In [1]:
```
from ANI1_release.ANI1_release.readers.lib import pyanitools as pya
from sklearn.model_selection import train_test_split


import torch
from torch import nn
import numpy as np
import torchani


from torch.optim import SGD, Adam
import torch.nn.functional as F
import random
from tqdm import tqdm
import math
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
```

2) (April 21) Network construction and workflow development. At this point you should have a working code that can train the network, demonstrated on small subset of the data.

In [2]:
```
from ANI1_release.ANI1_release.readers.lib.pyanitools import anidataloader
# data = anidataloader("../../ANI1_dataset/ANI-1_release/ani_gdb_s01.h5")
data = anidataloader("ANI1_release/ANI1_release/ani_gdb_s04.h5")
data_iter = data.__iter__()

mols = next(data_iter)
# Extract the data
P = mols['path']
X = mols['coordinates']
E = mols['energies']
S = mols['species']
sm = mols['smiles']

# Print the data
print("Path:   ", P)
print("  Smiles:      ","".join(sm))
print("  Symbols:     ", S)
print("  Coordinates: ", X.shape)
print("  Energies:    ", E.shape, "\n")




data_iter = data.__iter__()
count=0
count_conf =0
for mol in data_iter:
    count+=1
    count_conf += len(mol['energies'])
print(count)
print(count_conf)
```

```
Path:    /gdb11_s04/gdb11_s04-0
  Smiles:      [H]N([H])C([H])(C([H])([H])[H])C([H])([H])[H]
  Symbols:     ['C', 'C', 'C', 'N', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
', 'H']
  Coordinates:  (15840, 13, 3)
  Energies:    (15840,)

61
651936
```

customized aev function used to calculate aev of each molecule

In [3]:
```
def calc_aev(X,S):
    Rcr=5.2
    Rca=3.5
    EtaR=torch.tensor([16],dtype=torch.float)
    ShfR=torch.tensor([0.900000,1.168750,1.437500,1.706250,1.975000,2.243750
    EtaA=torch.tensor([8],dtype=torch.float)
    Zeta=torch.tensor([32],dtype=torch.float)
    ShfA=torch.tensor([0.900000,1.550000,2.200000,2.850000],dtype=torch.floa
    ShfZ=torch.tensor([0.19634954,0.58904862,0.9817477,1.3744468,1.7671459,2
    num_species=4
    X=torch.tensor(X,dtype=torch.float)
    mapping={"H":0, "C":1, "N":2, "O":3}
    elements = np.array([mapping[atom] for atom in S])
    elements2=torch.tensor(elements,dtype=torch.long).repeat(X.shape[0],1)
    aevs_X=torchani.AEVComputer(Rcr,Rca,EtaR,ShfR,EtaA,Zeta,ShfA,ShfZ,num_sp
    aev_calc=aevs_X((elements2,X))
    molecule_aev=aev_calc[1]
    molecule_aev_shape=aev_calc[1].shape
    return molecule_aev,molecule_aev_shape,S
```

