NNEOSB

November 2, 2022

Contents

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300
import random
import csv
import pandas as pd
import torch
from torch import nn # pytorch neural networks
from torch.utils.data import Dataset, DataLoader # pytorch dataset structures
from torchvision.transforms import ToTensor # pytorch transformer
# from torch.utils.data import DataLoader
# from torchvision import datasets
# from torchvision.transforms import ToTensor
```

1 Introduction

The conserved variables are (D, S_i, τ) and they are related to primitive variables, $w = (\rho, v^i, \epsilon, p)$, defined in the local rest frame of the fluid through (in units of light speed $c = 1$). The P2C is explicitly given:

$$D = \rho W\,, \quad S_i = \rho h W^2 v_i\,, \quad \tau = \rho h W^2 - p - D\,, \tag{1}$$

where we used

$$W = (1 - v^2)^{-1/2}\,, \quad h = 1 + \epsilon + \frac{p}{\rho}\,. \tag{2}$$

Our first goal is to reproduce the results from this paper. We first focus on what they call **NNEOS** networks. These are networks which are trained to infer information on the equation of state (EOS). In its simplest form, the EOS is the thermodynamical relation connecting the pressure to the fluid's rest-mass density and internal energy $p = \bar{p}(\rho, \epsilon)$. We consider an **analytical Γ-law EOS** as a benchmark:

$$\bar{p}(\rho, \varepsilon) = (\Gamma - 1)\rho\epsilon \,, \tag{3}$$

and we fix $\Gamma = 5/3$ in order to fully mimic the situation of the paper.

2 Generating training data

We generate training data for the NNEOS networks as follows. We create a training set by randomly sampling the EOS on a uniform distribution over $\rho \in (0, 10.1)$ and $\epsilon \in (0, 2.02)$. Below, we first focus on the implementation of **NNEOSB** as called in the paper, meaning we also make the derivatives of the EOS part of the output. So we compute three quantities:

- p, using the EOS defined above
- $\chi := \partial p/\partial \rho$, inferred from the EOS
- $\kappa := \partial p/\partial \epsilon$, inferred from the EOS

[2]:
```
# Define the three functions determining the output
def eos(rho, eps, Gamma = 5/3):
    """Computes the analytical gamma law EOS from rho and epsilon"""
    return (Gamma - 1) * rho * eps

def chi(rho, eps, Gamma = 5/3):
    """Computes dp/drho from EOS"""
    return (Gamma - 1) * eps

def kappa(rho, eps, Gamma = 5/3):
    """Computes dp/deps from EOS"""
    return (Gamma - 1) * rho
```

[3]:
```
# Define ranges of parameters to be sampled (see paper Section 2.1)
rho_min = 0
rho_max = 10.1
eps_min = 0
eps_max = 2.02
```

Note: the code in comment below was used to generate the data. It has now been saved separately in a folder called "data".

[4]:
```
# number_of_datapoints = 10000 # 80 000 for train, 10 000 for test
# data = []

# for i in range(number_of_datapoints):
#     rho = random.uniform(rho_min, rho_max)
#     eps = random.uniform(eps_min, eps_max)
```

```
#     new_row = [rho, eps, eos(rho, eps), chi(rho, eps), kappa(rho, eps)]

#     data.append(new_row)
```

[5]:
```
# header = ['rho', 'eps', 'p', 'chi', 'kappa']

# with open('NNEOS_data_test.csv', 'w', newline = '') as file:
#     writer = csv.writer(file)
#     # write header
#     writer.writerow(header)
#     # write data
#     writer.writerows(data)
```

[6]:
```
# Import data
data_train = pd.read_csv("data/NNEOS_data_train.csv")
data_test = pd.read_csv("data/NNEOS_data_test.csv")
print("The training data has " + str(len(data_train)) + " instances")
print("The test data has " + str(len(data_test)) + " instances")
data_train
```

```
The training data has 80000 instances
The test data has 10000 instances
```

[6]:

	rho	eps	p	chi	kappa
0	9.770794	0.809768	5.274717	0.539845	6.513863
1	10.093352	0.575342	3.871421	0.383561	6.728901
2	1.685186	1.647820	1.851255	1.098547	1.123457
3	1.167718	0.408377	0.317913	0.272251	0.778479
4	7.750848	1.069954	5.528700	0.713303	5.167232
…	…	…	…	…	…
79995	3.985951	1.642317	4.364131	1.094878	2.657301
79996	6.948815	0.809021	3.747824	0.539347	4.632543
79997	8.423227	1.125142	6.318217	0.750095	5.615485
79998	4.748173	0.774870	2.452810	0.516580	3.165449
79999	2.927483	0.616751	1.203686	0.411167	1.951655

[80000 rows x 5 columns]

In case we want to visualize the datapoints (not useful, nothing significant happening).

[7]:
```
# rho = data_train['rho']
# eps = data_train['eps']

# plt.figure(figsize = (12,10))
# plt.plot(rho, eps, 'o', color = 'black', alpha = 0.005)
# plt.grid()
# plt.xlabel(r'$\rho$')
```

