Variational Quantum Classifier applied on the Iris Dataset

1. Introduction

In this tutorial we show how the `Variational Quantum Classifier` or `VQC` can be applied on the Iris Dataset.

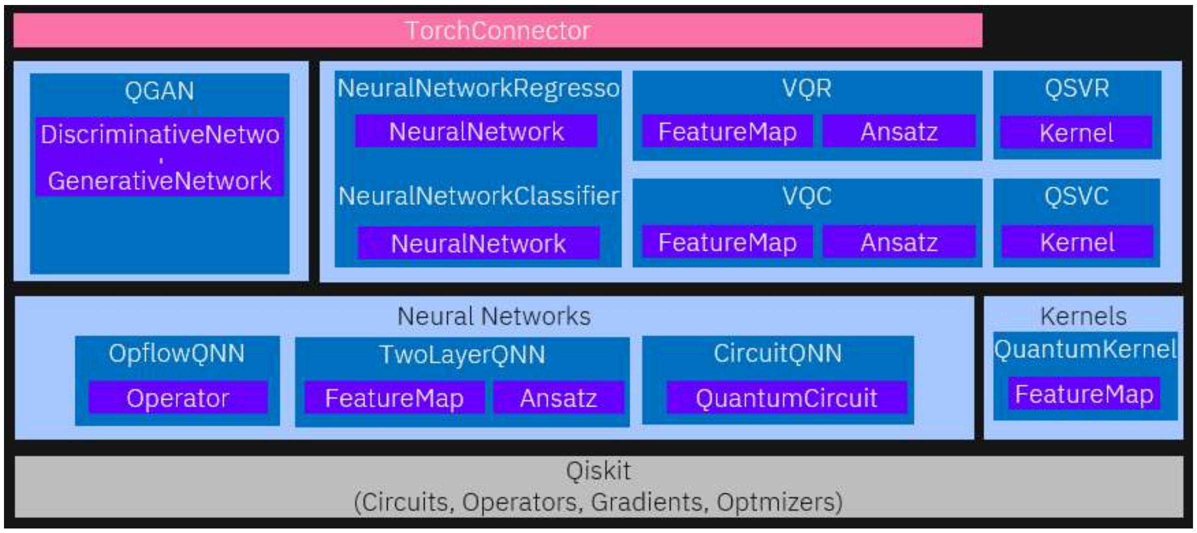
Although the Iris Dataset is well known within the Data Scientist and Data Architect community, we also provide an introduction to it, be it for junior professionals.

After a short data analysis exercise on the Iris Dataset, we make use of a `Variational Quantum Classifiers` or `VQC` to predict the species of a flower, based on its features.

We try our several options for the VQC.

2. Qiskit Machine Learning Module

Within the Qiskit Machine Learning module, VQC is an essential part, as can be seen in the picture below.
A VQC relies on an Feature Map and an Ansatz.



2.1. Feature Map

A Feature Map is a function which maps a vector with data to a Feature Space.
The main reason for doing so is to present the Machine Learning algorithm with data that it is

more appropriate for regression or classification.
A wide variety of functions and transformations can be written as Feature Maps.
Anyhow, at a given moment in our code, we will need to define a Feature Map.

2.2. Ansatz

In Quantum Algorithms, the term "Ansatz" is widely used. The "Ansatz" is the first guess. A starting point.
Of course, if the starting point is good, so will the result.
The Ansatz usually describes a "subroutine" consisting of a set of Gates applied to specific Qubits. But these Gates only define the base structure, while the types of Gates and/or their free parameters can be optimized by the VQC.

3. The Iris Dataset

3.1. Introducing the Iris Dataset

The Iris Datset, or the Iris Flower Dataset, or the Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper "The use of multiple measurements in taxonomic problems" as an example of linear discriminant analysis. It is sometimes called Anderson's Iris data set because Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species.

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

Our objective in this tutorial is to create a Machine Learning Model that predicts as accurately as possible the Iris Flower Species, based on 4 given features: sepal length, sepal width, petal length and petal width.

But before doing so, let us perform an basic analysis on the Iris Dataset.

3.2. Basic Analysis on the Iris Dataset

We first download the Iris Dataset after importing some key tools.

Then we perform some elementary analysis on the Iris Dataset, so that you have a deep understanding of it. You can allways comment out these lines of code.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import time

iris_data = load_iris()
iris_data_df = pd.DataFrame(data=iris_data.data, columns=iris_data.feature_names)
iris_data_df["target"] = iris_data.target
```

We now perform some elementary analyses.

In [2]:
```
# What are the names of the species and features?
print(set(iris_data["target"]))
print(iris_data["target_names"])
print(iris_data["feature_names"])
```

```
{0, 1, 2}
['setosa' 'versicolor' 'virginica']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

In [3]:
```
# What is the shape of the Iris Dataset?
iris_data_df.shape
```

Out[3]:
```
(150, 5)
```

In [4]:
```
# What are the data types?
iris_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    int32  
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

In [5]:
```
# Let us double check on the missing values
iris_data_df.isnull().sum()
```

Out[5]:
```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64
```

In [7]:
```
# What are the first 5 rows in our Iris Dataset?
iris_data_df.head()
```

Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [8]:
```
# What are the last 5 rows in our Iris Dataset?
iris_data_df.tail()
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