In [4]:
```
import ANI1_release.ANI1_release.readers.lib.pyanitools as pya
seq=list()

# Set the HDF5 file containing the data
hdf5file = 'ANI1_release/ANI1_release/ani_gdb_s04.h5'

# Construct the data loader class
adl = pya.anidataloader(hdf5file)

# Print the species of the data set one by one
for data in adl:

    # Extract the data
    P = data['path']
    X = data['coordinates']
    E = data['energies']
    S = data['species']
    sm = data['smiles']

    # Print the data
    print("Path:   ", P)
    print("  Smiles:      ","".join(sm))
    print("  Symbols:     ", S)
    print("  Coordinates: ", X.shape)
    print("  Energies:    ", E.shape, "\n")




    seq.append(data)




list_coords=[]
for i in range(count):
    list_coords.append(seq[i]['coordinates'])


list_species=[]
for i in range(count):
    list_species.append(seq[i]['species'])


list_energies=[]
for i in range(count):
    list_energies.append(seq[i]['energies'])


list_species_coords_=zip(list_coords,list_species)
list_aevs_=[calc_aev(X,S) for X,S in list_species_coords_]

# print(list_aevs_[0][2][0])

aev_shape_array=[]
for i in range(count):
    aev_shape_array.append(list_aevs_[i][0].shape[2])
```

```
print(aev_shape_array)
# Closes the H5 data file

adl.cleanup()
```

```
Path:    /gdb11_s04/gdb11_s04-0
  Smiles:       [H]N([H])C([H])(C([H])([H])[H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'N', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
', 'H']
  Coordinates:  (15840, 13, 3)
  Energies:     (15840,)

Path:    /gdb11_s04/gdb11_s04-1
  Smiles:       [H]OC([H])(C([H])([H])[H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'O', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
']
  Coordinates:  (14400, 12, 3)
  Energies:     (14400,)

Path:    /gdb11_s04/gdb11_s04-10
  Smiles:       [H]OC(=O)O[H]
  Symbols:      ['O', 'C', 'O', 'O', 'H', 'H']
  Coordinates:  (5760, 6, 3)
  Energies:     (5760,)

Path:    /gdb11_s04/gdb11_s04-11
  Smiles:       [H]C([H])([H])C([H])([H])C([H])([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
', 'H', 'H']
  Coordinates:  (17280, 14, 3)
  Energies:     (17280,)

Path:    /gdb11_s04/gdb11_s04-12
  Smiles:       [H]N([H])C([H])([H])C([H])([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'N', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
', 'H']
  Coordinates:  (15840, 13, 3)
  Energies:     (15840,)

Path:    /gdb11_s04/gdb11_s04-13
  Smiles:       [H]OC([H])([H])C([H])([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'O', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
']
  Coordinates:  (14400, 12, 3)
  Energies:     (14400,)

Path:    /gdb11_s04/gdb11_s04-14
  Smiles:       [H]N([H])C([H])([H])C([H])([H])N([H])[H]
  Symbols:      ['N', 'C', 'C', 'N', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
']
  Coordinates:  (14400, 12, 3)
  Energies:     (14400,)

Path:    /gdb11_s04/gdb11_s04-15
  Smiles:       [H]OC([H])([H])C([H])([H])N([H])[H]
  Symbols:      ['N', 'C', 'C', 'O', 'H', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (12960, 11, 3)
  Energies:     (12960,)
```

```
Path:    /gdb11_s04/gdb11_s04-16
  Smiles:       [H]OC([H])([H])C([H])([H])O[H]
  Symbols:      ['O', 'C', 'C', 'O', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-17
  Smiles:       [H]N(C([H])([H])[H])C([H])([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'N', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
