0724-small_hybrid_model_v5

July 25, 2023

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
```

[2]:
```
import numpy as np
```

[3]:
```
import sys
sys.path.append('..')

from my_code import functions as f
```

1 Data

[4]:
```
file_path = '../data/energies/Trial/Strings_Energies_4_aa.txt'  # Replace with␣
 ↪the actual path to your 'data.txt' file
string_list, number_list = f.read_data_file(file_path)
score_list = np.array(number_list)/100
angles_list = np.array([f.string_to_numbers(string) for string in string_list])
```

[5]:
```
X, Y, X_validation, Y_validation = f.create_validating_set(angles_list,␣
 ↪score_list, percentage=0.1)
```

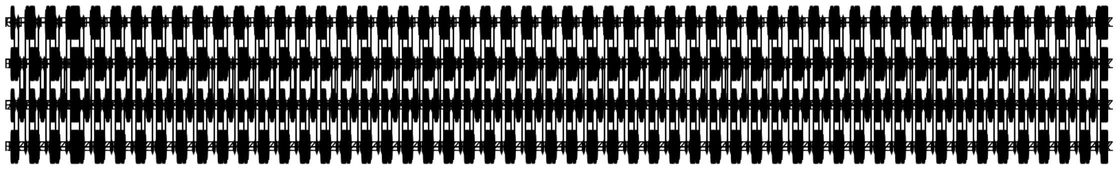
[6]:
```
# Define the dataset
input_data = torch.tensor(X, dtype=torch.float32)
target_data = torch.tensor(Y, dtype=torch.float32).view(-1, 1)

# Define the validation set
input_validation = torch.tensor(X_validation, dtype=torch.float32)
target_validation = torch.tensor(Y_validation, dtype=torch.float32).view(-1, 1)
```

2 Quantum node

[7]:
```
def qml_RZZ(params, wires):
    """
    RZZ gate.
    """
    qml.CNOT(wires=wires)
    qml.RZ(params, wires=wires[1])
    qml.CNOT(wires=wires)
```

[8]:
```
import pennylane as qml

n_qubits = 4
n_layers_block = 50
n_layers_embedding = 3
n_layers = n_layers_block + n_layers_embedding
n_params = 5
dev = qml.device("default.qubit", wires=n_qubits)

@qml.qnode(dev)
def qnode(inputs, weights):

    # state preparation (we create an embedding with 3 layers, paper: 2001.
 ↪03622)
    for i in range(n_layers_embedding):

        # angle embedding for each qubit
        qml.AngleEmbedding(inputs, wires=range(n_qubits))

        # ZZ rotation for neighboring qubits
        for x in range(2):
            for j in range(x,n_qubits,2):
                qml_RZZ(weights[i,j,0], wires=[j, (j+1)%n_qubits])

        # rotations for each qubit
        for j in range(n_qubits):
            qml.RY(weights[i,j,1], wires=j)

        # last angle embedding
    qml.AngleEmbedding(inputs, wires=range(n_qubits))

    #layers
    for i in range(n_layers_embedding, n_layers):
        # rotations for each qubit
        for j in range(n_qubits):
            qml.RX(weights[i,j,2], wires=j)
            qml.RZ(weights[i,j,3], wires=j)

        # ZZ rotation for neighboring qubits
        for x in range(2):
            for j in range(x,n_qubits,2):
                qml_RZZ(weights[i,j,4], wires=[j, (j+1)%n_qubits])
    # rotations for each qubit
    for j in range(n_qubits):
        qml.RX(weights[i,j,0], wires=j)
        qml.RZ(weights[i,j,1], wires=j)