In [9]:
```
# Some more statistical insights:
iris_data_df.describe()
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

In [6]:
```
# Are there a lot of duplicates?
# Should we perform any delete operations to balance the Dataset?
# Check for duplicate entries
iris_data_df[iris_data_df.duplicated()]
```

Out[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
142	5.8	2.7	5.1	1.9	2

In [7]:
```
# Check the balance
iris_data_df['target'].value_counts()
```

Out[7]:
```
0    50
1    50
2    50
Name: target, dtype: int64
```

There is no need to delete any rows, the Datset is sufficiently balanced.

4. VQC on Iris Dataset

Preliminary: The complete syntax for invoking the VQC can be found here: https://qiskit.org/documentation/machine-learning/stubs/qiskit_machine_learning.algorithms.VQC.html?highlight=vqc

4.1. The Optimizer in VQC

We first run the import statements, nicely grouped together.

In [8]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from qiskit import Aer, QuantumCircuit
from qiskit.utils import QuantumInstance, algorithm_globals

from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap
from qiskit.algorithms.optimizers import COBYLA
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit_machine_learning.exceptions import QiskitMachineLearningError
from IPython.display import clear_output
```

In [ ]:
```
The VQC is a variational algorithm where the measured expectation value is interpretec
During training, an instance of an optimizer is used. The default is `SLSQP`(which sta
We will try out the following optimizers:
- COBYLA
- SLSQP
- SPSA: here the number of callbacks is equal to the double of the number of iteratior
```

COBYLA = Constrained Optimization BY Linear Approximation. COBYLA is a numerical optimization method for constrained problems where the derivative of the objective function is not known. That is, COBYLA can find the vector that has the minimal (or maximal) without knowing the gradient of the function.

In [9]:
```
optimizer = COBYLA(maxiter=120, tol=0.001)
```

4.2. The Callback Function

The callback can access intermediate data during training. On each iteration the optimizer invokes the callback and passes current weights as an array and a computed value as a float of the objective function being optimized. This allows to track how well the training process is going on.

In [10]:
```python
# Callback function that draws a live plot when the .fit() method is called
def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()
```

The Callback Function is named `callback_graph`. It is clear now why it will be called for each iteration of the Optimizer and will be passed two parameters:

- The current weights
- The value of the objective function at those weights

For our function, we append the value of the objective function to an array so we can plot iteration versus objective function value and update the graph with each iteration.

Please note you can do whatever you want with a callback function as long as it gets the two parameters mentioned passed.

4.3. Data Normalization and One-Hot Encoding

In Machine Learning it is a common practice to normalize the data. This is because we want the independent variables to have equivalent values and impact during the training process. In this tutorial we make use of the MinMaxScaler.

MinMaxScaler transforms features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between -1 and 1. The default is [0, 1] and that is what we will do in our Tutorial.

In [11]:
```python
mms = MinMaxScaler()
X_features = iris_data['data']
X_features = mms.fit_transform(X_features)
```

When consulting the documentation of VQC, we learn that:

- VQC only supports one-hot encoded labels; e.g., data like [1, 0, 0], [0, 1, 0], [0, 0, 1].
- Multi-label classification is not supported; e.g., data like [1, 1, 0], [0, 1, 1], [1, 0, 1], [1, 1, 1].

That is why we need `OneHotEncoder` which transforms the labels 0, 1 and 2 as follows:

- 0 becomes [1, 0, 0]
- 1 becomes [0, 1, 0]
- 2 becomes [0, 0, 1]

In [14]:
```python
labels = iris_data['target']
labels = OneHotEncoder(sparse=False).fit_transform(labels.reshape(-1 , 1))
```

4.4. Definition of a seed

In the code that follows, we will use a pseudo-random number generator (a.k.a. `PRNG`).
PRNG is algorithm that generates sequence of numbers approximating the properties of random numbers.
We want these random numners to be reproduced so that the outcomes of our Machine Learning effort are not influenced by variability in the pseudo random numbers. These random numbers can be reproduced using the seed value `algorithm_globals.random_seed`.

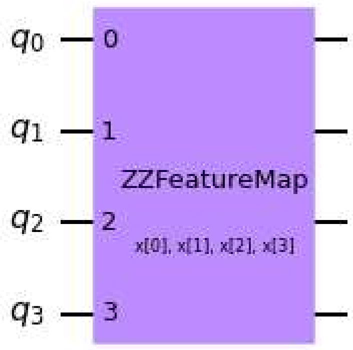
In [16]:
```
algorithm_globals.random_seed = 123
```

4.5. Definition of the Feature Map

As already announced in the introduction, we define our feature map.
We start with 1 repetition and full entanglement.
As we have 4 features (sepal length, sepal width, petal length, petal width) and as such we make use of 4 Qubits.

In [17]:
```
feature_dim = 4
ZZ_feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=1, entanglement='ful
ZZ_feature_map.draw('mpl')
```

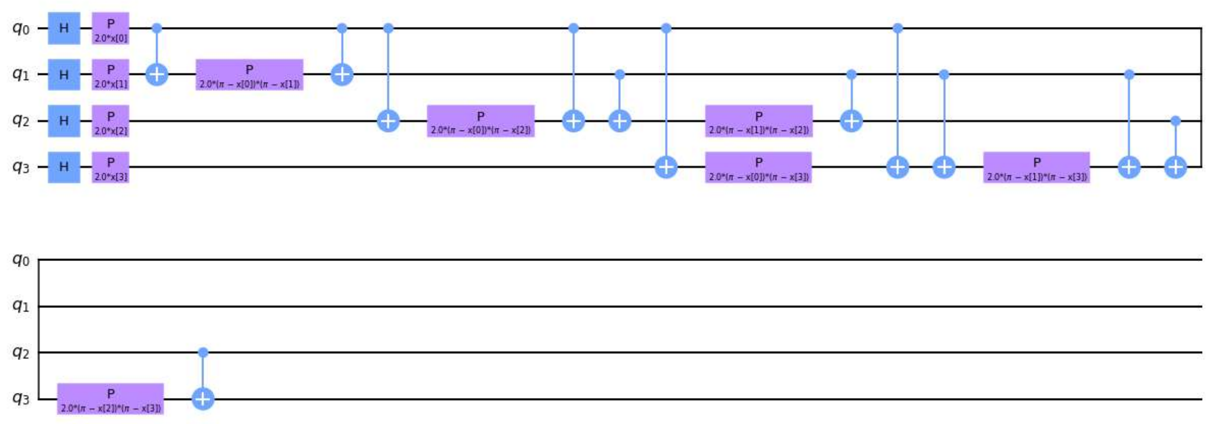
Out[17]:



As this picture is not very clarifying. So let us decompose the Circuit and illustrate how it is constructed.