', 'H']
  Coordinates:  (15840, 13, 3)
  Energies:     (15840,)

Path:    /gdb11_s04/gdb11_s04-18
  Smiles:       [H]C([H])([H])OC([H])([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'O', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
']
  Coordinates:  (14400, 12, 3)
  Energies:     (14400,)

Path:    /gdb11_s04/gdb11_s04-19
  Smiles:       [H]C([H])=C([H])C([H])([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
']
  Coordinates:  (14400, 12, 3)
  Energies:     (14400,)

Path:    /gdb11_s04/gdb11_s04-2
  Smiles:       [H]C([H])([H])N(C([H])([H])[H])C([H])([H])[H]
  Symbols:      ['C', 'N', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
', 'H']
  Coordinates:  (15840, 13, 3)
  Energies:     (15840,)

Path:    /gdb11_s04/gdb11_s04-20
  Smiles:       [H]C(=O)C([H])([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'O', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-21
  Smiles:       [H]C([H])([H])C([H])([H])C#N
  Symbols:      ['C', 'C', 'C', 'N', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (10080, 9, 3)
  Energies:     (10080,)

Path:    /gdb11_s04/gdb11_s04-22
  Smiles:       [H]C([H])=C([H])C([H])([H])N([H])[H]
  Symbols:      ['N', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (12960, 11, 3)
  Energies:     (12960,)

Path:    /gdb11_s04/gdb11_s04-23
  Smiles:       [H]OC([H])([H])C([H])=C([H])[H]
  Symbols:      ['O', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-24
```

```
  Smiles:       [H]OC([H])([H])C([H])=O
  Symbols:      ['O', 'C', 'C', 'O', 'H', 'H', 'H', 'H']
  Coordinates:  (8640, 8, 3)
  Energies:     (8640,)

Path:    /gdb11_s04/gdb11_s04-25
  Smiles:       [H]OC([H])([H])C#N
  Symbols:      ['O', 'C', 'C', 'N', 'H', 'H', 'H']
  Coordinates:  (7200, 7, 3)
  Energies:     (7200,)

Path:    /gdb11_s04/gdb11_s04-26
  Smiles:       [H]N=C([H])N([H])C([H])([H])[H]
  Symbols:      ['C', 'N', 'C', 'N', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-27
  Smiles:       [H]C(=O)N([H])C([H])([H])[H]
  Symbols:      ['C', 'N', 'C', 'O', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (10080, 9, 3)
  Energies:     (10080,)

Path:    /gdb11_s04/gdb11_s04-28
  Smiles:       [H]N=C([H])N([H])N([H])[H]
  Symbols:      ['N', 'N', 'C', 'N', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (10080, 9, 3)
  Energies:     (10080,)

Path:    /gdb11_s04/gdb11_s04-29
  Smiles:       [H]C(=O)N([H])N([H])[H]
  Symbols:      ['N', 'N', 'C', 'O', 'H', 'H', 'H', 'H']
  Coordinates:  (6837, 8, 3)
  Energies:     (6837,)

Path:    /gdb11_s04/gdb11_s04-3
  Smiles:       [H]C([H])=C(C([H])([H])[H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
']
  Coordinates:  (14400, 12, 3)
  Energies:     (14400,)

Path:    /gdb11_s04/gdb11_s04-30
  Smiles:       [H]N=C([H])N([H])O[H]
  Symbols:      ['O', 'N', 'C', 'N', 'H', 'H', 'H', 'H']
  Coordinates:  (8640, 8, 3)
  Energies:     (8640,)

Path:    /gdb11_s04/gdb11_s04-31
  Smiles:       [H]ON([H])C([H])=O
  Symbols:      ['O', 'N', 'C', 'O', 'H', 'H', 'H']
  Coordinates:  (7200, 7, 3)
  Energies:     (7200,)

Path:    /gdb11_s04/gdb11_s04-32
  Smiles:       [H]C([H])=NN([H])C([H])([H])[H]
  Symbols:      ['C', 'N', 'N', 'C', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11518, 10, 3)
  Energies:     (11518,)
```

```
Path:    /gdb11_s04/gdb11_s04-33
  Smiles:       [H]C([H])=C([H])OC([H])([H])[H]