    # measurement
    return [qml.expval(qml.PauliZ(wires=i)) for i in range(n_qubits)]
```

[9]:
```
qml.drawer.use_style("black_white")
fig, ax = qml.draw_mpl(qnode, expansion_strategy="device")([i for i in␣
 ↪range(n_qubits)], np.zeros((n_layers, n_qubits, n_params)))
fig.set_size_inches((16,3))
```



[10]:
```
weight_shapes = {"weights": (n_layers, n_qubits, n_params)}
```

[11]:
```
qlayer = qml.qnn.TorchLayer(qnode, weight_shapes)
```

3 Hybrid model

[12]:
```
input_dim = input_data.size(1)

layers =  [nn.Linear(input_dim*1, input_dim*2), nn.ReLU()]
layers += [nn.Linear(input_dim*2, input_dim*3), nn.ReLU()]
layers += [nn.Linear(input_dim*3, input_dim*3), nn.ReLU()]
layers += [nn.Linear(input_dim*3, input_dim*2), nn.ReLU()]
layers += [nn.Linear(input_dim*2, input_dim*1)]
layers += [qlayer]
layers += [nn.Linear(input_dim*1, input_dim*1)]
layers += [nn.Linear(input_dim*1, input_dim*1), nn.ReLU()] * 2
layers += [nn.Linear(input_dim*1, 1           )]
Net = nn.Sequential(*layers)
```

[13]:
```
# Create an instance of the network
model = Net
```

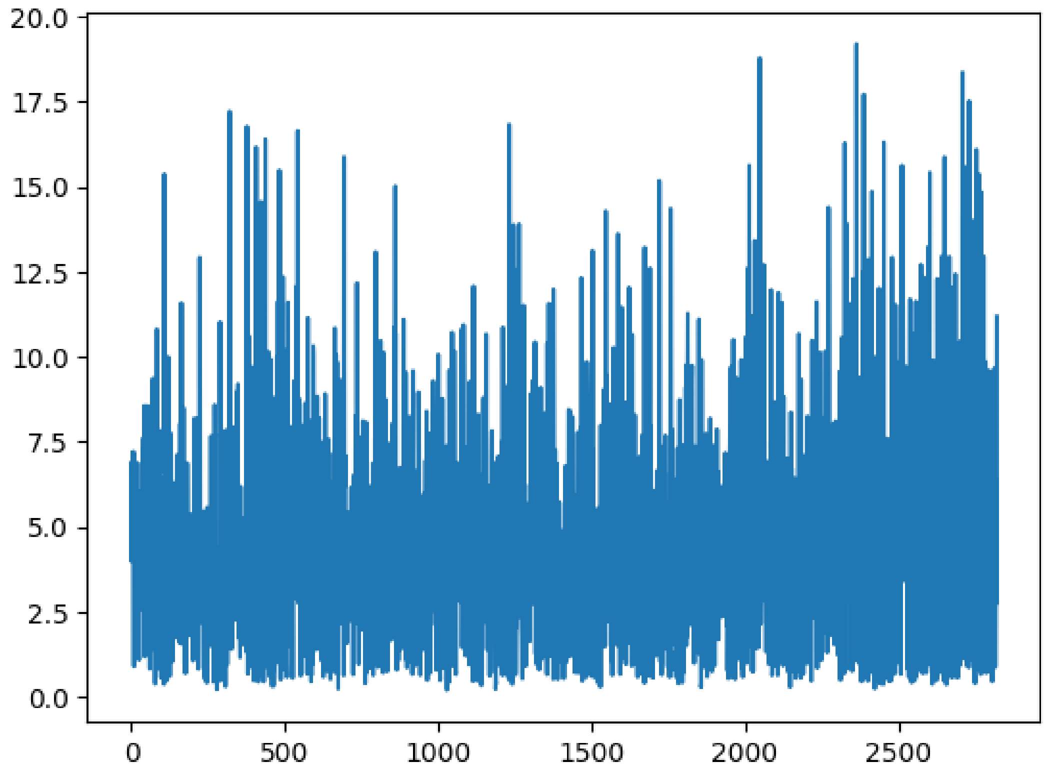
[14]:
```
import time
```