```
# plt.ylabel(r'$\epsilon$')
# plt.title('Training data')
# plt.show()
```

3 Getting data into PyTorch's DataLoader

Below: all_data is of the type $(\rho, \epsilon, p, \chi, \kappa)$ as generated above.

[8]:
```
class CustomDataset(Dataset):
    """See PyTorch tutorial: the following three methods HAVE to be␣
↪implemented"""

    def __init__(self, all_data, transform=None, target_transform=None):
        self.transform = transform
        self.target_transform = target_transform

        # Separate features (rho and eps) from the labels (p, chi, kappa)
        # (see above to get how data is organized)
        features = []
        labels = []

        for i in range(len(all_data)):
            # Separate the features
            new_feature = [all_data['rho'][i], all_data['eps'][i]]
            features.append(torch.tensor(new_feature, dtype = torch.float32))
            # Separate the labels
            new_label = [all_data['p'][i], all_data['chi'][i],␣
↪all_data['kappa'][i]]
            labels.append(torch.tensor(new_label, dtype = torch.float32))

        # Save as instance variables to the dataloader
        self.features = features
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    # TODO: I don't understand transform and target_transform --- but this is␣
↪not used now!
    def __getitem__(self, idx):
        feature = self.features[idx]
        if self.transform:
            feature = transform(feature)
        label = self.labels[idx]
        if self.target_transform:
            feature = target_transform(label)
```

```
        return feature, label
```

Note that the following cell may be confusing. "data_train" refers to the data that was generatd above, see the pandas table. "training_data" is defined similarly as in the PyTorch tutorial, see this page and this is an instance of the class CustomDataset defined above.

[9]:
```
# Make training and test data, as in the tutorial
training_data = CustomDataset(data_train)
test_data = CustomDataset(data_test)
```

[10]:
```
# Check if this is done correctly
print(training_data.features[:2])
print(training_data.labels[:2])
print(training_data.__len__())
print(test_data.__len__())
```

```
[tensor([9.7708, 0.8098]), tensor([10.0934,  0.5753])]
[tensor([5.2747, 0.5398, 6.5139]), tensor([3.8714, 0.3836, 6.7289])]
80000
10000
```

[11]:
```
# Now call DataLoader on the above CustomDataset instances:
train_dataloader = DataLoader(training_data, batch_size=32)
test_dataloader = DataLoader(test_data, batch_size=32)
```

4 Building the neural networks

We will follow this part of the PyTorch tutorial. For more information, see the documentation page of torch.nn. We take the parameters of NNEOSB in the paper, see Table 1. **Question:** I'm not sure how the ReLU at the output layer is done... For now, we just use sigmoids in the hidden layers. **To do** check other activation functions.

[12]:
```
# Define hyperparameters of the model here. Will first of all put two hidden␣
 ↪layers
device = "cpu"
size_HL_1 = 400
size_HL_2 = 600

# Implement neural network
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        #self.flatten = nn.Flatten()
        self.stack = nn.Sequential(
            nn.Linear(2, size_HL_1),
            nn.Sigmoid(),
            nn.Linear(size_HL_1, size_HL_2),
```

```
            nn.Sigmoid(),
            nn.Linear(size_HL_2, 3) ### Q: Does this have to become ReLU? Gives␣
↪an error!
            ###nn.ReLU() # ???
        )

    def forward(self, x):
        #x = self.flatten(x) ### no flatten needed, as our input and output are␣
↪1D?
        logits = self.stack(x)
        return logits
```

5 Training the neural network

Now we generate an instance of the above neural network in `model` (note: running this cell will create a 'fresh' model!)

[13]:
```
model = NeuralNetwork().to(device)
print(model)
```

```
NeuralNetwork(
  (stack): Sequential(
    (0): Linear(in_features=2, out_features=400, bias=True)
    (1): Sigmoid()
    (2): Linear(in_features=400, out_features=600, bias=True)
    (3): Sigmoid()
    (4): Linear(in_features=600, out_features=3, bias=True)
  )
)
```

Save hyperparameters and loss function - note that we follow the paper. I think that their loss function agrees with MSELoss. The paper uses the Adam optimizer. More details on optimizers can be found here. Required argument `params` can be filled in by calling `model` which contains the neural network. For simplicity we will train for 10 epochs here. **Question:** how many epochs should be used? What size for the batches,…

[14]:
```
# Save hyperparameters here --- see paper!!!
learning_rate = 6e-4
batch_size = 32
epochs = 200
# Initialize the loss function
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

The train and test loops are implemented below (copy pasted from this part of the tutorial):

[15]:
```
def train_loop(dataloader, model, loss_fn, optimizer, report_progress = False):
    """The training loop of the algorithm"""
```

```
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # If we want to report progress during training (not recommended -␣
 ↪obstructs view)
        if report_progress:
            if batch % 100 == 0:
                loss, current = loss.item(), batch * len(X)
                print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")


def test_loop(dataloader, model, loss_fn):
    """The testing loop of the algorithm"""
    num_batches = len(dataloader)
    test_loss = 0

    # Predict and compute losses
    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()

    average_test_loss = test_loss/num_batches
    return average_test_loss
```

The following function allows us to select a subset of the training data with the same size as the training set (if desired) and use it as a separate test set. **Question:** will the authors have computed the loss on the *whole* training set, or 10 000 random instances sampled from the training set?

