Export_models

December 20, 2022

Contents

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300
import random
import csv
import h5py
import pandas as pd
import torch
import torchvision
from torch import nn
from torch.utils.data import Dataset, DataLoader
import matplotlib.cm as cm
import pickle
```

1 Export the NNC2P model

This is the model architecture:

[2]:
```
# Define hyperparameters of the model here. Will first of all put two hidden␣
↪layers
# total of 800 neurons for the one in the paper
device = "cpu"
size_HL_1 = 600
size_HL_2 = 200

# Implement neural network
class NeuralNetwork(nn.Module):
    def __init__(self):
```

```
        super(NeuralNetwork, self).__init__()
        #self.flatten = nn.Flatten()
        self.stack = nn.Sequential(
            nn.Linear(3, size_HL_1),
            nn.Sigmoid(),
            nn.Linear(size_HL_1, size_HL_2),
            nn.Sigmoid(),
            nn.Linear(size_HL_2, 1)
        )

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

We import NNC2Pv0, which was on par with the models in the paper.

[3]:
```
NNC2P = torch.load('Models/NNC2Pv0t2.pth')
model = NNC2P
```

In case we want to view the variables, uncomment the following:

[4]:
```
# NNC2P.state_dict()
```

1.1 Save the matrices as a CSV

Note that converting from Torch tensor to Numpy array does **not** cause loss of information:

[29]:
```
test_exact = NNC2P.state_dict()["stack.0.weight"]
test_exact_value = test_exact[0][0].item()
print('%.25f' % test_exact_value)
print("---")
test_exact_np = test_exact.numpy()
test_exact_value_np = test_exact_np[0][0]
print('%.25f' % test_exact_value_np)
```

```
-0.3647063672542572021484375
---
-0.3647063672542572021484375
```

1.1.1 Saving and loading as CSV

Save the values: (refresher on Pickle)

[57]:
```
# State dict contains all the variables
state_dict = NNC2P.state_dict().items()
# Names to save the files:
```

```
file_names           = ["weight0", "bias0", "weight2", "bias2", "weight4",␣
↪"bias4"]
save_names          = ["Models/paramvals/" + name + ".csv" for name in␣
↪file_names]
flat_save_names = ["Models/paramvals/" + name + "_flat.csv" for name in␣
↪file_names]
no_comma_flat_save_names = ["Models/paramvals/" + name + "_flat_no_comma.csv"␣
↪for name in file_names]

# Save each one:
counter = 0
for param_name, item in state_dict:
    # Get appropriate names
    name                      = file_names[counter]
    save_name          = save_names[counter]
    flat_save_name = flat_save_names[counter]
    no_comma_flat_save_name = no_comma_flat_save_names[counter]
    # Get the matrix and flatten it as well
    matrix_np    = item.numpy()
    flat_matrix_np    = matrix_np.flatten()
    # The following save txt is only important for stuff done within this␣
↪noteboo!
    np.savetxt(no_comma_flat_save_name, flat_matrix_np, delimiter=",", fmt="%0.
↪35f")

    np.savetxt(save_name, matrix_np, delimiter=",", fmt="%0.35f")
    # Note: due to weird Fortran stuff, have to append a 0 at the start of the␣
↪file
    flat_matrix_np    = np.insert(flat_matrix_np, 0, 0)
    np.savetxt(flat_save_name, flat_matrix_np, delimiter=",", newline=',\n',␣
↪fmt="%0.35f")



    counter += 1
```

Read the files:

[96]:
```
weight0 = np.loadtxt('Models/paramvals/weight0.csv', delimiter=",")
bias0      = np.loadtxt('Models/paramvals/bias0.csv', delimiter=",")
s = np.shape(bias0)[0]
bias0 = np.reshape(bias0, (s, 1))
weight2 = np.loadtxt('Models/paramvals/weight2.csv', delimiter=",")
bias2      = np.loadtxt('Models/paramvals/bias2.csv', delimiter=",")
s = np.shape(bias2)[0]
bias2 = np.reshape(bias2, (s, 1))
weight4 = np.loadtxt('Models/paramvals/weight4.csv', delimiter=",")
s = np.shape(weight4)[0]
```

```
weight4 = np.reshape(weight4, (1, s))
bias4      = np.loadtxt('Models/paramvals/bias4.csv', delimiter=",")
bias4 = np.reshape(bias4, (1, 1))

weights_and_biases = [weight0, bias0, weight2, bias2, weight4, bias4]
```

Same for flat: **NOTE** for numpy (here), we load "no_comma" files since otherwise there's an error. For Fortran, we use the files **WITHOUT** "no comma".