```
[23]: # time
start_time = time.time()


# Define the loss function and optimizer
criterion = nn.MSELoss()  # Mean Squared Error loss
# optimizer = optim.Adam(model.parameters(), lr=0.001)  # Adam optimizer with␣
 ↪learning rate 0.001
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Training loop
num_epochs = 10
batch_size = 32

losses = []
losses_epochs = []

for epoch in range(num_epochs):
    # Shuffle the dataset
    indices = torch.randperm(input_data.size(0))
    input_data = input_data[indices]
    target_data = target_data[indices]

    losses_epochs.append(0)

    # Mini-batch training
    for i in range(0, input_data.size(0), batch_size):
        inputs = input_data[i:i+batch_size]
        targets = target_data[i:i+batch_size]

        # Forward pass
        outputs = model(inputs)

        # Compute the loss
        loss = criterion(outputs, targets)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Store the loss
        losses.append(loss.item())

        print('- Epoch [{}/{}], i: [{}/{}], Loss: {:.4f}'.format(epoch+1,␣
 ↪num_epochs, i, input_data.size(0), loss.item()), end='\r')
```

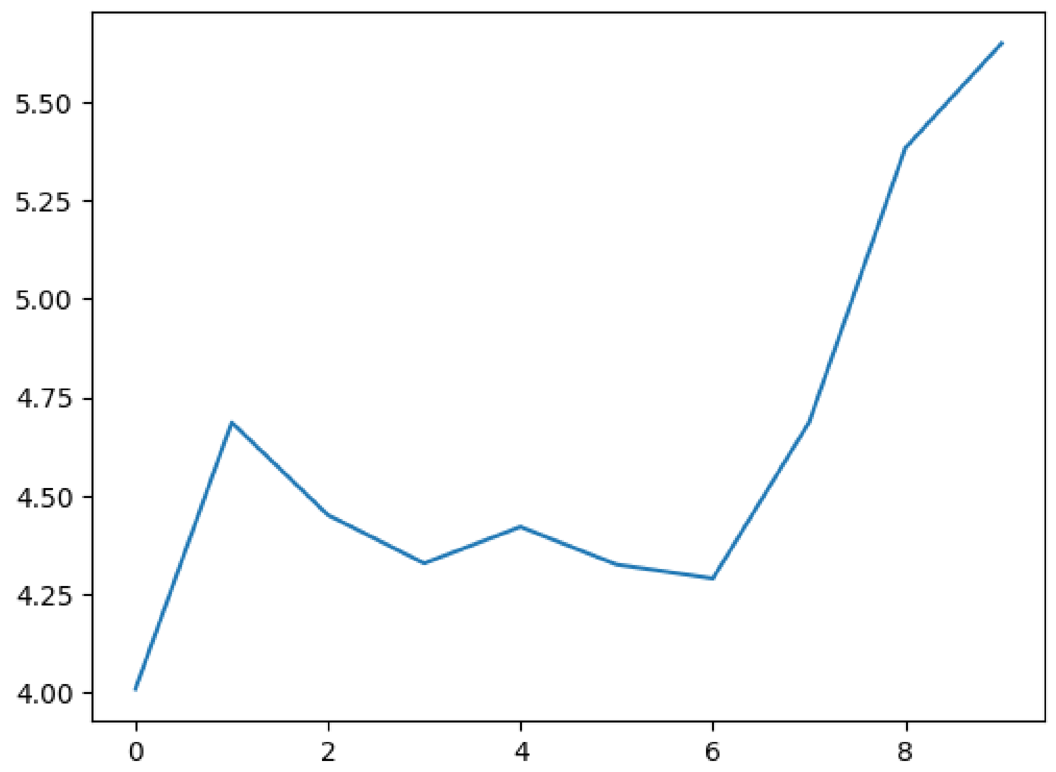
```
        # add to the epoch loss
        losses_epochs[-1] += loss.item() / (input_data.size(0) / batch_size)