[16]:
```
def get_subset_train_dataloader(data_train, size = 10000):
    """Creates a 'subset' of dataloader for computing loss on training data.
        This way we can 'test' on training data too - to check the claim of the␣
 ↪paper about overfitting. """

    # Get random ids to sample
    random_ids =  np.random.choice(len(data_train), size, replace=False)

    # the following is a pandas dataframe
    sampled_train_data = data_train.iloc[random_ids]
```

```
    # relabel the indices
    sampled_train_data.index = [i for i in range(len(sampled_train_data))]
    new_dataset = CustomDataset(sampled_train_data)

    # Make it a dataloader and return it
    new_dataloader = DataLoader(new_dataset, batch_size=32)

    return new_dataloader
```

[17]:
```
# Testing to see if this works!
hey = [i for i in range(15)]
print(hey)
print(hey[-5:], len(hey[-5:]))
print(hey[-10:-5], len(hey[-10:-5]))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[10, 11, 12, 13, 14] 5
[5, 6, 7, 8, 9] 5
```

[18]:
```
# Train the model!
test_losses = []
train_losses = []
counter = -5 # we skip the very first few iterations before changing learning␣
 ↪rate
adaptation_threshold = 0.0005 # 0.05 % (paper)
adaptation_indices = []

print("Training the model . . .")
for t in range(epochs):
    # Train
    train_loop(train_dataloader, model, loss_fn, optimizer)
    # Test
    average_test_loss = test_loop(test_dataloader, model, loss_fn)
    test_losses.append(average_test_loss)

    # Also test on the training data
    subset = get_subset_train_dataloader(data_train)
    # test on this subset
    average_train_loss = test_loop(subset, model, loss_fn)
    train_losses.append(average_train_loss)

    # Update the learning rate - see Appendix B of the paper
    # only check if update needed after 10 epochs
    if counter >= 10:
        current_average = np.mean(test_losses[-5:])
        previous_average = np.mean(test_losses[-10:-5])
        # If we did not improve the test loss sufficiently, going to adapt LR
```

```python
        if current_average/previous_average >= 0.05:
            # Reset counter (note: will increment later, so set to -1 st it␣
↪becomes 0)
            counter = -1
            learning_rate = 0.5*learning_rate
            print(f"Adapting learning rate to {learning_rate}")
            # Change optimizer
            optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
            # Add the epoch time for plotting later on
            adaptation_indices.append(t)

    # Another epoch passed - increment counter
    counter += 1

    # Report progress:
    print(f"\n Epoch {t+1} \n --------------")
    print(f"Average loss of: {average_test_loss} for test data")
    print(f"Average loss of: {average_train_loss} for train data")

print("Done!")
```

```
Training the model . . .

 Epoch 1
 --------------
Average loss of: 0.0017823913499501686 for test data
Average loss of: 0.0017260212710457513 for train data

 Epoch 2
 --------------
Average loss of: 0.0004918487218488305 for test data
Average loss of: 0.0004916590779421691 for train data

 Epoch 3
 --------------
Average loss of: 0.00016748606430600021 for test data
Average loss of: 0.00016857311222553492 for train data

 Epoch 4
 --------------
Average loss of: 6.980137053972703e-05 for test data
Average loss of: 6.803461293455216e-05 for train data

 Epoch 5
 --------------
Average loss of: 0.0017871337313466846 for test data
Average loss of: 0.0017906745680765555 for train data
```

```
 Epoch 6
 --------------
Average loss of: 0.00014033970706549417 for test data
Average loss of: 0.00014054515291084968 for train data

 Epoch 7
 --------------
Average loss of: 0.0010092918587753328 for test data
Average loss of: 0.0009967701188987866 for train data

 Epoch 8
 --------------
Average loss of: 4.778926101837347e-05 for test data
Average loss of: 4.679444930933679e-05 for train data

 Epoch 9
 --------------
Average loss of: 0.003728222890426747 for test data
Average loss of: 0.0037496253839149453 for train data

 Epoch 10
 --------------
Average loss of: 0.00043513274912651 for test data
Average loss of: 0.00043189073890483325 for train data

 Epoch 11
 --------------
Average loss of: 0.0010971768069997453 for test data
Average loss of: 0.0010885801900865529 for train data

 Epoch 12
 --------------
Average loss of: 0.00010233953938363477 for test data
Average loss of: 0.00010126772785400109 for train data

 Epoch 13
 --------------
Average loss of: 3.383543650871113e-05 for test data
Average loss of: 3.406924012572893e-05 for train data

 Epoch 14
 --------------
Average loss of: 9.073701172098391e-06 for test data
Average loss of: 8.823581257905326e-06 for train data

 Epoch 15
 --------------
Average loss of: 1.1161160966349443e-05 for test data
```

```
Average loss of: 1.1051952772565613e-05 for train data

 Epoch 16
 --------------
Average loss of: 0.00010027311113133012 for test data
Average loss of: 9.89542253528009e-05 for train data

 Epoch 17
 --------------
Average loss of: 1.7420071030712348e-05 for test data
Average loss of: 1.702395224791357e-05 for train data

 Epoch 18
 --------------
