0717-model

July 27, 2023

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

[ ]:
```
%load_ext autoreload
%autoreload 2
from my_code import model as m
from my_code import layers
from my_code import functions as f
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

1 DATA

[ ]:
```
from pennylane import numpy as np
file_path = '../data/energies/Trial/Strings_Energies.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, requires_grad=False)/1000
angles_list = np.array([f.string_to_angles(string) for string in string_list],␣
 ↪requires_grad=False)
```

[ ]:
```
X, Y = angles_list, score_list
```

2 Model

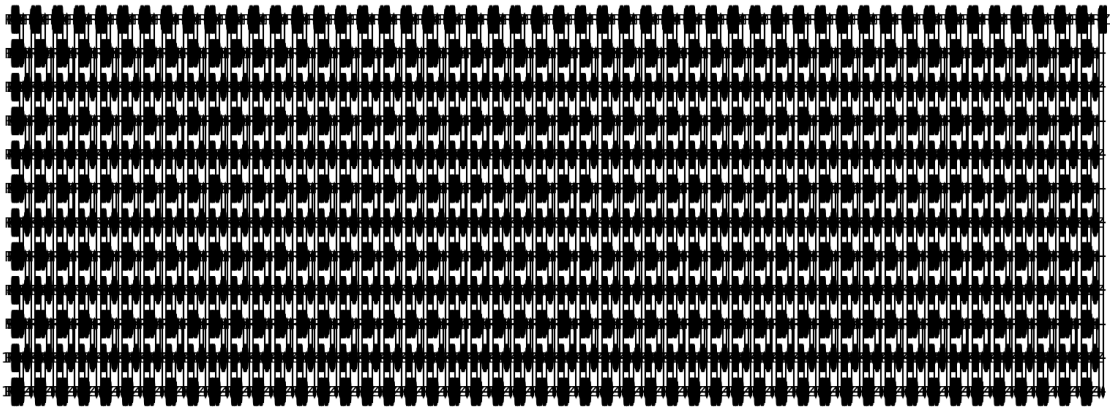
2.1 Define

[ ]:
```
circuit_layers = \
    [layers.angle_preparation()] + \
    [layers.rotationX_layer(), layers.rotationZ_layer(), layers.
 ↪rotationZZ_layer()] * 50 +  \
    [layers.mesurament(qubits=[0])]
```

[ ]:
```
import pennylane as qml
SCORE_PREDICTOR = m.model(
    n_qubits_data = 12,
```