    # time
    # Compute elapsed time and remaining time
    elapsed_time = time.time() - start_time
    avg_time_per_epoch = elapsed_time / (epoch + 1)
    remaining_epochs = num_epochs - (epoch + 1)
    estimated_remaining_time = avg_time_per_epoch * remaining_epochs

    # Convert remaining time to hours, minutes, and seconds for better␣
↪readability
    hours, remainder = divmod(estimated_remaining_time, 3600)
    minutes, seconds = divmod(remainder, 60)

    # Print the loss and remaining time for this epoch
    print('Epoch [{}/{}], Loss: {:.4f}, Time remaining: ~{}h {}m {:.0f}s'.
↪format(
        epoch+1, num_epochs, losses_epochs[-1], hours, minutes, seconds))
```

```
Epoch [1/10], Loss: 4.0086, Time remaining: ~0.0h 34.0m 25s
Epoch [2/10], Loss: 4.6856, Time remaining: ~0.0h 30.0m 40s
Epoch [3/10], Loss: 4.4504, Time remaining: ~0.0h 27.0m 35s
Epoch [4/10], Loss: 4.3278, Time remaining: ~0.0h 24.0m 3s
Epoch [5/10], Loss: 4.4203, Time remaining: ~0.0h 20.0m 3s
Epoch [6/10], Loss: 4.3245, Time remaining: ~0.0h 16.0m 27s
Epoch [7/10], Loss: 4.2892, Time remaining: ~0.0h 12.0m 39s
Epoch [8/10], Loss: 4.6862, Time remaining: ~0.0h 8.0m 20s
Epoch [9/10], Loss: 5.3831, Time remaining: ~0.0h 4.0m 22s
Epoch [10/10], Loss: 5.6487, Time remaining: ~0.0h 0.0m 0s
```

[24]:
```
#plot the loss
import matplotlib.pyplot as plt
plt.plot(losses)
plt.show()
```



[25]:
```
#plot the loss
import matplotlib.pyplot as plt
plt.plot(losses_epochs)
plt.show()
```



[26]:
```
avg_loss = 0
for x, (i, t) in enumerate(zip((input_validation), target_validation)):
    loss = criterion(model(i), t)
    avg_loss += loss/len(target_validation)
    print('i: {}, target: {:.3f}, output: {:.3f}, loss: {:.3f}'.format(x, t.
 ↪item(), model(i).item(), loss))

print('Average loss: {:.3f}'.format(avg_loss))
```

```
i: 0, target: -2.038, output: -0.337, loss: 2.894
i: 1, target: -1.026, output: -0.337, loss: 0.475
i: 2, target: -1.715, output: -0.337, loss: 1.898
i: 3, target: -1.323, output: -0.337, loss: 0.972
i: 4, target: -1.600, output: -0.337, loss: 1.594
i: 5, target: 0.566, output: -0.337, loss: 0.815
i: 6, target: -1.281, output: -0.337, loss: 0.891
i: 7, target: -0.574, output: -0.337, loss: 0.056
i: 8, target: -1.425, output: -0.337, loss: 1.184
i: 9, target: 0.183, output: -0.337, loss: 0.271
i: 10, target: -0.438, output: -0.337, loss: 0.010
i: 11, target: 0.266, output: -0.337, loss: 0.364
i: 12, target: -0.515, output: -0.337, loss: 0.032
```