[97]:
```
# weight0_flat = np.loadtxt('Models/paramvals/weight0_flat_no_comma.csv',␣
 ↪delimiter=",")
# bias0_flat      = np.loadtxt('Models/paramvals/bias0_flat_no_comma.csv',␣
 ↪delimiter=",")
# weight2_flat = np.loadtxt('Models/paramvals/weight2_flat_no_comma.csv',␣
 ↪delimiter=",")
# bias2_flat      = np.loadtxt('Models/paramvals/bias2_flat_no_comma.csv',␣
 ↪delimiter=",")
# weight4_flat = np.loadtxt('Models/paramvals/weight4_flat_no_comma.csv',␣
 ↪delimiter=",")
# bias4_flat      = np.loadtxt('Models/paramvals/bias4_flat_no_comma.csv',␣
 ↪delimiter=",")

# weights_and_biases_flat = [weight0_flat, bias0_flat, weight2_flat,␣
 ↪bias2_flat, weight4_flat, bias4_flat]
```

[98]:
```
print('%.25f' % weight0[0][0])
```

```
-0.3647063672542572021484375
```

[99]:
```
np.shape(weight0_flat)
```

[99]:
```
(1800,)
```

(Below: old pickle version)

[100]:
```
#   reload pickled data from file
# test_name = save_names[0]
# with open(test_name, 'rb') as f:
#     test_load = pickle.load(f)
```

[101]:
```
# # Save the loaded versions in the appropriate variables
# with open('Models/paramvals/weight0.csv', 'rb') as f:
#     weight0 = pickle.load(f)

# with open('Models/paramvals/bias0.csv', 'rb') as f:
#     bias0 = pickle.load(f)
#     s = np.shape(bias0)[0]
#     bias0 = np.reshape(bias0, (s, 1))
```

```
# with open('Models/paramvals/weight2.csv', 'rb') as f:
#     weight2 = pickle.load(f)

# with open('Models/paramvals/bias2.csv', 'rb') as f:
#     bias2 = pickle.load(f)
#     s = np.shape(bias2)[0]
#     bias2 = np.reshape(bias2, (s, 1))

# with open('Models/paramvals/weight4.csv', 'rb') as f:
#     weight4 = pickle.load(f)

# with open('Models/paramvals/bias4.csv', 'rb') as f:
#     bias4 = pickle.load(f)
#     s = np.shape(bias4)[0]
#     bias4 = np.reshape(bias4, (s, 1))

# # Gather together in a list of all variables
# weights_and_biases = [weight0, bias0, weight2, bias2, weight4, bias4]
```

Same for flattened arrays:

[102]:
```
# with open('Models/paramvals/bias0_flat.csv', 'r') as file:
#     csvreader = csv.reader(file)
# #     for row in csvreader:
# #         print(row)
```

[103]:
```
# # Save the loaded versions in the appropriate variables
# with open('Models/paramvals/weight0_flat.csv', 'rb') as f:
#     weight0_flat = pickle.load(f)

# # with open('Models/paramvals/bias0_flat.csv', 'rb') as f:
# #     bias0_flat = pickle.load(f)


# with open('Models/paramvals/weight2_flat.csv', 'rb') as f:
#     weight2_flat = pickle.load(f)

# with open('Models/paramvals/bias2_flat.csv', 'rb') as f:
#     bias2_flat = pickle.load(f)

# with open('Models/paramvals/weight4_flat.csv', 'rb') as f:
#     weight4_flat = pickle.load(f)

# with open('Models/paramvals/bias4_flat.csv', 'rb') as f:
#     bias4_flat = pickle.load(f)

# # Gather together in a list of all variables
```

```
# weights_and_biases_flat = [weight0_flat, bias0_flat, weight2_flat,␣
↪bias2_flat, weight4_flat, bias4_flat]
```

[104]:
```
# Print the shape for each parameter:
for i in range(len(weights_and_biases)):
    print("For the file: ", file_names[i])
    # Read the values
    shape = np.shape(weights_and_biases[i])
    print("The shape is equal to ", shape)
```

```
For the file:  weight0
The shape is equal to  (600, 3)
For the file:  bias0
The shape is equal to  (600, 1)
For the file:  weight2
The shape is equal to  (200, 600)
For the file:  bias2
The shape is equal to  (200, 1)
For the file:  weight4
The shape is equal to  (1, 200)
For the file:  bias4
The shape is equal to  (1, 1)
```

