NIThePmini_school_L2-data_encoding

September 15, 2020

1 Encoding classical data on quantum computers

This tutorial demonstrates various data encoding techniques using pieces of code from PennyLane and Qiskit - two open source libraries for quantum computing. For more info, check out: https://qiskit.org/documentation/ and https://pennylane.readthedocs.io/en/stable/

1.1 Basis encoding - PennyLane

1.1.1 PennyLane templates

[2]:
```
# Basis encoding example with PennyLane

import pennylane as qml
from pennylane import numpy as np

# import the template
from pennylane.templates.embeddings import BasisEmbedding
```

[7]:
```
data = np.array([1,1,1])

dev = qml.device('default.qubit', wires=3)

@qml.qnode(dev)
def circuit(data):
    BasisEmbedding(features=data, wires=range(3))
    return qml.expval(qml.PauliZ(0)), qml.expval(qml.PauliZ(1)), qml.expval(qml.
 ↪PauliZ(2))
```

[8]:
```
circuit(data)
```

[8]:
```
array([-1., -1., -1.])
```

1.2 Angle encoding - PennyLane

Import some data

[19]:
```
from sklearn.datasets import load_iris # import Iris data set
from sklearn.utils import shuffle
```

```
from sklearn.preprocessing import normalize # import a normilisation function
np.random.seed(42) # set a seed
```

[20]:
```
x, Y = load_iris().data, load_iris().target
x, Y = shuffle(x,Y)

# take the first 5
x = x[:5]
Y = Y[:5]
```

[21]:
```
print(x, Y)
```

```
[[6.1 2.8 4.7 1.2]
 [5.7 3.8 1.7 0.3]
 [7.7 2.6 6.9 2.3]
 [6.  2.9 4.5 1.5]
 [6.8 2.8 4.8 1.4]] [1 0 2 1 1]
```

[22]:
```
# normalize the data
data = normalize(x)
print(data)
```

```
[[0.73659895 0.33811099 0.56754345 0.14490471]
 [0.8068282  0.53788547 0.24063297 0.04246464]
 [0.70600618 0.2383917  0.63265489 0.21088496]
 [0.73350949 0.35452959 0.55013212 0.18337737]
 [0.76467269 0.31486523 0.53976896 0.15743261]]
```

Import the PennyLane template for angle embedding

[23]:
```
from pennylane.templates.embeddings import AngleEmbedding
```

[28]:
```
num_qubits = 4

dev = qml.device('default.qubit', wires=num_qubits)

@qml.qnode(dev)
def circuit(data):
    # apply Hadamards to all qubits in the circuit
    for i in range(num_qubits):
        qml.Hadamard(wires=i)
    AngleEmbedding(features=data, wires=range(num_qubits), rotation='Y')
    return qml.expval(qml.PauliZ(0)), qml.expval(qml.PauliZ(1)), qml.expval(qml.
 ↪PauliZ(2)), qml.expval(qml.PauliZ(3))
```

[31]:
```
circuit(data[3])
```

[31]:
```
array([-0.6694807 , -0.34714924, -0.52279986, -0.18235135])
```

[32]:
```
# encode all data
@qml.qnode(dev)
def circuit(data):
    # apply Hadamards to all qubits in the circuit
    for i in range(num_qubits):
        qml.Hadamard(wires=i)

    for i in range(len(data)):
        AngleEmbedding(features=data[i], wires=range(num_qubits), rotation='Y')
    return qml.expval(qml.PauliZ(0)), qml.expval(qml.PauliZ(1)), qml.expval(qml.
↪PauliZ(2)), qml.expval(qml.PauliZ(3))
```

[33]:
```
circuit(data)
```

[33]:
```
array([ 0.56960308, -0.97740396, -0.57357236, -0.67359663])
```

1.3 Angle encoding - Qiskit

[34]:
```
from qiskit import *
from qiskit.circuit.library import ZFeatureMap
```

```
/Users/amyami187/anaconda3/lib/python3.6/importlib/_bootstrap.py:219:
RuntimeWarning: numpy.ufunc size changed, may indicate binary incompatibility.
Expected 192 from C header, got 216 from PyObject
  return f(*args, **kwds)
```

[36]:
```
featuremap_circ = ZFeatureMap(4, reps=1)
```

[37]:
```
print(featuremap_circ)
```

```
q_0:  H  U1(2.0*x[0])

q_1:  H  U1(2.0*x[1])

q_2:  H  U1(2.0*x[2])

q_3:  H  U1(2.0*x[3])
```

[39]:
```
# assign data to circuit parameters
circ_data0 = featuremap_circ.assign_parameters(data[0]/2)

# print
print(circ_data0)
```

```
q_0:   H    U1(0.7366)

q_1:   H   U1(0.33811)

q_2:   H   U1(0.56754)

q_3:   H    U1(0.1449)
```

[41]:
```
# combine the featuremap circuit with assigned parameters of the second data␣
 ↪point
circ_data1 = circ_data0.combine(featuremap_circ.assign_parameters(data[1]/2))
print(circ_data1)
```

```
q_0:   H    U1(0.7366)   H   U1(0.80683)

q_1:   H   U1(0.33811)   H   U1(0.53789)

q_2:   H   U1(0.56754)   H   U1(0.24063)

q_3:   H    U1(0.1449)   H   U1(0.042465)
```

[42]:
```
# More generally, ...
circ = QuantumCircuit(4)

for i in range(len(data)):
    circ = circ.combine(featuremap_circ.assign_parameters(data[i]/2))

print(circ)
```

```
                                          »
q_0:   H    U1(0.7366)   H   U1(0.80683)    H   U1(0.70601)   H »
                                         »
q_1:   H   U1(0.33811)   H   U1(0.53789)    H   U1(0.23839)   H »
                                         »
q_2:   H   U1(0.56754)   H   U1(0.24063)    H   U1(0.63265)   H »
                                        »
q_3:   H    U1(0.1449)   H   U1(0.042465)   H   U1(0.21088)   H »
                                        »
«
«q_0:   U1(0.73351)   H   U1(0.76467)
«
«q_1:   U1(0.35453)   H   U1(0.31487)
«
«q_2:   U1(0.55013)   H   U1(0.53977)
«
```

```
«q_3:   U1(0.18338)   H   U1(0.15743)
«
```

1.4 Higher order encoding - Qiskit

[43]:
```
from qiskit.circuit.library import ZZFeatureMap
```

[44]:
```
featuremap_circ = ZZFeatureMap(2, reps=1)
print(featuremap_circ)
```

```
                    »
q_0:   H   U1(2.0*x[0])    »
                   »
q_1:   H   U1(2.0*x[1])   X  »
                   »
«
«q_0:
«
«q_1:   U1(2.0*(3.14159265358979 - x[0])*(3.14159265358979 - x[1]))   X
«
```

[ ]:
```

```