```
i: 13, target: -1.352, output: -0.337, loss: 1.029
i: 14, target: -1.248, output: -0.337, loss: 0.829
i: 15, target: -1.726, output: -0.337, loss: 1.928
i: 16, target: -1.765, output: -0.337, loss: 2.040
i: 17, target: -0.564, output: -0.337, loss: 0.051
i: 18, target: -0.334, output: -0.337, loss: 0.000
i: 19, target: -0.890, output: -0.337, loss: 0.306
i: 20, target: -1.495, output: -0.337, loss: 1.341
i: 21, target: -0.574, output: -0.337, loss: 0.056
i: 22, target: -0.090, output: -0.337, loss: 0.061
i: 23, target: -0.168, output: -0.337, loss: 0.028
i: 24, target: -2.089, output: -0.337, loss: 3.069
i: 25, target: -2.053, output: -0.337, loss: 2.944
i: 26, target: -0.940, output: -0.337, loss: 0.363
i: 27, target: -0.635, output: -0.337, loss: 0.089
i: 28, target: -1.295, output: -0.337, loss: 0.917
i: 29, target: -0.659, output: -0.337, loss: 0.104
i: 30, target: -1.018, output: -0.337, loss: 0.463
i: 31, target: -0.641, output: -0.337, loss: 0.093
i: 32, target: -0.442, output: -0.337, loss: 0.011
i: 33, target: -1.094, output: -0.337, loss: 0.572
i: 34, target: -1.575, output: -0.337, loss: 1.532
i: 35, target: -0.919, output: -0.337, loss: 0.339
i: 36, target: -1.753, output: -0.337, loss: 2.005
i: 37, target: -1.702, output: -0.337, loss: 1.864
i: 38, target: -0.614, output: -0.337, loss: 0.077
i: 39, target: -0.629, output: -0.337, loss: 0.085
i: 40, target: -0.262, output: -0.337, loss: 0.006
i: 41, target: -1.358, output: -0.337, loss: 1.043
i: 42, target: -1.249, output: -0.337, loss: 0.832
i: 43, target: -2.004, output: -0.337, loss: 2.779
i: 44, target: -1.394, output: -0.337, loss: 1.117
i: 45, target: -1.459, output: -0.337, loss: 1.259
i: 46, target: -0.777, output: -0.337, loss: 0.193
i: 47, target: -1.120, output: -0.337, loss: 0.613
i: 48, target: -0.429, output: -0.337, loss: 0.008
i: 49, target: -1.742, output: -0.337, loss: 1.974
i: 50, target: -0.999, output: -0.337, loss: 0.438
i: 51, target: -0.753, output: -0.337, loss: 0.173
i: 52, target: -0.398, output: -0.337, loss: 0.004
i: 53, target: -1.444, output: -0.337, loss: 1.225
i: 54, target: -1.792, output: -0.337, loss: 2.117
i: 55, target: -0.905, output: -0.337, loss: 0.322
i: 56, target: -0.643, output: -0.337, loss: 0.094
i: 57, target: -0.539, output: -0.337, loss: 0.041
i: 58, target: -0.789, output: -0.337, loss: 0.204
i: 59, target: -1.127, output: -0.337, loss: 0.624
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[26], line 5
      3     loss = criterion(model(i), t)
      4     avg_loss += loss/len(target_validation)
----> 5     print('i: {}, target: {:.3f}, output: {:.3f}, loss: {:.3f}'.
 ↪format(x, t.item(), model(i).item(), loss))
      7 print('Average loss: {:.3f}'.format(avg_loss))

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\torch\nn\modules\module.
 ↪py:1501, in Module._call_impl(self, *args, **kwargs)
   1496 # If we don't have any hooks, we want to skip the rest of the logic in
   1497 # this function, and just call forward.
   1498 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1499         or _global_backward_pre_hooks or _global_backward_hooks
   1500         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1501     return forward_call(*args, **kwargs)
   1502 # Do not call functions when jit is used
   1503 full_backward_hooks, non_full_backward_hooks = [], []

File d:
 ↪\Raul\Programs\envs\PennyLane\lib\site-packages\torch\nn\modules\container.py
 ↪217, in Sequential.forward(self, input)
    215 def forward(self, input):
    216     for module in self:
--> 217         input = module(input)
    218     return input