Average loss of: 1.013671858388919e-05 for test data
Average loss of: 9.70077214475668e-06 for train data
Adapting learning rate to 0.0003

 Epoch 19
 --------------
Average loss of: 1.0052255467203845e-05 for test data
Average loss of: 9.654529601455568e-06 for train data

 Epoch 20
 --------------
Average loss of: 8.374545058658508e-06 for test data
Average loss of: 8.199265211530587e-06 for train data

 Epoch 21
 --------------
Average loss of: 1.2378793755964546e-05 for test data
Average loss of: 1.2429277864376867e-05 for train data

 Epoch 22
 --------------
Average loss of: 5.489215645604836e-06 for test data
Average loss of: 5.447202713225787e-06 for train data

 Epoch 23
 --------------
Average loss of: 5.141209373705758e-06 for test data
Average loss of: 5.073710837951075e-06 for train data

 Epoch 24
 --------------
Average loss of: 4.148107721015514e-06 for test data
Average loss of: 4.072896935338013e-06 for train data
```

```
 Epoch 25
 --------------
Average loss of: 5.263829167413936e-06 for test data
Average loss of: 5.160126716763074e-06 for train data

 Epoch 26
 --------------
Average loss of: 3.522602074821343e-06 for test data
Average loss of: 3.4254739040797302e-06 for train data

 Epoch 27
 --------------
Average loss of: 7.905531073136578e-06 for test data
Average loss of: 7.949140630728924e-06 for train data

 Epoch 28
 --------------
Average loss of: 6.235866153518076e-06 for test data
Average loss of: 6.179988064488248e-06 for train data

 Epoch 29
 --------------
Average loss of: 2.9153932721931542e-06 for test data
Average loss of: 2.8753999995620315e-06 for train data
Adapting learning rate to 0.00015

 Epoch 30
 --------------
Average loss of: 7.88133109718371e-06 for test data
Average loss of: 7.816045280677893e-06 for train data

 Epoch 31
 --------------
Average loss of: 4.24673103457818e-06 for test data
Average loss of: 4.179012048059002e-06 for train data

 Epoch 32
 --------------
Average loss of: 2.6462546912762408e-06 for test data
Average loss of: 2.575691082101069e-06 for train data

 Epoch 33
 --------------
Average loss of: 4.111713299668821e-06 for test data
Average loss of: 4.103939334137872e-06 for train data

 Epoch 34
 --------------
```

```
Average loss of: 1.5616023918660074e-06 for test data
Average loss of: 1.4305276898756365e-06 for train data

 Epoch 35
 --------------
Average loss of: 2.921958694335912e-06 for test data
Average loss of: 2.9321225867487448e-06 for train data

 Epoch 36
 --------------
Average loss of: 2.6446220128067166e-06 for test data
Average loss of: 2.635567590969643e-06 for train data

 Epoch 37
 --------------
Average loss of: 4.141676849606109e-06 for test data
Average loss of: 4.150908116949118e-06 for train data

 Epoch 38
 --------------
Average loss of: 1.9740436819381593e-06 for test data
Average loss of: 1.9758612170219024e-06 for train data

 Epoch 39
 --------------
Average loss of: 3.6239909652718464e-06 for test data
Average loss of: 3.6384054204008223e-06 for train data

 Epoch 40
 --------------
Average loss of: 1.1076798044816238e-06 for test data
Average loss of: 1.0783792916186393e-06 for train data
Adapting learning rate to 7.5e-05

 Epoch 41
 --------------
Average loss of: 1.6888753684100356e-06 for test data
Average loss of: 1.6698498466967677e-06 for train data

 Epoch 42
 --------------
Average loss of: 2.6453087931773093e-06 for test data
Average loss of: 2.6791487947740882e-06 for train data

 Epoch 43
 --------------
Average loss of: 3.4960457836999336e-06 for test data
Average loss of: 3.5052631543707024e-06 for train data
```

```
 Epoch 44
 --------------
Average loss of: 2.464926109069303e-06 for test data
Average loss of: 2.450544416089728e-06 for train data

 Epoch 45
 --------------
Average loss of: 2.6518469312054477e-06 for test data
Average loss of: 2.5819985313746337e-06 for train data

 Epoch 46
 --------------
Average loss of: 3.906813404659661e-06 for test data
Average loss of: 3.942644100559486e-06 for train data

 Epoch 47
 --------------
Average loss of: 1.8710323491921273e-06 for test data
Average loss of: 1.813160674290835e-06 for train data

 Epoch 48
 --------------
Average loss of: 2.0092966489331543e-06 for test data
Average loss of: 2.011616749940457e-06 for train data

 Epoch 49
 --------------
Average loss of: 1.8069462299309452e-06 for test data
Average loss of: 1.796225814659077e-06 for train data

 Epoch 50
 --------------
Average loss of: 2.2076530924165743e-06 for test data
Average loss of: 2.1852336277761104e-06 for train data

 Epoch 51
 --------------
Average loss of: 2.1473438889011153e-06 for test data
Average loss of: 2.1259425527289435e-06 for train data
Adapting learning rate to 3.75e-05

 Epoch 52
 --------------
Average loss of: 2.0926544916488845e-06 for test data
Average loss of: 2.0251650464826188e-06 for train data

 Epoch 53
```

```
 --------------
Average loss of: 2.713061472043943e-06 for test data
Average loss of: 2.7216986704509135e-06 for train data

 Epoch 54
 --------------
Average loss of: 2.6419167255381108e-06 for test data
Average loss of: 2.6403614972256302e-06 for train data