  Symbols:      ['C', 'O', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-34
  Smiles:       [H]C(=O)OC([H])([H])[H]
  Symbols:      ['C', 'O', 'C', 'O', 'H', 'H', 'H', 'H']
  Coordinates:  (8640, 8, 3)
  Energies:     (8640,)

Path:    /gdb11_s04/gdb11_s04-35
  Smiles:       [H]C([H])=NOC([H])([H])[H]
  Symbols:      ['C', 'O', 'N', 'C', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (10080, 9, 3)
  Energies:     (10080,)

Path:    /gdb11_s04/gdb11_s04-36
  Smiles:       [H]C#CC([H])([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-37
  Smiles:       [H]C#CC([H])([H])N([H])[H]
  Symbols:      ['N', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (10080, 9, 3)
  Energies:     (10080,)

Path:    /gdb11_s04/gdb11_s04-38
  Smiles:       [H]C#CC([H])([H])O[H]
  Symbols:      ['O', 'C', 'C', 'C', 'H', 'H', 'H', 'H']
  Coordinates:  (8640, 8, 3)
  Energies:     (8640,)

Path:    /gdb11_s04/gdb11_s04-39
  Smiles:       [H]C([H])=C([H])C([H])=C([H])[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-4
  Smiles:       [H]C([H])([H])C(=O)C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'O', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11519, 10, 3)
  Energies:     (11519,)

Path:    /gdb11_s04/gdb11_s04-40
  Smiles:       [H]C(=O)C([H])=C([H])[H]
  Symbols:      ['C', 'C', 'C', 'O', 'H', 'H', 'H', 'H']
  Coordinates:  (8640, 8, 3)
  Energies:     (8640,)

Path:    /gdb11_s04/gdb11_s04-41
  Smiles:       [H]C([H])=C([H])C#N
  Symbols:      ['C', 'C', 'C', 'N', 'H', 'H', 'H']
  Coordinates:  (7200, 7, 3)
```

```
  Energies:     (7200,)

Path:    /gdb11_s04/gdb11_s04-42
  Smiles:       [H]C(=O)C([H])=O
  Symbols:      ['O', 'C', 'C', 'O', 'H', 'H']
  Coordinates:  (5760, 6, 3)
  Energies:     (5760,)

Path:    /gdb11_s04/gdb11_s04-43
  Smiles:       [H]C(=O)C#N
  Symbols:      ['O', 'C', 'C', 'N', 'H']
  Coordinates:  (4320, 5, 3)
  Energies:     (4320,)

Path:    /gdb11_s04/gdb11_s04-44
  Smiles:       N#CC#N
  Symbols:      ['N', 'C', 'C', 'N']
  Coordinates:  (2880, 4, 3)
  Energies:     (2880,)

Path:    /gdb11_s04/gdb11_s04-45
  Smiles:       [H]C([H])=NN=C([H])[H]
  Symbols:      ['C', 'N', 'N', 'C', 'H', 'H', 'H', 'H']
  Coordinates:  (8640, 8, 3)
  Energies:     (8640,)

Path:    /gdb11_s04/gdb11_s04-46
  Smiles:       [H]C#CC([H])=C([H])[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H', 'H', 'H']
  Coordinates:  (8640, 8, 3)
  Energies:     (8640,)

Path:    /gdb11_s04/gdb11_s04-47
  Smiles:       [H]C#CC([H])=O
  Symbols:      ['O', 'C', 'C', 'C', 'H', 'H']
  Coordinates:  (5760, 6, 3)
  Energies:     (5760,)

Path:    /gdb11_s04/gdb11_s04-48
  Smiles:       [H]C#CC#N
  Symbols:      ['C', 'C', 'C', 'N', 'H']
  Coordinates:  (4320, 5, 3)
  Energies:     (4320,)

Path:    /gdb11_s04/gdb11_s04-49
  Smiles:       [H]C#CC#C[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H']
  Coordinates:  (5760, 6, 3)
  Energies:     (5760,)

Path:    /gdb11_s04/gdb11_s04-5
  Smiles:       [H]N=C(N([H])[H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'N', 'N', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-51
  Smiles:       [H]C(=NN([H])[H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'N', 'N', 'H', 'H', 'H', 'H', 'H', 'H']
```

```
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-52
  Smiles:       [H]ON=C([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'N', 'O', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (9966, 9, 3)