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\torch\nn\modules\module.
 ↪py:1501, in Module._call_impl(self, *args, **kwargs)
   1496 # If we don't have any hooks, we want to skip the rest of the logic in
   1497 # this function, and just call forward.
   1498 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1499         or _global_backward_pre_hooks or _global_backward_hooks
   1500         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1501     return forward_call(*args, **kwargs)
   1502 # Do not call functions when jit is used
   1503 full_backward_hooks, non_full_backward_hooks = [], []

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\qnn\torch.py:
 ↪408, in TorchLayer.forward(self, inputs)
    405     results = torch.stack(reconstructor)
    406 else:
    407     # calculate the forward pass as usual
--> 408     results = self._evaluate_qnode(inputs)
```

```
    410 # reshape to the correct number of batch dims
    411 if has_batch_dim:

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\qnn\torch.py:
 ↪429, in TorchLayer._evaluate_qnode(self, x)
    417 """Evaluates the QNode for a single input datapoint.
    418
    419 Args:
   (…)
    423     tensor: output datapoint
    424 """
    425 kwargs = {
    426     **{self.input_arg: x},
    427     **{arg: weight.to(x) for arg, weight in self.qnode_weights.items()}
    428 }
--> 429 res = self.qnode(**kwargs)
    431 if isinstance(res, torch.Tensor):
    432     return res.type(x.dtype)

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\qnode.py:950,␣
 ↪in QNode.__call__(self, *args, **kwargs)
    948     self.execute_kwargs.pop("mode")
    949 # pylint: disable=unexpected-keyword-arg
--> 950 res = qml.execute(
    951     [self.tape],
    952     device=self.device,
    953     gradient_fn=self.gradient_fn,
    954     interface=self.interface,
    955     gradient_kwargs=self.gradient_kwargs,
    956     override_shots=override_shots,
    957     **self.execute_kwargs,
    958 )
    960 res = res[0]
    962 # convert result to the interface in case the qfunc has no parameters

File d:
 ↪\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\interfaces\executio
 ↪py:511, in execute(tapes, device, gradient_fn, interface, grad_on_execution,␣
 ↪gradient_kwargs, cache, cachesize, max_diff, override_shots, expand_fn,␣
 ↪max_expansion, device_batch_transform)
    503     # use qml.interfaces so that mocker can spy on it during testing
    504     cached_execute_fn = qml.interfaces.cache_execute(
    505         batch_execute,
    506         cache,
   (…)
    509         pass_kwargs=new_device_interface,
    510     )
--> 511     results = cached_execute_fn(tapes, execution_config=config)
```

```
    512     return batch_fn(results)
    514 # the default execution function is batch_execute
    515 # use qml.interfaces so that mocker can spy on it during testing

File d:
 ↪\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\interfaces\executio .
 ↪py:231, in cache_execute.<locals>.wrapper(tapes, **kwargs)
    228 repeated = {}
    230 for i, tape in enumerate(tapes):
--> 231     h = tape.hash
    233     if h in hashes.values():
    234         # Tape already exists within ``tapes``. Determine the
    235         # index of the first occurrence of the tape, store this,
    236         # and continue to the next iteration.
    237         idx = list(hashes.keys())[list(hashes.values()).index(h)]

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\tape\qscript.p :
 ↪238, in QuantumScript.hash(self)
    236 """int: returns an integer hash uniquely representing the quantum␣
 ↪script"""
    237 fingerprint = []
--> 238 fingerprint.extend(op.hash for op in self.operations)
    239 fingerprint.extend(m.hash for m in self.measurements)
    240 fingerprint.extend(self.trainable_params)

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\tape\qscript.p :
 ↪238, in <genexpr>(.0)
    236 """int: returns an integer hash uniquely representing the quantum␣
 ↪script"""
    237 fingerprint = []
--> 238 fingerprint.extend(op.hash for op in self.operations)
    239 fingerprint.extend(m.hash for m in self.measurements)
    240 fingerprint.extend(self.trainable_params)