In [18]:
```
ZZ_feature_map.decompose().draw('mpl')
```

Out[18]:

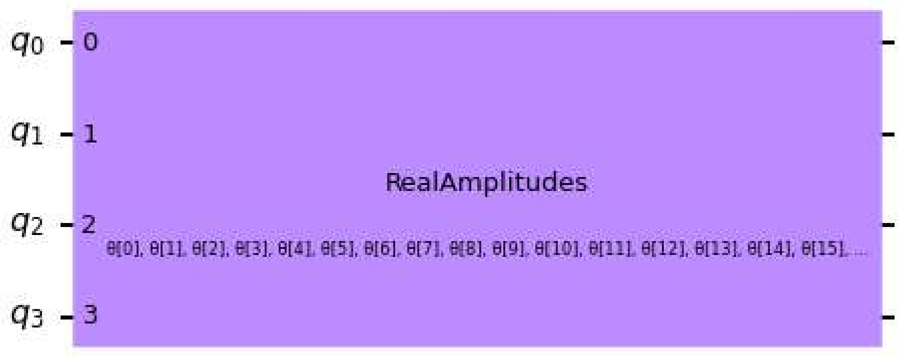

q_0
q_1
q_2
q_3
H
P
2.0*x[0]
2.0*x[1]
2.0*x[2]
2.0*x[3]
2.0*(π − x[0])*(π − x[1])
2.0*(π − x[0])*(π − x[2])
2.0*(π − x[1])*(π − x[2])
2.0*(π − x[0])*(π − x[3])
2.0*(π − x[1])*(π − x[3])
2.0*(π − x[2])*(π − x[3])

4.6. Definition of the Ansatz

We call the Ansatz the "Variational Circuit". We start with full entanglement and 4 repetitions.

In [19]:
```python
num_qubits = feature_dim
variational_circ = RealAmplitudes(num_qubits, entanglement='full', reps=4)
variational_circ.draw('mpl')
```

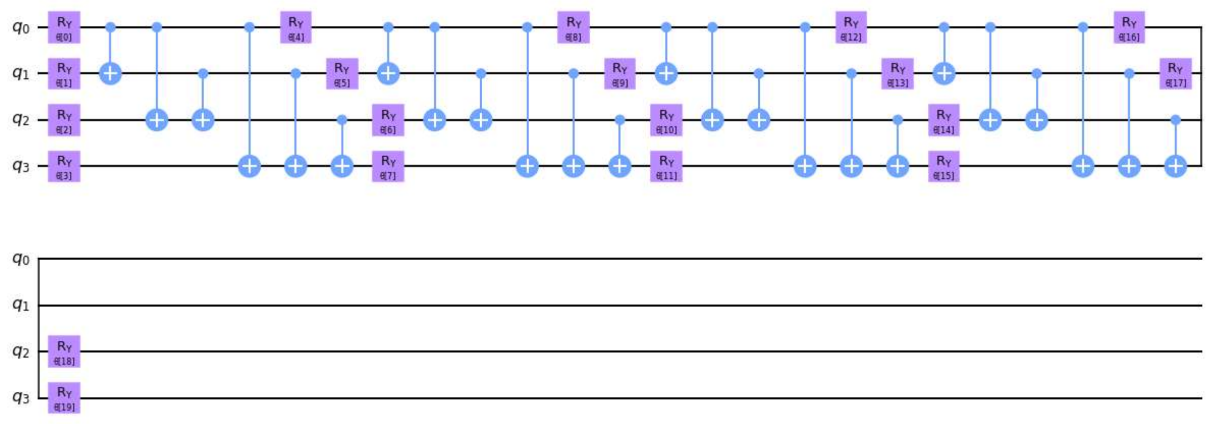
Out[19]:


q_0
q_1
q_2
q_3
0
1
2
3
RealAmplitudes
θ[0], θ[1], θ[2], θ[3], θ[4], θ[5], θ[6], θ[7], θ[8], θ[9], θ[10], θ[11], θ[12], θ[13], θ[14], θ[15], ...

Let us also decompose this Circuit and illustrate how it is constructed.

In [20]:
```python
variational_circ.decompose().draw('mpl')
```

Out[20]:



4.7. Definition of the Backend and Initial Point

We not only define the Backend but also an initial point where the optimizer should start.

In [21]:
```
backend = Aer.get_backend('statevector_simulator')
quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=algorithm_globa
        seed_transpiler=algorithm_globals.random_seed)
initial_point = algorithm_globals.random.random(variational_circ.num_parameters)
# Number of shots does not make  sense. In case qasm that makes sense, it is noisy.
# We should end with qasm - Aer
```

To learn more about how to specify a Quantum Instance, please surf to https://qiskit.org/documentation/stubs/qiskit.utils.QuantumInstance.html?highlight=quantum%20instance#qiskit.utils.QuantumInstance

