataLoader
object at 0x7f5456899d50>, 'valid': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3f10550>, 'test': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3f10610>, 'model': {}}
{'name': '1200_nM_AR_out', 'train': <torch.utils.data.dataloader.DataLoader
object at 0x7f5456899c60>, 'valid': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3f7c430>, 'test': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3fbbeb0>, 'model': {}}
{'name': '400_nM_AR_out', 'train': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3f7fdf0>, 'valid': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3fb8040>, 'test': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3ff3fa0>, 'model': {}}
{'name': '50_nM_AR_out', 'train': <torch.utils.data.dataloader.DataLoader object
at 0x7f52e3fbbfa0>, 'valid': <torch.utils.data.dataloader.DataLoader object at
0x7f52e3ff0070>, 'test': <torch.utils.data.dataloader.DataLoader object at
0x7f52e3e23fa0>, 'model': {}}
{'name': '100_nM_AR_out', 'train': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3ff3eb0>, 'valid': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3e20190>, 'test': <torch.utils.data.dataloader.DataLoader
object at 0x7f52e3e5bd30>, 'model': {}}
6
featIn: 12000
```

[6]:
```python
for d in datasets:
    print(f"-- {d['name']} --")
    print(f"train: {len(d['train'])}")
    print(f"valid: {len(d['valid'])}")
    print(f"test : {len(d['test'])}")
```

```
-- 1800_nM_AR_out --
train: 8
valid: 4
test : 1
-- 800_nM_AR_out --
train: 8
valid: 4
test : 1
-- 1200_nM_AR_out --
train: 8
valid: 4
test : 1
-- 400_nM_AR_out --
train: 8
valid: 4
test : 1
-- 50_nM_AR_out --
train: 8
valid: 4
```

```
test : 1
-- 100_nM_AR_out --
train: 8
valid: 4
test : 1
```

[7]:
```
manager = mp.Manager()
return_dict = manager.dict()
```

[ ]:
```

```

[8]:
```
processes = []
devices = [torch.device("cuda:0"),torch.device("cuda:1"),torch.device("cuda:
 ↪2"),torch.device("cuda:3")]
epochs = 10000
error_margin = 20
for d in datasets:
    processes.append(mp.Process(target=train, args=(d["train"], d["valid"],␣
 ↪d["name"], featIn, return_dict, epochs, error_margin, devices[0])))
    devices.append(devices.pop(0))

processes
```

[8]:
```
[<Process name='Process-2' parent=1575574 initial>,
 <Process name='Process-3' parent=1575574 initial>,
 <Process name='Process-4' parent=1575574 initial>,
 <Process name='Process-5' parent=1575574 initial>,
 <Process name='Process-6' parent=1575574 initial>,
 <Process name='Process-7' parent=1575574 initial>]
```

[9]:
```
print(return_dict)
```

```
{}
```

[10]:
```
processesList = list(range(len(processes)))

while processesList:
    run = processesList[:4]
    processesList = processesList[4:]
    for i in run:
        processes[i].start()
    for i in run:
        processes[i].join()
        processes[i].terminate()
print(return_dict)
```

```
training 1800_nM_AR_out on cuda:0…
training 800_nM_AR_out on cuda:1…
training 1200_nM_AR_out on cuda:2…
```

```
training 400_nM_AR_out on cuda:3…
training 50_nM_AR_out on cuda:0…
training 100_nM_AR_out on cuda:1…
{'1800_nM_AR_out': {'path': './Models/smallTrain/1800_nM_AR_out.pt', 'acc':
0.54}, '800_nM_AR_out': {'path': './Models/smallTrain/800_nM_AR_out.pt', 'acc':
0.37000000000000005}, '1200_nM_AR_out': {'path':
'./Models/smallTrain/1200_nM_AR_out.pt', 'acc': 0.53}, '400_nM_AR_out': {'path':
'./Models/smallTrain/400_nM_AR_out.pt', 'acc': 0.43000000000000005},
'50_nM_AR_out': {'path': './Models/smallTrain/50_nM_AR_out.pt', 'acc':
0.8500000000000001}, '100_nM_AR_out': {'path':
'./Models/smallTrain/100_nM_AR_out.pt', 'acc': 0.52}}
```

[11]:
```
results = dict(return_dict)
with open("./Models/smallTrain/results.json", 'w') as file:
    json_object = json.dumps(results, indent=4)
    file.write(json_object)
```