  Energies:     (9966,)

Path:    /gdb11_s04/gdb11_s04-54
  Smiles:       [H]C([H])([H])C1([H])C([H])([H])C1([H])[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
']
  Coordinates:  (14400, 12, 3)
  Energies:     (14400,)

Path:    /gdb11_s04/gdb11_s04-55
  Smiles:       [H]N([H])C1([H])C([H])([H])C1([H])[H]
  Symbols:      ['N', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (12960, 11, 3)
  Energies:     (12960,)

Path:    /gdb11_s04/gdb11_s04-56
  Smiles:       [H]OC1([H])C([H])([H])C1([H])[H]
  Symbols:      ['O', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-57
  Smiles:       [H]N1C([H])([H])C1([H])C([H])([H])[H]
  Symbols:      ['C', 'C', 'C', 'N', 'H', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (12960, 11, 3)
  Energies:     (12960,)

Path:    /gdb11_s04/gdb11_s04-58
  Smiles:       [H]C([H])([H])C1([H])OC1([H])[H]
  Symbols:      ['C', 'C', 'C', 'O', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11520, 10, 3)
  Energies:     (11520,)

Path:    /gdb11_s04/gdb11_s04-59
  Smiles:       [H]C([H])([H])N1C([H])([H])C1([H])[H]
  Symbols:      ['C', 'N', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (12960, 11, 3)
  Energies:     (12960,)

Path:    /gdb11_s04/gdb11_s04-6
  Smiles:       [H]N([H])C(=O)C([H])([H])[H]
  Symbols:      ['C', 'C', 'N', 'O', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (6048, 9, 3)
  Energies:     (6048,)

Path:    /gdb11_s04/gdb11_s04-60
  Smiles:       [H]C1([H])C([H])([H])C([H])([H])C1([H])[H]
  Symbols:      ['C', 'C', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H
']
  Coordinates:  (14400, 12, 3)
  Energies:     (14400,)
```

```
Path:    /gdb11_s04/gdb11_s04-61
  Smiles:       [H]N1C([H])([H])C([H])([H])C1([H])[H]
  Symbols:      ['C', 'C', 'N', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (12960, 11, 3)
  Energies:     (12960,)

Path:    /gdb11_s04/gdb11_s04-62
  Smiles:       [H]C1([H])OC([H])([H])C1([H])[H]
  Symbols:      ['C', 'C', 'O', 'C', 'H', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (11461, 10, 3)
  Energies:     (11461,)

Path:    /gdb11_s04/gdb11_s04-7
  Smiles:       [H]OC(=O)C([H])([H])[H]
  Symbols:      ['C', 'C', 'O', 'O', 'H', 'H', 'H', 'H']
  Coordinates:  (8507, 8, 3)
  Energies:     (8507,)

Path:    /gdb11_s04/gdb11_s04-8
  Smiles:       [H]N=C(N([H])[H])N([H])[H]
  Symbols:      ['N', 'C', 'N', 'N', 'H', 'H', 'H', 'H', 'H']
  Coordinates:  (10080, 9, 3)
  Energies:     (10080,)

Path:    /gdb11_s04/gdb11_s04-9
  Smiles:       [H]N([H])C(=O)N([H])[H]
  Symbols:      ['N', 'C', 'N', 'O', 'H', 'H', 'H', 'H']
  Coordinates:  (8640, 8, 3)
  Energies:     (8640,)
```

```
/home/joyce/miniconda3/lib/python3.9/site-packages/torchani/aev.py:236: Use
rWarning: __floordiv__ is deprecated, and its behavior will change in a fut
ure version of pytorch. It currently rounds toward 0 (like the 'trunc' func
tion NOT 'floor'). This results in incorrect rounding for negative values.
To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), o
r for actual floor division, use torch.div(a, b, rounding_mode='floor').
  pair_sizes = counts * (counts - 1) // 2
[384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384,
384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384,
384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384,
384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384, 384,
384]
```