 Epoch 55
 --------------
Average loss of: 2.7357749519133256e-06 for test data
Average loss of: 2.743939059071479e-06 for train data

 Epoch 56
 --------------
Average loss of: 2.7603601431848347e-06 for test data
Average loss of: 2.754117815765931e-06 for train data

 Epoch 57
 --------------
Average loss of: 1.4894148708204018e-06 for test data
Average loss of: 1.4259529771094186e-06 for train data

 Epoch 58
 --------------
Average loss of: 9.381144171713615e-07 for test data
Average loss of: 9.058513226594363e-07 for train data

 Epoch 59
 --------------
Average loss of: 1.1697984415315758e-06 for test data
Average loss of: 1.1630199945904487e-06 for train data

 Epoch 60
 --------------
Average loss of: 2.636883913112161e-06 for test data
Average loss of: 2.6388132946144345e-06 for train data

 Epoch 61
 --------------
Average loss of: 8.593646336053664e-07 for test data
Average loss of: 8.434105359755719e-07 for train data

 Epoch 62
 --------------
Average loss of: 2.7426770142215964e-06 for test data
Average loss of: 2.774399296632704e-06 for train data
```

```
Adapting learning rate to 1.875e-05

 Epoch 63
 --------------
Average loss of: 1.0521505595179608e-06 for test data
Average loss of: 1.0382492573539051e-06 for train data

 Epoch 64
 --------------
Average loss of: 1.6617349013321715e-07 for test data
Average loss of: 1.619047445830639e-07 for train data

 Epoch 65
 --------------
Average loss of: 1.6254727866296412e-07 for test data
Average loss of: 1.5548895876777393e-07 for train data

 Epoch 66
 --------------
Average loss of: 1.583920994420305e-07 for test data
Average loss of: 1.4738991364702552e-07 for train data

 Epoch 67
 --------------
Average loss of: 1.54470954303945e-07 for test data
Average loss of: 1.4565095642883143e-07 for train data

 Epoch 68
 --------------
Average loss of: 1.5080221739578183e-07 for test data
Average loss of: 1.5114684005491002e-07 for train data

 Epoch 69
 --------------
Average loss of: 1.4854311697041693e-07 for test data
Average loss of: 1.4063483666078695e-07 for train data

 Epoch 70
 --------------
Average loss of: 1.4985146002447638e-07 for test data
Average loss of: 1.5024343019822496e-07 for train data

 Epoch 71
 --------------
Average loss of: 1.5591955552020643e-07 for test data
Average loss of: 1.4740863653891698e-07 for train data

 Epoch 72
```

```
 --------------
Average loss of: 1.5990364150997055e-07 for test data
Average loss of: 1.5477942037168526e-07 for train data

 Epoch 73
 --------------
Average loss of: 1.6269609886513208e-07 for test data
Average loss of: 1.540805494694639e-07 for train data
Adapting learning rate to 9.375e-06

 Epoch 74
 --------------
Average loss of: 1.6322777613319418e-07 for test data
Average loss of: 1.5965061597050784e-07 for train data

 Epoch 75
 --------------
Average loss of: 9.422505023723553e-08 for test data
Average loss of: 9.148482027026743e-08 for train data

 Epoch 76
 --------------
Average loss of: 9.287588048722953e-08 for test data
Average loss of: 1.0558500117811261e-07 for train data

 Epoch 77
 --------------
Average loss of: 9.144670154803124e-08 for test data
Average loss of: 8.615748381685762e-08 for train data

 Epoch 78
 --------------
Average loss of: 9.014431044713714e-08 for test data
Average loss of: 7.97972905019968e-08 for train data

 Epoch 79
 --------------
Average loss of: 8.894785228988123e-08 for test data
Average loss of: 8.56435939078856e-08 for train data

 Epoch 80
 --------------
Average loss of: 8.782052637675057e-08 for test data
Average loss of: 7.94265366803171e-08 for train data

 Epoch 81
 --------------
Average loss of: 8.677367845211048e-08 for test data
```

```
Average loss of: 7.672331003104686e-08 for train data

 Epoch 82
 --------------
Average loss of: 8.57994585964535e-08 for test data
Average loss of: 7.825783039273108e-08 for train data

 Epoch 83
 --------------
Average loss of: 8.487612235193012e-08 for test data
Average loss of: 9.672973851084766e-08 for train data

 Epoch 84
 --------------
Average loss of: 8.407245267302331e-08 for test data
Average loss of: 7.617923517700953e-08 for train data
Adapting learning rate to 4.6875e-06

 Epoch 85
 --------------
Average loss of: 8.323489942440007e-08 for test data
Average loss of: 8.372315954611971e-08 for train data

 Epoch 86
 --------------
Average loss of: 1.549247231375997e-07 for test data
Average loss of: 1.5142827758775555e-07 for train data

 Epoch 87
 --------------
Average loss of: 1.5179634415042924e-07 for test data
Average loss of: 1.4680651595052638e-07 for train data

 Epoch 88
 --------------
Average loss of: 1.498851235756562e-07 for test data
Average loss of: 1.4200734175793707e-07 for train data

 Epoch 89
 --------------
Average loss of: 1.4797051439306498e-07 for test data
Average loss of: 1.4385175560642492e-07 for train data

 Epoch 90
 --------------
Average loss of: 1.45942886396429e-07 for test data
Average loss of: 1.3684024449598502e-07 for train data
```

```
 Epoch 91
 --------------
Average loss of: 1.4386466575931485e-07 for test data
Average loss of: 1.3683084256272398e-07 for train data