[105]:
```
# Same for their flattened versions:
for i in range(len(weights_and_biases_flat)):
    print("For the file: ", flat_save_names[i])
    # Read the values
    shape = np.shape(weights_and_biases_flat[i])
    print("The shape is equal to ", shape)
```

```
For the file:  Models/paramvals/weight0_flat.csv
The shape is equal to  (1800,)
For the file:  Models/paramvals/bias0_flat.csv
The shape is equal to  (600,)
For the file:  Models/paramvals/weight2_flat.csv
The shape is equal to  (120000,)
For the file:  Models/paramvals/bias2_flat.csv
The shape is equal to  (200,)
For the file:  Models/paramvals/weight4_flat.csv
The shape is equal to  (200,)
For the file:  Models/paramvals/bias4_flat.csv
The shape is equal to  ()
```

Play around with some examples

[88]:
```
# # Read the example file
# print(example)
# print(np.shape(example))
```

[89]:
```
# test_load_value = test_load[0][0]
# print('%.25f' % test_load_value)
```

1.1.2 Predicting using the values in the arrays

When we are going to implement this in the Gmunu code, we can no longer use any of the built-in tools of PyTorch.

[106]:
```
## One specific test case for the data
rho,eps,v,p,D,S,tau = 9.83632270803203,1.962038705851822,0.2660655147967911,12.
↪866163917605371,10.204131145455385,12.026584842282125,22.131296926293793
```

This is how the PyTorch implementation works:

[107]:
```
input_test = torch.tensor([D, S, tau])
exact_result = p
print("Exact:")
print(exact_result)
# print(input_test)
with torch.no_grad():
    pred = model(input_test).item()

print("Pytorch prediction:")
print(pred)
```

```
Exact:
12.866163917605371
Pytorch prediction:
12.866371154785156
```

Now, we have to try and get the same output, but by defining all intermediate steps ourselves!

[108]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))

def compute_prediction(x):
    """Input is a np. array of size 1x3"""
    x = np.matmul(weight0, x) + bias0
    x = sigmoid(x)
    x = np.matmul(weight2, x) + bias2
    x = sigmoid(x)
    x = np.matmul(weight4, x) + bias4
    return x[0][0]
```

[109]:
```
input_test = np.array([[D, S, tau]])
print(np.shape(input_test))
input_test = np.transpose(input_test)
print(np.shape(input_test))
```

```
(1, 3)
(3, 1)
```

[110]:
```
our_prediction = compute_prediction(input_test)
print(our_prediction)
print(pred)
```

```
12.866371869133928
12.866371154785156
```

Now we compute rho and eps from this (see appendix A of central paper)

[111]:
```
v_star = S/(tau + D + our_prediction)
W_star = 1/np.sqrt(1-v_star**2)

rho_star = D/W_star
eps_star = (tau + D*(1 - W_star) + our_prediction*(1 - W_star**2))/(D*W_star)
print("Our calculations:")
print(rho_star, eps_star)
print("Exact results:")
print(rho, eps)
```

```
Our calculations:
9.836326155512264 1.9620391642983483
Exact results:
9.83632270803203 1.962038705851822
```

1.1.3 Now save it into an hdf5 file

[112]:
```
# # Open an HDF5 file for writing
# with h5py.File("NNC2Pv0_params.h5", "w") as f:
#     # Save the weights and biases of the network to the HDF5 file
#     f.create_dataset("NNC2Pv0_params", data=NNC2P.state_dict())
```

1.2 More advanced: using Torch script

There exist two ways of converting a PyTorch model to Torch Script. The first is known as tracing, a mechanism in which the structure of the model is captured by evaluating it once using example inputs, and recording the flow of those inputs through the model. This is suitable for models that make limited use of control flow. The second approach is to add explicit annotations to your model that inform the Torch Script compiler that it may directly parse and compile your model code, subject to the constraints imposed by the Torch Script language.

[10]:
```
example = torch.tensor([1, 1, 0.5])
example
```

[10]:
```
tensor([1.0000, 1.0000, 0.5000])
```

[12]:
```
traced_script_module = torch.jit.trace(model, example)
traced_script_module
```

[12]:
```
NeuralNetwork(
  original_name=NeuralNetwork
  (stack): Sequential(
    original_name=Sequential
    (0): Linear(original_name=Linear)
    (1): Sigmoid(original_name=Sigmoid)
    (2): Linear(original_name=Linear)
    (3): Sigmoid(original_name=Sigmoid)
    (4): Linear(original_name=Linear)
  )
)
```

[16]:
```
output = traced_script_module(torch.tensor([1,1,0.5]))
output
```

[16]:
```
tensor([0.0595], grad_fn=<AddBackward0>)
```

[17]:
```
traced_script_module.save("traced_NNC2P_model.pt")
```

To do: finish it

[ ]:
```
```