data generator that batches into a molecules'confirmations

In [5]:
```
def data_gen(X,y, batchsize):
    '''
    Generator for data
    '''
    if len(X)//batchsize == 0:
        return X[:batchsize],y[:batchsize]

    for i in range(len(X)//batchsize):
        yield X[i*batchsize:(i+1)*batchsize],y[i*batchsize:(i+1)*batchsize]
    i+=1
    yield X[i*batchsize:],y[i*batchsize:]
```

In [6]:
```
from functools import wraps
from time import time

def timing(f):
    @wraps(f)
    def wrap(*args, **kw):
        ts = time()
        result = f(*args, **kw)
        te = time()
        print('func:%r  took: %2.4f sec' % (f.__name__,  te-ts))
        return result
    return wrap
```

trainer class takes X's that correspond to the molecules' coordinates and molecules actual energy, atom_types is a list of lists of atoms that make up that individual molecule. Loss is collected for each molecule then added iteratively.

In [7]:
```
class Trainer():
    def __init__(self, model, optimizer_type, learning_rate, epoch, batch_si
        """ The class for training the model
        model: nn.Module
            A pytorch model
        optimizer_type: 'adam' or 'sgd'
        learning_rate: float
        epoch: int
        batch_size: int
        input_transform: func
            transforming input. Can do reshape here
        """
        self.model = model
        if optimizer_type == "sgd":
            self.optimizer = SGD(model.parameters(), learning_rate,momentum=
        elif optimizer_type == "adam":
            self.optimizer = Adam(model.parameters(), learning_rate)

        self.epoch = epoch
        self.batch_size = batch_size
        self.input_transform = input_transform


    @timing
    def train(self, inputs, outputs, val_inputs, val_outputs, test_inputs,te
        """ train self.model with specified arguments
        inputs: np.array, The shape of input_transform(input) should be (nda
        outputs: np.array shape (ndata,)
        val_nputs: np.array, The shape of input_transform(val_input) should
        val_outputs: np.array shape (ndata,)
        early_stop: bool
        l2: bool
        silent: bool. Controls whether or not to print the train and val err
        """

        train_gen_coords_energies=[]
        for i in range(count):
            train_gen=data_gen(inputs[i],outputs[i],self.batch_size)
            train_gen_coords_energies.append(next(train_gen))

        train_gen_coords=[]
        for i in range(count):
            train_gen_coords.append(train_gen_coords_energies[i][0])

        train_gen_energies=[]
        for i in range(count):
            train_gen_energies.append(train_gen_coords_energies[i][1])


        list_species_coords=zip(train_gen_coords,atom_types)
        list_aevs=[calc_aev(X,S) for X,S in list_species_coords]

#         for i in range(count):
#             print(list_aevs[i][0].shape)



        losses = []
        val_losses = []
```

```
        val_losses = []
        test_losses=[]
        weights = self.model.state_dict()
        lowest_val_loss = np.inf

        lowest_test_loss = np.inf




        for n_epoch in tqdm(range(self.epoch), leave=False):
            self.model.train()

            epoch_loss = 0
            epoch_acc = 0


            for i in range(len(list_aevs)):
                batch_predictions = self.model(list_aevs[i][0],list_aevs[i][
                batch_importance = len(train_gen_coords_energies[i][1]) / le
                loss = nn.MSELoss()(batch_predictions, torch.tensor(train_ge


                if l2:
                    l2_lambda = 1e-5
                    l2_norm = sum(p.pow(2.0).sum() for p in self.model.param
                    loss = loss + l2_lambda * l2_norm
                self.optimizer.zero_grad()
                loss.backward(retain_graph=True)
                self.optimizer.step()
                epoch_loss += loss.detach().cpu().item() * batch_importance

            val_loss = self.evaluate(val_inputs, val_outputs, atom_types,pri
            if n_epoch % 10 ==0 and not silent:
                print("Epoch %d/%d - Loss: %.3f" % (n_epoch + 1, self.epoch,
                print("              Val_loss: %.3f " % (val_loss))

            losses.append(epoch_loss)
            val_losses.append(val_loss)

            test_loss=self.evaluate(test_inputs, test_outputs, atom_types,pr
            if n_epoch % 10 ==0 and not silent:
                print("Epoch %d/%d - Loss: %.3f" % (n_epoch + 1, self.epoch,
                print("              Test_loss: %.3f " % (test_loss))

            test_losses.append(test_loss)

            if early_stop:
                if val_loss < lowest_val_loss:
                    lowest_val_loss = val_loss
                    weights = self.model.state_dict()

            if early_stop:
                if val_loss < lowest_test_loss:
                    lowest_test_loss = test_loss
                    weights = self.model.state_dict()
        if draw_curve:
            plt.figure()
            plt.plot(np.arange(self.epoch) + 1,losses,label='Training loss')
```

```
            plt.plot(np.arange(self.epoch) + 1,losses,label='Training loss')
            plt.plot(np.arange(self.epoch) + 1,val_losses,label='Validation
            plt.xlabel('Epochs')
            plt.ylabel('Loss')
            plt.legend()

        if early_stop:
            self.model.load_state_dict(weights)


        return {"losses": losses,  "val_losses": val_losses, "test_losses":

    def evaluate(self, inputs, outputs, atom_types,print_acc=True):
        gen_coords_energies=[]
        for i in range(count):
            gen=data_gen(inputs[i],outputs[i],self.batch_size)
            gen_coords_energies.append(next(gen))

        gen_coords=[]
        for i in range(count):
            gen_coords.append(gen_coords_energies[i][0])

        gen_energies=[]
        for i in range(count):
            gen_energies.append(gen_coords_energies[i][1])


        list_species_coords=zip(gen_coords,atom_types)
        list_aevs=[calc_aev(X,S) for X,S in list_species_coords]


        losses = 0

        for i in range(len(list_aevs)):

            batch_importance = len(gen_coords_energies[i][1]) / len(outputs)

            with torch.no_grad():
                batch_predictions = self.model(list_aevs[i][0],list_aevs[i][
                loss = nn.MSELoss()(batch_predictions, torch.tensor(gen_ener


            losses += loss.detach().cpu().item() * batch_importance

        return losses
```

neural network that takes atom type iteratively than predicts an energy for each molecule