 Epoch 92
 --------------
Average loss of: 1.4193656847790687e-07 for test data
Average loss of: 1.429411714133575e-07 for train data

 Epoch 93
 --------------
Average loss of: 1.3997328777624642e-07 for test data
Average loss of: 1.3591397828422371e-07 for train data

 Epoch 94
 --------------
Average loss of: 1.380518514584189e-07 for test data
Average loss of: 1.330039069466112e-07 for train data

 Epoch 95
 --------------
Average loss of: 1.3591591940525403e-07 for test data
Average loss of: 1.2512529931024535e-07 for train data
Adapting learning rate to 2.34375e-06

 Epoch 96
 --------------
Average loss of: 1.3420203670673033e-07 for test data
Average loss of: 1.2480804601951466e-07 for train data

 Epoch 97
 --------------
Average loss of: 7.771917037376326e-08 for test data
Average loss of: 7.679740332768518e-08 for train data

 Epoch 98
 --------------
Average loss of: 7.715280965328293e-08 for test data
Average loss of: 7.793844612077288e-08 for train data

 Epoch 99
 --------------
Average loss of: 7.683872578042201e-08 for test data
Average loss of: 7.340215814364232e-08 for train data

 Epoch 100
 --------------
```

```
Average loss of: 7.663336409465386e-08 for test data
Average loss of: 6.966425524624923e-08 for train data

 Epoch 101
 --------------
Average loss of: 7.634261311722412e-08 for test data
Average loss of: 8.452858307377894e-08 for train data

 Epoch 102
 --------------
Average loss of: 7.610083760316677e-08 for test data
Average loss of: 7.336873950269585e-08 for train data

 Epoch 103
 --------------
Average loss of: 7.586449169120926e-08 for test data
Average loss of: 7.487711635245961e-08 for train data

 Epoch 104
 --------------
Average loss of: 7.565269290822448e-08 for test data
Average loss of: 7.12564616555654e-08 for train data

 Epoch 105
 --------------
Average loss of: 7.541054304257019e-08 for test data
Average loss of: 7.695784856518579e-08 for train data

 Epoch 106
 --------------
Average loss of: 7.51300265138567e-08 for test data
Average loss of: 7.124799089650495e-08 for train data
Adapting learning rate to 1.171875e-06

 Epoch 107
 --------------
Average loss of: 7.492553573101236e-08 for test data
Average loss of: 6.874477850868906e-08 for train data

 Epoch 108
 --------------
Average loss of: 6.163545001040671e-08 for test data
Average loss of: 5.933509895273888e-08 for train data

 Epoch 109
 --------------
Average loss of: 6.146454257594714e-08 for test data
Average loss of: 6.209007592772356e-08 for train data
```

```
 Epoch 110
 --------------
Average loss of: 6.126443677820057e-08 for test data
Average loss of: 6.768837962409705e-08 for train data

 Epoch 111
 --------------
Average loss of: 6.106516586249326e-08 for test data
Average loss of: 5.514449438650693e-08 for train data

 Epoch 112
 --------------
Average loss of: 6.08722306497812e-08 for test data
Average loss of: 5.191489921658147e-08 for train data

 Epoch 113
 --------------
Average loss of: 6.067255266459459e-08 for test data
Average loss of: 5.859620533481149e-08 for train data

 Epoch 114
 --------------
Average loss of: 6.048404754395846e-08 for test data
Average loss of: 6.384966706860315e-08 for train data

 Epoch 115
 --------------
Average loss of: 6.028839948816168e-08 for test data
Average loss of: 5.5383320569334204e-08 for train data

 Epoch 116
 --------------
Average loss of: 6.009513993993201e-08 for test data
Average loss of: 6.275539051850579e-08 for train data

 Epoch 117
 --------------
Average loss of: 5.98883693705045e-08 for test data
Average loss of: 5.6670732373759845e-08 for train data
Adapting learning rate to 5.859375e-07

 Epoch 118
 --------------
Average loss of: 5.971109745379877e-08 for test data
Average loss of: 5.524138541277055e-08 for train data

 Epoch 119
```

```
 --------------
Average loss of: 6.134069216784973e-08 for test data
Average loss of: 5.5001194931138e-08 for train data

 Epoch 120
 --------------
Average loss of: 6.125862537856075e-08 for test data
Average loss of: 5.807539750912075e-08 for train data

 Epoch 121
 --------------
Average loss of: 6.115835866137404e-08 for test data
Average loss of: 5.3733612058960745e-08 for train data

 Epoch 122
 --------------
Average loss of: 6.109268607366529e-08 for test data
Average loss of: 5.397005337773878e-08 for train data

 Epoch 123
 --------------
Average loss of: 6.099362266514181e-08 for test data
Average loss of: 5.963830268536057e-08 for train data

 Epoch 124
 --------------
Average loss of: 6.091591657083152e-08 for test data
Average loss of: 5.3983016611349225e-08 for train data

 Epoch 125
 --------------
Average loss of: 6.08261716976018e-08 for test data
Average loss of: 6.010159674165646e-08 for train data

 Epoch 126
 --------------
Average loss of: 6.073820716524503e-08 for test data
Average loss of: 5.435300137834744e-08 for train data

 Epoch 127
 --------------
Average loss of: 6.065200877425702e-08 for test data
Average loss of: 5.939154980241344e-08 for train data

 Epoch 128
 --------------
Average loss of: 6.055743566665836e-08 for test data
Average loss of: 5.5611680451988466e-08 for train data
```