```
    circuit_layers = circuit_layers
)
```

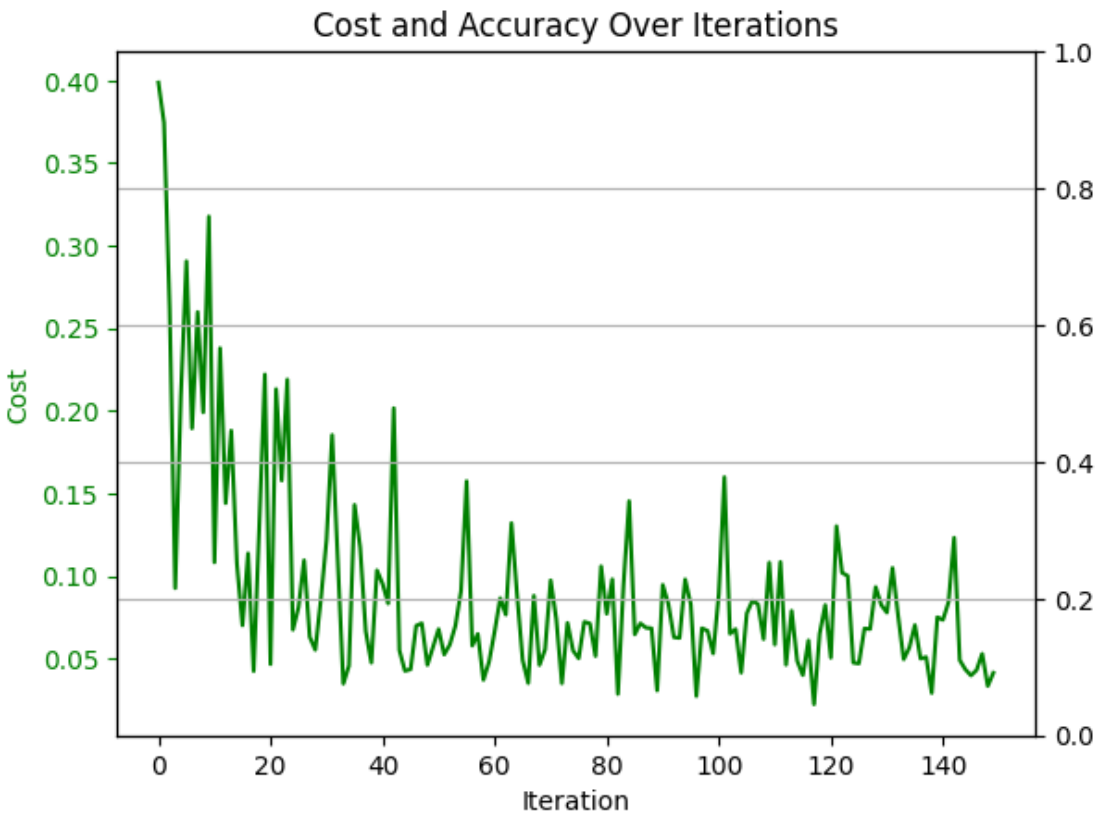
[ ]:
```
SCORE_PREDICTOR.draw_circuit(size=(15, 6))
```



2.2 Train

[ ]:
```
SCORE_PREDICTOR.set_data(
    data_X = X,
    data_Y = Y,
    # data_validation = data_validation
)
```

[ ]:
```
SCORE_PREDICTOR.train(
    epochs = 100,
    batch_size = 10,
    optimizer = qml.SPSAOptimizer(maxiter=1000, c=0.15, a=0.2),
    randomize_batches = False,
    initialize_params = True,
    plot_options={
        'accuracy': False,
        'accuracy_validation': False,
        'plot_every': 5
    }
)
```




```
Epoch: 1 | Iter:    50 | Cost: 0.0414446 | Accuracy: 0.9585554
Epoch: 1 | Iter:    51 | Cost: 0.0534124 | Accuracy: 0.9465876
Epoch: 1 | Iter:    52 | Cost: 0.0391504 | Accuracy: 0.9608496
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[19], line 1
----> 1 SCORE_PREDICTOR.train(
      2     epochs = 100,
      3     batch_size = 10,
      4     optimizer = qml.SPSAOptimizer(maxiter=1000, c=0.15, a=0.2),
      5     randomize_batches = False,
      6     initialize_params = True,
      7     plot_options={
      8         'accuracy': False,
      9         'accuracy_validation': False,
     10         'plot_every': 5
     11     }
     12 )
```

```
File d:\Raul\OneDrive - Cornell University\Code\peptide-QML\Notebooks\..
 ↪\my_code\model.py:167, in model.train(self, epochs, optimizer, batch_size,␣
 ↪randomize_batches, initialize_params, plot, plot_options)
    164 Y_batch = data_Y_batches[it]
    166 # Update parameters and append cost
--> 167 params, cost = self.optimizer.step_and_cost(self.cost, X_batch, Y_batch␣
 ↪*self.params)
    168 self.params = params[2:]
    169 self.costs.append(cost)

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\optimize\spsa.
 ↪py:200, in SPSAOptimizer.step_and_cost(self, objective_fn, *args, **kwargs)
    196 new_args = self.apply_grad(g, args)
    198 self.k += 1
--> 200 forward = objective_fn(*args, **kwargs)
    202 # unwrap from list if one argument, cleaner return
    203 if len(new_args) == 1:

File d:\Raul\OneDrive - Cornell University\Code\peptide-QML\Notebooks\..
 ↪\my_code\model.py:114, in model.cost(self, X, Y, *params)
    113 def cost(self, X, Y, *params):
--> 114     output = [self.variational_classifier(x, params) for x in X]
    115     cost = self.loss(Y, output)
    116     self.last_cost = { #TODO history of costs and parameters
    117         'X': X,
    118         'Y': Y,
    119         'output': output
    120     }

File d:\Raul\OneDrive - Cornell University\Code\peptide-QML\Notebooks\..
 ↪\my_code\model.py:114, in <listcomp>(.0)
    113 def cost(self, X, Y, *params):
--> 114     output = [self.variational_classifier(x, params) for x in X]
    115     cost = self.loss(Y, output)
    116     self.last_cost = { #TODO history of costs and parameters
    117         'X': X,
    118         'Y': Y,
    119         'output': output
    120     }

File d:\Raul\OneDrive - Cornell University\Code\peptide-QML\Notebooks\..
 ↪\my_code\model.py:95, in model.variational_classifier(self, input, params,␣
 ↪draw, draw_options)
     91     fig.set_size_inches(draw_options['size'])
     93 bias = params[-1]  if self.bias else 0
---> 95 output = circuit(input, params) + bias
     96 return output
```

```
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:287, in cache_execute.<locals>.wrapper(tapes, **kwargs)
    282         return (res, []) if return_tuple else res
    284 else:
    285     # execute all unique tapes that do not exist in the cache
    286     # convert to list as new device interface returns a tuple
--> 287     res = list(fn(execution_tapes.values(), **kwargs))
    289 final_res = []
    291 for i, tape in enumerate(tapes):

File d:
 ↪\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\interfaces\executio␣
 ↪py:210, in cache_execute.<locals>.fn(tapes, **kwargs)
    208 def fn(tapes: Sequence[QuantumTape], **kwargs):  # pylint:␣
 ↪disable=function-redefined
```

```
    209     tapes = [expand_fn(tape) for tape in tapes]
--> 210     return original_fn(tapes, **kwargs)

File d:\Raul\Programs\envs\PennyLane\lib\contextlib.py:79, in ContextDecorator.
 ↪__call__.<locals>.inner(*args, **kwds)
     76 @wraps(func)
     77 def inner(*args, **kwds):
     78     with self._recreate_cm():
---> 79         return func(*args, **kwds)

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\_qubit_device.
 ↪py:603, in QubitDevice.batch_execute(self, circuits)
    598 for circuit in circuits:
    599     # we need to reset the device here, else it will
    600     # not start the next computation in the zero state
    601     self.reset()
--> 603     res = self.execute(circuit)
    604     results.append(res)
    606 if self.tracker.active:

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\_qubit_device.
 ↪py:320, in QubitDevice.execute(self, circuit, **kwargs)
    317 self.check_validity(circuit.operations, circuit.observables)
    319 # apply all circuit operations
--> 320 self.apply(circuit.operations, rotations=self.
 ↪_get_diagonalizing_gates(circuit), **kwargs)
    322 # generate computational basis samples
    323 if self.shots is not None or circuit.is_sampled:

File d:
 ↪\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\devices\default_qub.t.
 ↪py:293, in DefaultQubit.apply(self, operations, rotations, **kwargs)
    291         self._state = self._apply_parametrized_evolution(self._state,␣
 ↪operation)
    292     else:
--> 293         self._state = self._apply_operation(self._state, operation)
    295 # store the pre-rotated state
    296 self._pre_rotated_state = self._state

File d:
 ↪\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\devices\default_qub.t.
 ↪py:336, in DefaultQubit._apply_operation(self, state, operation)
    333 matrix = self._asarray(self._get_unitary_matrix(operation), dtype=self.
 ↪C_DTYPE)
    335 if operation in diagonal_in_z_basis:
--> 336     return self._apply_diagonal_unitary(state, matrix, wires)
    337 if len(wires) <= 2:
    338     # Einsum is faster for small gates
```

```
    339     return self._apply_unitary_einsum(state, matrix, wires)