In [8]:
```
class ANI(nn.Module):
    def __init__(self):
        super().__init__()
        self.sub_nets = nn.ModuleDict({"C": ANI_sub(nn.Sequential(nn.Linear(
                                        "H": ANI_sub(nn.Sequential(nn.Linear(
                                        "N": ANI_sub(nn.Sequential(nn.Linear(
                                        "O": ANI_sub(nn.Sequential(nn.Linear(

    def forward(self, aevs, atom_types):
        num_conf, num_atoms, aev_vec=aevs.shape
        atomic_energies=torch.empty((num_conf, 0),dtype=torch.float)
        aev_shape=aevs.reshape(num_atoms, num_conf, aev_vec)
        for i in range(len(atom_types)):
            atom=self.sub_nets[atom_types[i]](aev_shape[i])
            atom=torch.tensor(atom,dtype=torch.float)

            torch.cat((atomic_energies,atom),1)

        total_energies = torch.sum(atomic_energies,dim=-1,dtype=torch.float)
        return total_energies

class ANI_sub(nn.Module):
    def __init__(self, architecture):
        super().__init__()
        self.layers= architecture

    def forward(self, aev):
        atomic_energy = self.layers(aev)
        return atomic_energy
```

train_test_split for tests, validation, and train tests for each molecule that is appended to each respective lists

In [9]:
```
train_X=[]
train_y=[]
test_X=[]
test_y=[]
val_X=[]
val_y=[]
for i in range(count):
    train_Xs, test_Xs, train_ys, test_ys = train_test_split(list_coords[i],l
    train_Xs, val_Xs, train_ys, val_ys = train_test_split(train_Xs,train_ys,

    train_X.append(train_Xs)
    train_y.append(train_ys)
    test_X.append(test_Xs)
    test_y.append(test_ys)
    val_X.append(val_Xs)
    val_y.append(val_ys)
```

trainer class running to measure loss using MSE

In [10]:
```
model=ANI()
trainer = Trainer(model, 'adam', 1e-3, 50, 60,)
```

In [11]:
```
log=trainer.train(train_X, train_y, val_X, val_y, test_X,test_y,list_species
```

```
  0%|                                                    | 0/50 [00:00<?, ?
it/s]/tmp/ipykernel_11365/3245812991.py:15: UserWarning: To copy construct
from a tensor, it is recommended to use sourceTensor.clone().detach() or so
urceTensor.clone().detach().requires_grad_(True), rather than torch.tensor
(sourceTensor).
  atom=torch.tensor(atom,dtype=torch.float)
Epoch 1/50 - Loss: 2238001.790
             Val_loss: 2237991.246
  2%|▏                                                 | 1/50 [00:05<04:14,  5.20
s/it]
Epoch 1/50 - Loss: 2238001.790
             Test_loss: 2237982.860
 20%|████████                                          | 10/50 [00:27<01:39,  2.48
s/it]
Epoch 11/50 - Loss: 2238001.790
             Val_loss: 2237991.246
 22%|████████▊                                         | 11/50 [00:29<01:36,  2.47
s/it]
Epoch 11/50 - Loss: 2238001.790
             Test_loss: 2237982.860
 40%|████████████████                                  | 20/50 [00:53<01:15,  2.53
s/it]
Epoch 21/50 - Loss: 2238001.790
             Val_loss: 2237991.246
 42%|████████████████▊                                 | 21/50 [00:55<01:12,  2.51
s/it]
Epoch 21/50 - Loss: 2238001.790
             Test_loss: 2237982.860
 60%|████████████████████████                          | 30/50 [01:17<00:49,  2.50
s/it]
Epoch 31/50 - Loss: 2238001.790
             Val_loss: 2237991.246
 62%|████████████████████████▊                         | 31/50 [01:20<00:48,  2.57
s/it]
Epoch 31/50 - Loss: 2238001.790
             Test_loss: 2237982.860
 80%|████████████████████████████████                  | 40/50 [01:43<00:25,  2.51
s/it]
Epoch 41/50 - Loss: 2238001.790
             Val_loss: 2237991.246
 82%|████████████████████████████████▊                 | 41/50 [01:46<00:22,  2.50
s/it]
Epoch 41/50 - Loss: 2238001.790
             Test_loss: 2237982.860
```

```
func:'train'  took: 129.5226 sec
```