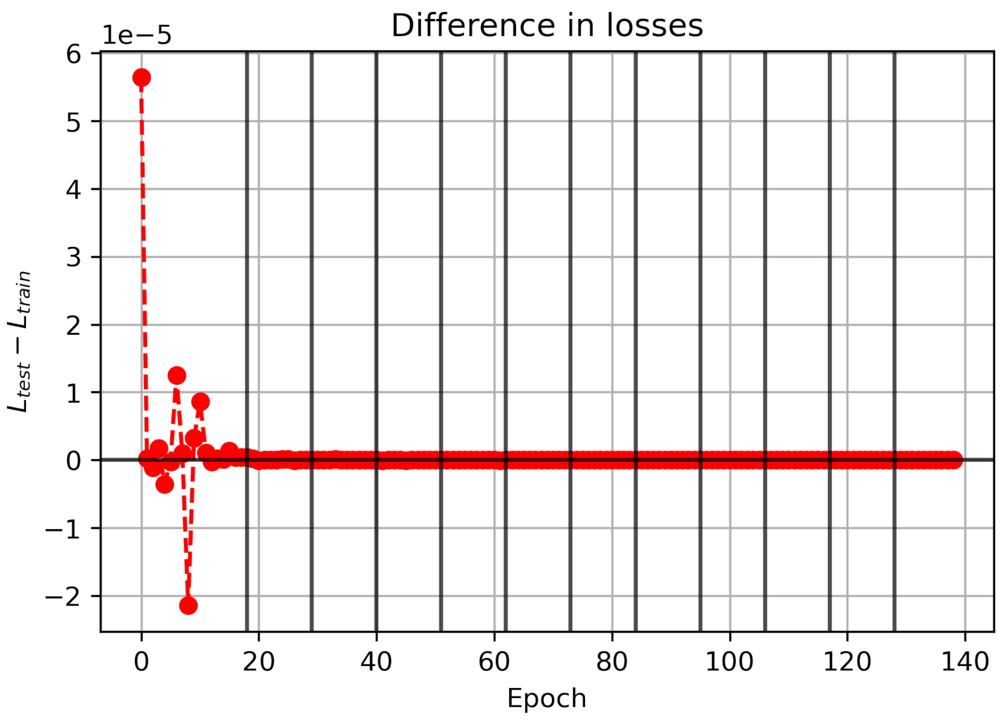
```
Adapting learning rate to 2.9296875e-07

 Epoch 129
 --------------
Average loss of: 6.047857805612103e-08 for test data
Average loss of: 6.383471852637659e-08 for train data

 Epoch 130
 --------------
Average loss of: 5.3099046744218374e-08 for test data
Average loss of: 5.101478875260433e-08 for train data

 Epoch 131
 --------------
Average loss of: 5.3044042711650826e-08 for test data
Average loss of: 5.031307970887555e-08 for train data

 Epoch 132
 --------------
Average loss of: 5.3015164393383774e-08 for test data
Average loss of: 4.6317751031819105e-08 for train data

 Epoch 133
 --------------
Average loss of: 5.297976562533491e-08 for test data
Average loss of: 5.1795664146769556e-08 for train data

 Epoch 134
 --------------
Average loss of: 5.2942649727475316e-08 for test data
Average loss of: 5.144164699464809e-08 for train data

 Epoch 135
 --------------
Average loss of: 5.2905538034984475e-08 for test data
Average loss of: 4.695262778220978e-08 for train data

 Epoch 136
 --------------
Average loss of: 5.286636715899899e-08 for test data
Average loss of: 5.115338599846048e-08 for train data

 Epoch 137
 --------------
Average loss of: 5.2829917565212023e-08 for test data
Average loss of: 4.363461437114951e-08 for train data

 Epoch 138
```

```
 --------------
Average loss of: 5.279018041731254e-08 for test data
Average loss of: 4.9564207982441874e-08 for train data

 Epoch 139
 --------------
Average loss of: 5.2751628360486974e-08 for test data
Average loss of: 5.5737008953558884e-08 for train data
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [18], in <cell line: 9>()
      8 print("Training the model . . .")
      9 for t in range(epochs):
     10     # Train
---> 11     train_loop(train_dataloader, model, loss_fn, optimizer)
     12     # Test
     13     average_test_loss = test_loop(test_dataloader, model, loss_fn)

Input In [15], in train_loop(dataloader, model, loss_fn, optimizer,␣
 ↪report_progress)
      7 loss = loss_fn(pred, y)
      9 # Backpropagation
---> 10 optimizer.zero_grad()
     11 loss.backward()
     12 optimizer.step()

File D:\Anaconda3\lib\site-packages\torch\optim\optimizer.py:267, in Optimizer.
 ↪zero_grad(self, set_to_none)
    265 if foreach:
    266     per_device_and_dtype_grads = defaultdict(lambda: defaultdict(list))
--> 267 with torch.autograd.profiler.record_function(self.
 ↪_zero_grad_profile_name):
    268     for group in self.param_groups:
    269         for p in group['params']:

File D:\Anaconda3\lib\site-packages\torch\autograd\profiler.py:488, in␣
 ↪record_function.__enter__(self)
    487 def __enter__(self):
--> 488     self.handle = torch.ops.profiler._record_function_enter(self.name,␣
 ↪self.args)
    489     return self

KeyboardInterrupt:
```

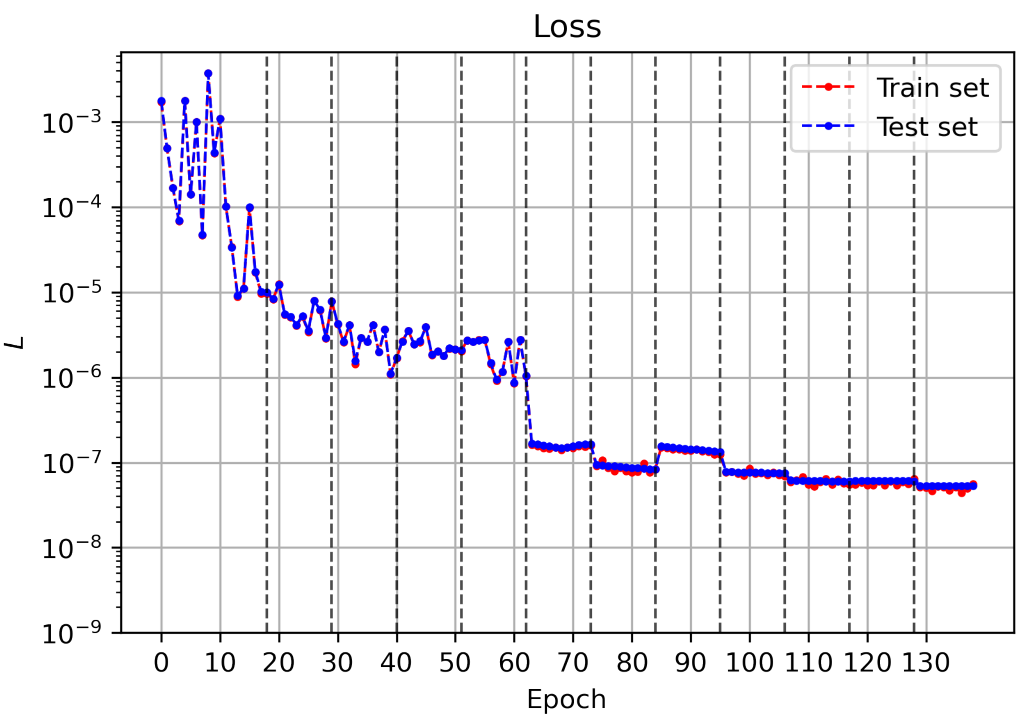
5.1 Results of training

[20]:
```
# Get the difference (need np.array)
test_losses_as_array = np.array(test_losses)
train_losses_as_array = np.array(train_losses)
difference = test_losses_as_array - train_losses_as_array

# Plot it
plt.figure()
plt.plot(difference, 'o--', color = 'red', label = "Difference")
plt.grid()
plt.xlabel("Epoch")
plt.ylabel(r'$L_{test} - L_{train}$')
plt.axhline(0, color = 'black', alpha = 0.7)
plt.title("Difference in losses")
# Plot when we adapted learning rate
for t in adaptation_indices:
    plt.axvline(t, color = 'black', alpha = 0.7)
plt.show()
```



[21]:
```
# Plot it
plt.figure()
lw = 1
ms = 2
```

```python
plt.plot(train_losses, 'o--', color = 'red', label = "Train set", lw = lw, ms =␣
 ↪ms)
plt.plot(test_losses, 'o--', color = 'blue', label = "Test set", lw = lw, ms =␣
 ↪ms)
plt.legend()
plt.grid()
plt.xlabel("Epoch")
xt_step = 10
xt = [i*xt_step for i in range(len(train_losses)//xt_step+1)]
plt.xticks(xt)
plt.ylabel(r'$L$')
plt.axhline(0, color = 'black', alpha = 0.7)
plt.title("Loss")
# Plot when we adapted learning rate
for t in adaptation_indices:
    plt.axvline(t, linestyle = "--", color = 'black', alpha = 0.7, lw = 1)
plt.yscale('log')
plt.ylim(10**(-9))
plt.show()
```



5.2 Save the neural network

[ ]:
```
torch.save(model, 'NNEOSBv0.pth')
```

[ ]:
```

```