4.8. Split up in Training and Test Data and launch the Training

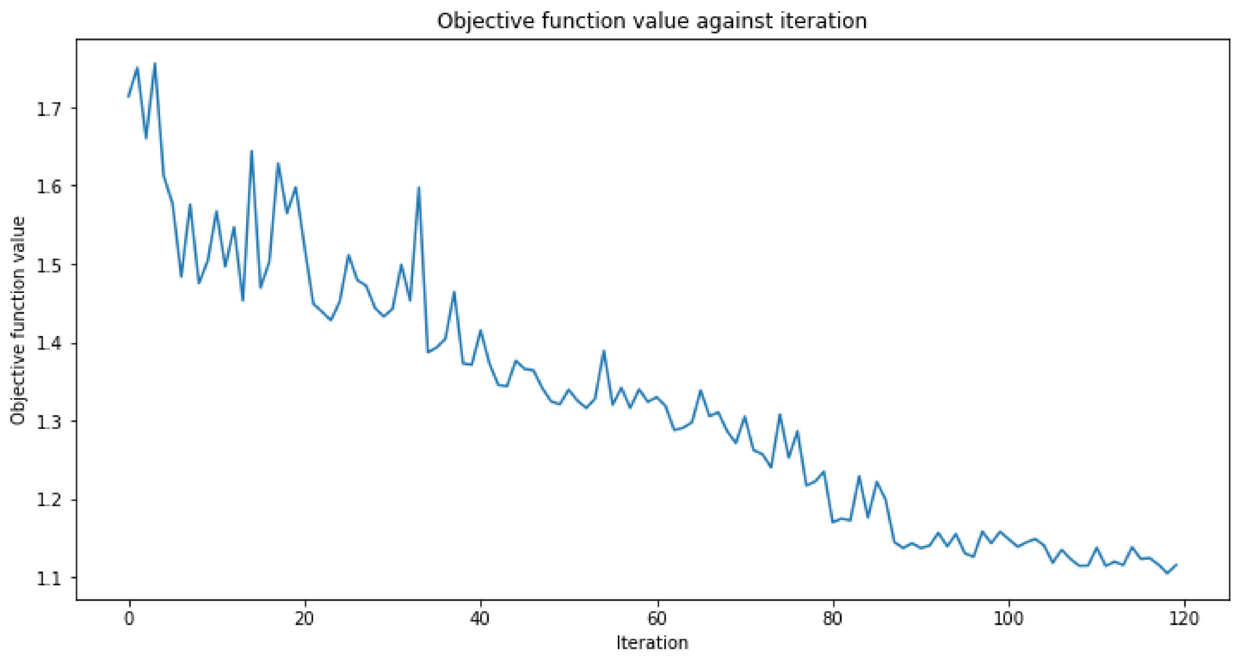
In [22]:
```
X_train, X_test, y_train, y_test = train_test_split(X_features, labels, test_size = 0.
                                                    random_state=algorithm_globals.ran
```

In [26]:
```
#initializing VQC object
vqc = VQC(feature_map=ZZ_feature_map,
          ansatz=variational_circ,
          optimizer=optimizer,
          quantum_instance=quantum_instance,
          initial_point=initial_point,
          callback=callback_graph)
```

We also want to have an idea about the elapsed time for training the model. That is why we start a timer.

In [27]:
```
start = time.time()
```

In [28]:
```
# create empty array for callback to store evaluations of the objective function
objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

# fit classifier to data
vqc.fit(X_train, y_train)

elapsed = time.time() - start
print(f"Fit in {elapsed}")

# return to default figsize
plt.rcParams["figure.figsize"] = (6, 4)

plt.show()
```



```
Fit in 240.79086327552795
```

4.9. Score the Classifier

In [29]:
```
print(f"Q train score: {vqc.score(X_train, y_train)}")
print(f"Q test score : {vqc.score(X_test, y_test)}")
```

```
Q train score: 0.825
Q test score : 0.8
```

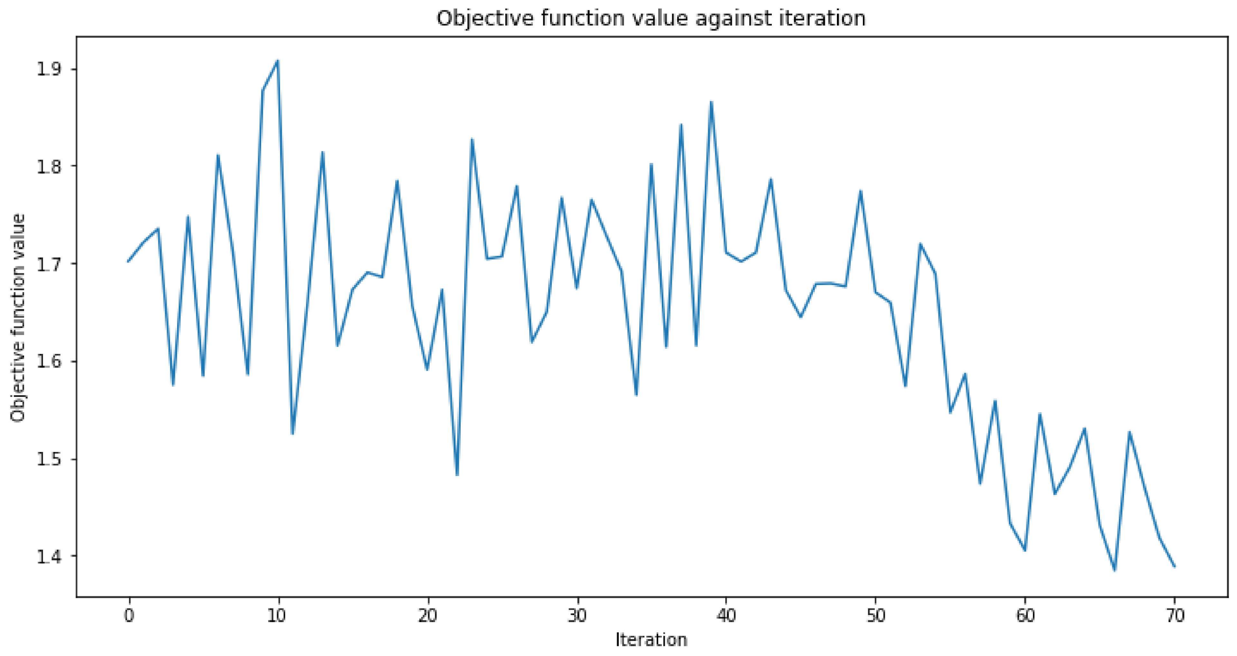
5. Additional Experiments

We have accomplished our initial set-up. Now we want to try out other settings and evaluate their impact.
We will write the code each time in one cell.
As such, the code each time makes up an integrated logical unit.

5.1. Alternative Optimizers

We will make use of the following optimizers:
https://qiskit.org/documentation/stubs/qiskit.algorithms.optimizers.html?highlight=qiskit%20algorithms%20optimizers#module-qiskit.algorithms.optimizers

In [30]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import time

iris_data = load_iris()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from qiskit import Aer, QuantumCircuit
from qiskit.utils import QuantumInstance, algorithm_globals

from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap
from qiskit.algorithms.optimizers import SPSA
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit_machine_learning.exceptions import QiskitMachineLearningError
from IPython.display import clear_output

optimizer = SPSA(maxiter=10)
#optimizer = COBYLA(maxiter=120, tol=0.001)

def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()

mms = MinMaxScaler()
X_features = iris_data['data']
X_features = mms.fit_transform(X_features)
labels = iris_data['target']
labels = OneHotEncoder(sparse=False).fit_transform(labels.reshape(-1, 1))
algorithm_globals.random_seed = 123

feature_dim = 4
ZZ_feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=1, entanglement='ful
num_qubits = feature_dim
variational_circ = RealAmplitudes(num_qubits, entanglement='full', reps=4)

backend = Aer.get_backend('statevector_simulator')
quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=algorithm_globa
        seed_transpiler=algorithm_globals.random_seed)
initial_point = algorithm_globals.random.random(variational_circ.num_parameters)

X_train, X_test, y_train, y_test = train_test_split(X_features, labels, test_size = 0.

vqc = VQC(feature_map=ZZ_feature_map,
          ansatz=variational_circ,
          optimizer=optimizer,
          quantum_instance=quantum_instance,
          initial_point=initial_point,
          callback=callback_graph)
start = time.time()

objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

vqc.fit(X_train, y_train)

elapsed = time.time() - start
print(f"Fit in {elapsed}")

plt.rcParams["figure.figsize"] = (6, 4)
plt.show()

print(f"Q train score: {vqc.score(X_train, y_train)}")
print(f"Q test score : {vqc.score(X_test, y_test)}")
```

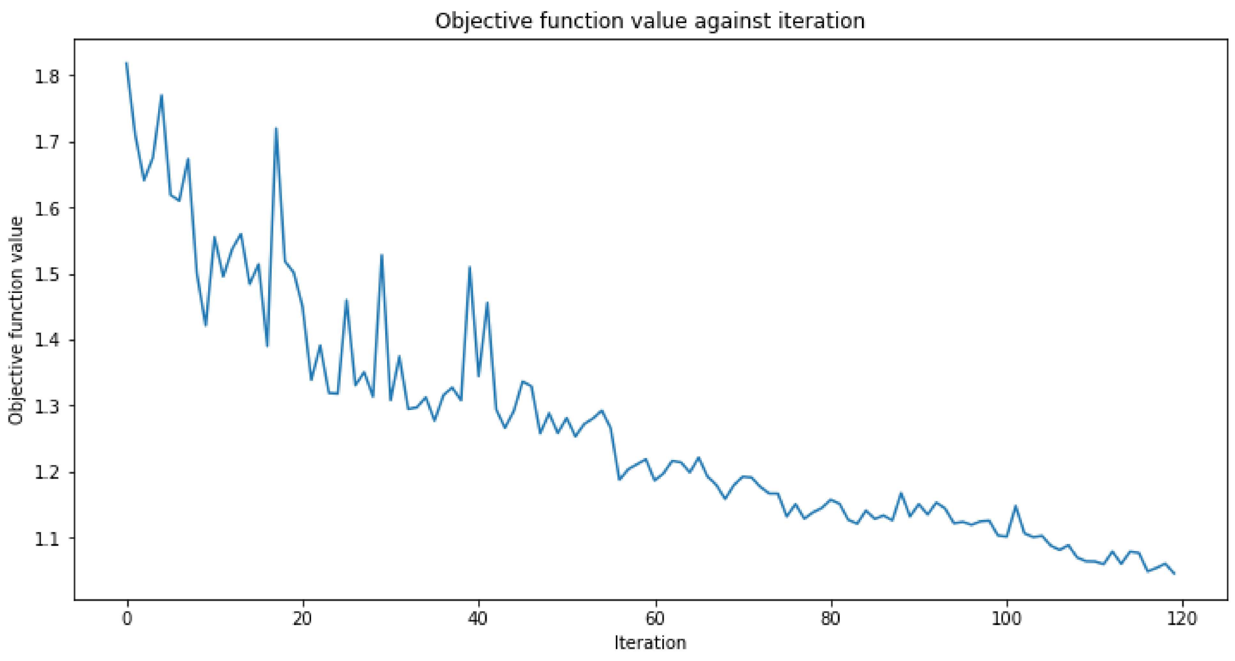


```
Fit in 138.7473428249359
Q train score: 0.575
Q test score : 0.5333333333333333
```

5.2. Other Parameters for the Feature Map

In [ ]:
```
We will make use of 2 repetitions and circular entanglement for the Feature Map.<br>
We make use of the COBYLA Optimizer because it performed much better than SPSA.
```

In [32]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import time

iris_data = load_iris()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from qiskit import Aer, QuantumCircuit
from qiskit.utils import QuantumInstance, algorithm_globals

from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap
from qiskit.algorithms.optimizers import COBYLA
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit_machine_learning.exceptions import QiskitMachineLearningError
from IPython.display import clear_output

optimizer = COBYLA(maxiter=120, tol=0.001)

def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()

mms = MinMaxScaler()
X_features = iris_data['data']
X_features = mms.fit_transform(X_features)
labels = iris_data['target']
labels = OneHotEncoder(sparse=False).fit_transform(labels.reshape(-1, 1))
algorithm_globals.random_seed = 123

feature_dim = 4
ZZ_feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=2, entanglement='cir
num_qubits = feature_dim
variational_circ = RealAmplitudes(num_qubits, entanglement='full', reps=4)

backend = Aer.get_backend('statevector_simulator')
quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=algorithm_globa
        seed_transpiler=algorithm_globals.random_seed)
initial_point = algorithm_globals.random.random(variational_circ.num_parameters)

X_train, X_test, y_train, y_test = train_test_split(X_features, labels, test_size = 0.

vqc = VQC(feature_map=ZZ_feature_map,
          ansatz=variational_circ,
          optimizer=optimizer,
          quantum_instance=quantum_instance,
          initial_point=initial_point,
          callback=callback_graph)
start = time.time()

objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

vqc.fit(X_train, y_train)

elapsed = time.time() - start
print(f"Fit in {elapsed}")

plt.rcParams["figure.figsize"] = (6, 4)

plt.show()

print(f"Q train score: {vqc.score(X_train, y_train)}")
print(f"Q test score : {vqc.score(X_test, y_test)}")
```



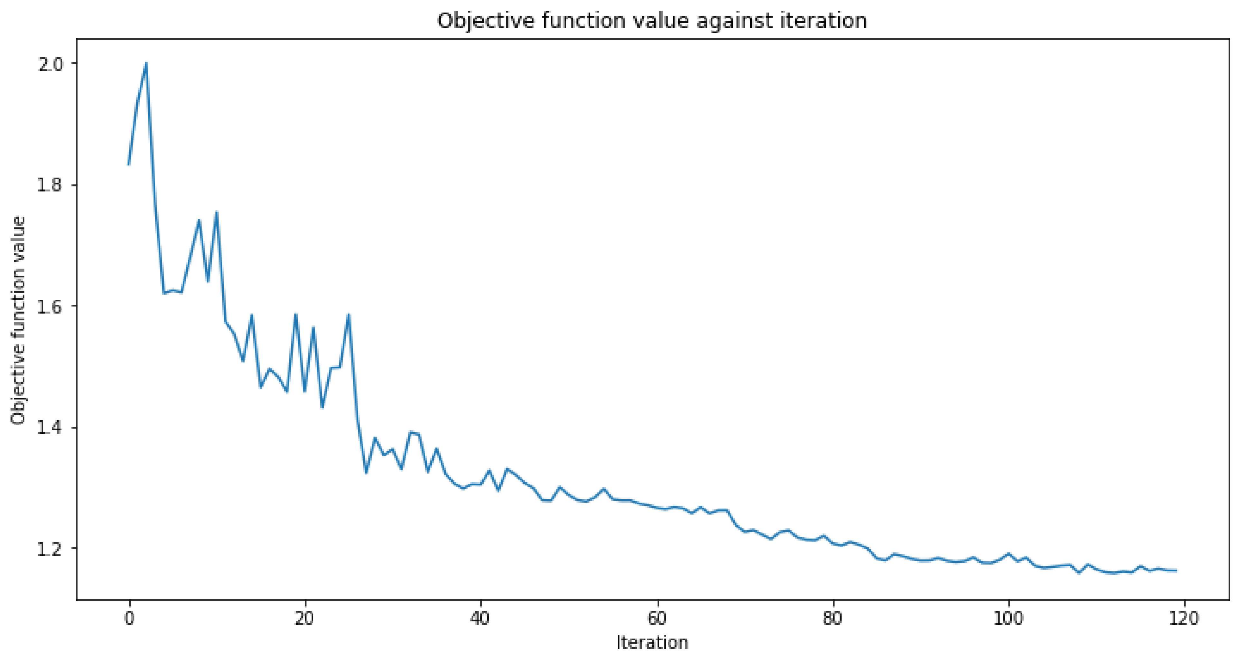
```
Fit in 210.19560956954956
Q train score: 0.8083333333333333
Q test score : 0.7333333333333333
```

5.3. Other Paramaters for the Ansatz

In [33]:
```
We will make use of circular entanglement and 3 repetitions for the Ansatz.<br>
We make use of the COBYLA Optimizer.
```

```
  Input In [33]
    We will make use of circular entanglement and 3 repetitions for the Ansatz.<br>
       ^
SyntaxError: invalid syntax
```

In [34]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import time

iris_data = load_iris()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from qiskit import Aer, QuantumCircuit
from qiskit.utils import QuantumInstance, algorithm_globals

from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap
from qiskit.algorithms.optimizers import COBYLA
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit_machine_learning.exceptions import QiskitMachineLearningError
from IPython.display import clear_output

optimizer = COBYLA(maxiter=120, tol=0.001)

def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()

mms = MinMaxScaler()
X_features = iris_data['data']
X_features = mms.fit_transform(X_features)
labels = iris_data['target']
labels = OneHotEncoder(sparse=False).fit_transform(labels.reshape(-1, 1))
algorithm_globals.random_seed = 123

feature_dim = 4
ZZ_feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=1, entanglement='ful
num_qubits = feature_dim
variational_circ = RealAmplitudes(num_qubits, entanglement='circular', reps=3)

backend = Aer.get_backend('statevector_simulator')
quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=algorithm_globa
        seed_transpiler=algorithm_globals.random_seed)
initial_point = algorithm_globals.random.random(variational_circ.num_parameters)

X_train, X_test, y_train, y_test = train_test_split(X_features, labels, test_size = 0.

vqc = VQC(feature_map=ZZ_feature_map,
          ansatz=variational_circ,
          optimizer=optimizer,
          quantum_instance=quantum_instance,
          initial_point=initial_point,
          callback=callback_graph)
start = time.time()

objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

vqc.fit(X_train, y_train)

elapsed = time.time() - start
print(f"Fit in {elapsed}")

plt.rcParams["figure.figsize"] = (6, 4)

plt.show()

print(f"Q train score: {vqc.score(X_train, y_train)}")
print(f"Q test score : {vqc.score(X_test, y_test)}")
```

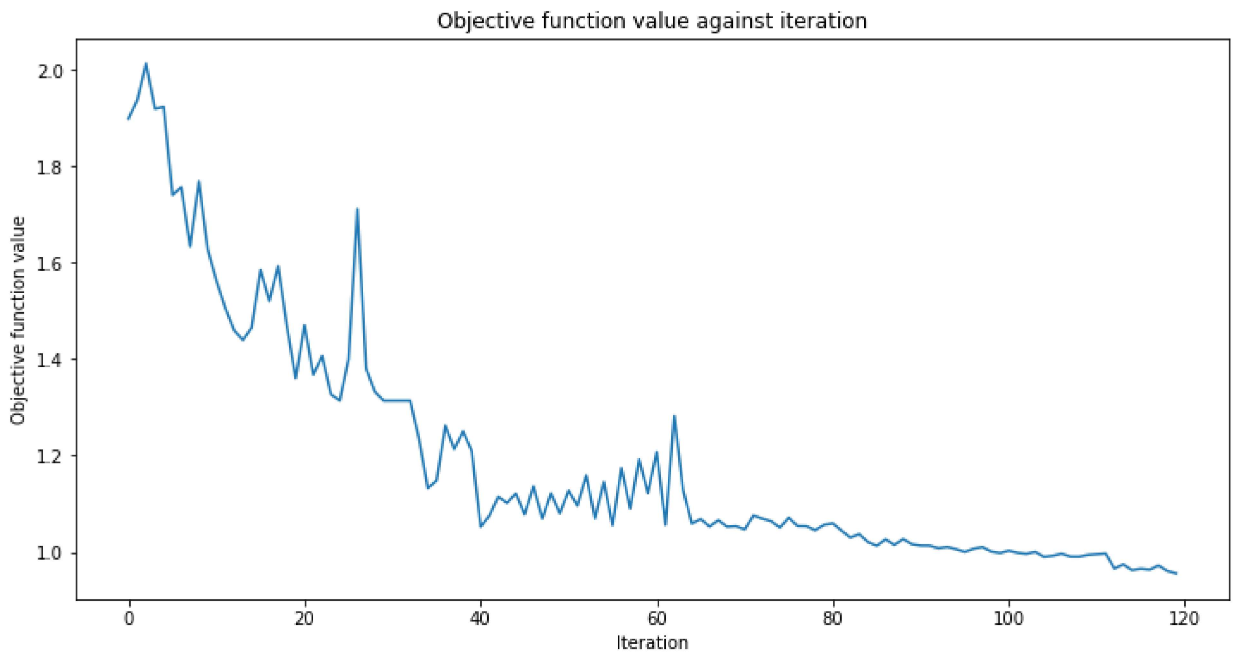


```
Fit in 216.05357551574707
Q train score: 0.7666666666666667
Q test score : 0.7
```

5.4. Other Ansatz Function

We make use of another Ansatz: EfficientSU2. We reuse our original settings for the Feature Map. We also mak use of the COBYLA Optimizer.

In [38]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import time

iris_data = load_iris()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from qiskit import Aer, QuantumCircuit
from qiskit.utils import QuantumInstance, algorithm_globals

from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap, EfficientSU2
from qiskit.algorithms.optimizers import COBYLA
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit_machine_learning.exceptions import QiskitMachineLearningError
from IPython.display import clear_output

optimizer = COBYLA(maxiter=120, tol=0.001)

def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()

mms = MinMaxScaler()
X_features = iris_data['data']
X_features = mms.fit_transform(X_features)
labels = iris_data['target']
labels = OneHotEncoder(sparse=False).fit_transform(labels.reshape(-1, 1))
algorithm_globals.random_seed = 123

feature_dim = 4
ZZ_feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=1, entanglement='ful
num_qubits = feature_dim
variational_circ = EfficientSU2(num_qubits)

backend = Aer.get_backend('statevector_simulator')
quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=algorithm_globa
        seed_transpiler=algorithm_globals.random_seed)
initial_point = algorithm_globals.random.random(variational_circ.num_parameters)

X_train, X_test, y_train, y_test = train_test_split(X_features, labels, test_size = 0.

vqc = VQC(feature_map=ZZ_feature_map,
          ansatz=variational_circ,
          optimizer=optimizer,
          quantum_instance=quantum_instance,
          initial_point=initial_point,
          callback=callback_graph)
start = time.time()

objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

vqc.fit(X_train, y_train)

elapsed = time.time() - start
print(f"Fit in {elapsed}")

plt.rcParams["figure.figsize"] = (6, 4)

plt.show()

print(f"Q train score: {vqc.score(X_train, y_train)}")
print(f"Q test score : {vqc.score(X_test, y_test)}")
```

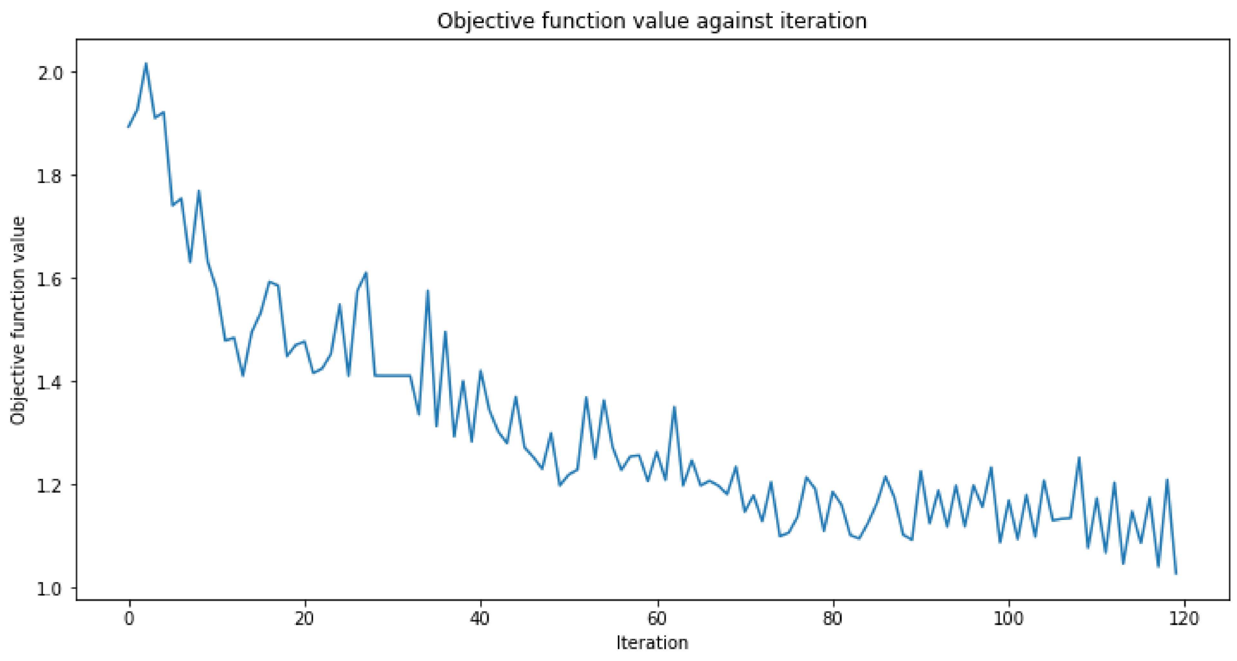


```
Fit in 260.9183466434479
Q train score: 0.8416666666666667
Q test score : 0.8333333333333334
```

5.5. Other Backend

We make use of another backend. We reuse the original parameter settings for the Optimizer, the Ansatz and the Feature Map.

In [39]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import time

iris_data = load_iris()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from qiskit import Aer, QuantumCircuit
from qiskit.utils import QuantumInstance, algorithm_globals

from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap, EfficientSU2
from qiskit.algorithms.optimizers import COBYLA
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit_machine_learning.exceptions import QiskitMachineLearningError
from IPython.display import clear_output

optimizer = COBYLA(maxiter=120, tol=0.001)

def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()

mms = MinMaxScaler()
X_features = iris_data['data']
X_features = mms.fit_transform(X_features)
labels = iris_data['target']
labels = OneHotEncoder(sparse=False).fit_transform(labels.reshape(-1, 1))
algorithm_globals.random_seed = 123

feature_dim = 4
ZZ_feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=1, entanglement='ful
num_qubits = feature_dim
variational_circ = EfficientSU2(num_qubits)

backend = Aer.get_backend('aer_simulator')
quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=algorithm_globa
        seed_transpiler=algorithm_globals.random_seed)
initial_point = algorithm_globals.random.random(variational_circ.num_parameters)

X_train, X_test, y_train, y_test = train_test_split(X_features, labels, test_size = 0.

vqc = VQC(feature_map=ZZ_feature_map,
          ansatz=variational_circ,
          optimizer=optimizer,
          quantum_instance=quantum_instance,
          initial_point=initial_point,
          callback=callback_graph)
start = time.time()

objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

vqc.fit(X_train, y_train)

elapsed = time.time() - start
print(f"Fit in {elapsed}")

plt.rcParams["figure.figsize"] = (6, 4)

plt.show()

print(f"Q train score: {vqc.score(X_train, y_train)}")
print(f"Q test score : {vqc.score(X_test, y_test)}")
```



```
Fit in 275.43174409866333
Q train score: 0.8416666666666667
Q test score : 0.6666666666666666
```

6. Some Conclusions

We tried out different parameters, without combining multiple parameter changes. When sticking to just one parameter change we find that the best results are achieved as follows:

- COBYLA as optmizer
- ZZFeatureMap as Feature Map with 1 repetition and full entanglement
- EfficientSU2 as Ansatz
- Statevector_simulator as backend

Of course, a wealth of more variations are possible !

In [94]:
```
import qiskit.tools.jupyter

%qiskit_version_table
%qiskit_copyright
```

Version Information

Qiskit Software	Version
`qiskit-terra`	0.20.2
`qiskit-aer`	0.10.4
`qiskit-ignis`	0.7.1
`qiskit-ibmq-provider`	0.19.1
`qiskit`	0.36.2
`qiskit-machine-learning`	0.4.0
System information	
Python version	3.9.12
Python compiler	MSC v.1916 64 bit (AMD64)
Python build	main, Apr 4 2022 05:22:27
OS	Windows
CPUs	6
Memory (Gb)	31.71529769897461
	Sun Jun 05 18:50:57 2022 Romance Daylight Time

This code is a part of Qiskit

© Copyright IBM 2017, 2022.

This code is licensed under the Apache License, Version 2.0. You may obtain a copy of this license in the LICENSE.txt file in the root directory of this source tree or at http://www.apache.org/licenses/LICENSE-2.0.

Any modifications or derivative works of this code must retain this copyright notice, and modified files need to carry a notice indicating that they have been altered from the originals.

Mentee

Eric Michiels

Mentor

Anton Dekusar

Date

May 2022