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\operation.py:
 ↪697, in Operator.hash(self)
    689 @property
    690 def hash(self):
    691     """int: Integer hash that uniquely represents the operator."""
    692     return hash(
    693         (
    694             str(self.name),
    695             tuple(self.wires.tolist()),
    696             str(self.hyperparameters.values()),
--> 697             _process_data(self),
    698         )
    699     )
```

```
File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\operation.py:
 ↪382, in _process_data(op)
    377 def _process_data(op):
    378     # Use qml.math.real to take the real part. We may get complex input␣
 ↪for
    379     # example when differentiating holomorphic functions with JAX: a␣
 ↪complex
    380     # valued QNode (one that returns qml.state) requires complex typed␣
 ↪inputs.
    381     if op.name in ("RX", "RY", "RZ", "PhaseShift", "Rot"):
--> 382         return str([qml.math.round(qml.math.real(d) % (2 * np.pi), 10)␣
 ↪for d in op.data])
    384     if op.name in ("CRX", "CRY", "CRZ", "CRot"):
    385         return str([qml.math.round(qml.math.real(d) % (4 * np.pi), 10)␣
 ↪for d in op.data])

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\operation.py:
 ↪382, in <listcomp>(.0)
    377 def _process_data(op):
    378     # Use qml.math.real to take the real part. We may get complex input␣
 ↪for
    379     # example when differentiating holomorphic functions with JAX: a␣
 ↪complex
    380     # valued QNode (one that returns qml.state) requires complex typed␣
 ↪inputs.
    381     if op.name in ("RX", "RY", "RZ", "PhaseShift", "Rot"):
--> 382         return str([qml.math.round(qml.math.real(d) % (2 * np.pi), 10)␣
 ↪for d in op.data])
    384     if op.name in ("CRX", "CRY", "CRZ", "CRot"):
    385         return str([qml.math.round(qml.math.real(d) % (4 * np.pi), 10)␣
 ↪for d in op.data])

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\autoray\autoray.py:29, i␣
 ↪do(fn, like, *args, **kwargs)
     25 from inspect import signature
     26 from collections import OrderedDict, defaultdict
---> 29 def do(fn, *args, like=None, **kwargs):
     30     """Do function named ``fn`` on ``(*args, **kwargs)``, peforming␣
 ↪single
     31     dispatch to retrieve ``fn`` based on whichever library defines the␣
 ↪class of
     32     the ``args[0]``, or the ``like`` keyword argument if specified.
   (…)
     76         <tf.Tensor: id=91, shape=(3, 3), dtype=float32>
     77     """
     78     backend = choose_backend(fn, *args, like=like, **kwargs)
```

```
KeyboardInterrupt:
```

4 Save the Notebook as a PDF

[19]:
```
# SAVE THE NOTEBOOK

from IPython.display import Javascript

# Define the function to save the notebook
def save_notebook():
    display(Javascript('IPython.notebook.save_notebook()'))

# Call the save_notebook function to save the notebook
save_notebook()
```

```
<IPython.core.display.Javascript object>
```

[22]:
```
import subprocess
import os

name_notebook = "0724-small_hybrid_model_v5.ipynb"

output_filename = "results/"+ name_notebook[:4] +"/" + name_notebook[:-6] + "_0.
 ↪pdf"

#check if the output file already exists
while os.path.exists(output_filename):
    print("The file {} already exists".format(output_filename))
    output_filename = output_filename[:-5] + str(int(output_filename[-5]) + 1)␣
 ↪+ ".pdf"
    print("Trying to save the file as {}".format(output_filename))


subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output",␣
 ↪output_filename, name_notebook])
```

[22]:
```
CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
'results/0724/0724-small_hybrid_model_v5_0.pdf',
'0724-small_hybrid_model_v5.ipynb'], returncode=0)
```

[ ]:
```

```