File d:
 ↪\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\devices\default_qub t.
 ↪py:926, in DefaultQubit._apply_diagonal_unitary(self, state, phases, wires)
    923 affected_indices = "".join(ABC_ARRAY[list(device_wires)].tolist())
    925 einsum_indices =␣
 ↪f"…{affected_indices},…{state_indices}->…{state_indices}"
--> 926 return self._einsum(einsum_indices, phases, state)

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\pennylane\numpy\wrapper.
 ↪py:117, in tensor_wrapper.<locals>._wrapped(*args, **kwargs)
    114         tensor_kwargs["requires_grad"] = _np.any([i.requires_grad for i␣
 ↪in tensor_args])
    116 # evaluate the original object
--> 117 res = obj(*args, **kwargs)
    119 if isinstance(res, _np.ndarray):
    120     # only if the output of the object is a ndarray,
    121     # then convert to a PennyLane tensor
    122     res = tensor(res, **tensor_kwargs)

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\autograd\tracer.py:48, i␣
 ↪primitive.<locals>.f_wrapped(*args, **kwargs)
     46     return new_box(ans, trace, node)
     47 else:
---> 48     return f_raw(*args, **kwargs)

File <__array_function__ internals>:180, in einsum(*args, **kwargs)

File d:\Raul\Programs\envs\PennyLane\lib\site-packages\numpy\core\einsumfunc.py
 ↪1371, in einsum(out, optimize, *operands, **kwargs)
   1369     if specified_out:
   1370         kwargs['out'] = out
-> 1371     return c_einsum(*operands, **kwargs)
   1373 # Check the kwargs to avoid a more cryptic error later, without having t␣
 ↪o
   1374 # repeat default values here
   1375 valid_einsum_kwargs = ['dtype', 'order', 'casting']

KeyboardInterrupt:
```

2.3 Try

[ ]:
```
# take 20 items from the data set randomly

import random
random.seed(42)
random_index = random.sample(range(0, len(X)), 20)
```

```
X_test = X[random_index]
Y_test = Y[random_index]

# predict the score for the 20 items
Y_predicted = [SCORE_PREDICTOR.predict(x) for x in X_test]
```

[ ]:
```
# print the results
for i in range(len(X_test)):
    print("String: {} \tScore: {:.3f} \tPredicted: {:.3f} \tDiference: {:.3f}".
 ↪format(i, Y_test[i].item(), Y_predicted[i].item(), abs(Y_test[i].item() -␣
 ↪Y_predicted[i].item())))
```

```
String: 0       Score: 0.300    Predicted: 0.370        Diference: 0.070
String: 1       Score: 0.667    Predicted: 0.353        Diference: 0.314
String: 2       Score: 0.244    Predicted: 0.445        Diference: 0.201
String: 3       Score: 0.514    Predicted: 0.430        Diference: 0.083
String: 4       Score: 0.006    Predicted: 0.379        Diference: 0.373
String: 5       Score: 0.772    Predicted: 0.407        Diference: 0.365
String: 6       Score: 0.316    Predicted: 0.386        Diference: 0.069
String: 7       Score: 0.581    Predicted: 0.404        Diference: 0.177
String: 8       Score: 0.475    Predicted: 0.400        Diference: 0.075
String: 9       Score: 0.404    Predicted: 0.354        Diference: 0.050
String: 10      Score: 0.587    Predicted: 0.335        Diference: 0.253
String: 11      Score: 0.331    Predicted: 0.398        Diference: 0.068
String: 12      Score: 0.677    Predicted: 0.390        Diference: 0.287
String: 13      Score: 0.153    Predicted: 0.402        Diference: 0.249
String: 14      Score: 0.270    Predicted: 0.388        Diference: 0.118
String: 15      Score: 0.119    Predicted: 0.416        Diference: 0.297
String: 16      Score: 0.449    Predicted: 0.399        Diference: 0.050
String: 17      Score: 0.675    Predicted: 0.422        Diference: 0.253
String: 18      Score: 0.401    Predicted: 0.423        Diference: 0.022
String: 19      Score: 0.364    Predicted: 0.381        Diference: 0.016
```

[ ]:

[ ]: