My Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

IsingBornMachine

Implementation of Quantum Ising Born Machine using Rigetti Forest Platform, with two approaches:

1. Training using MMD as a Cost function.
2. Training using Stein Discrepancy as a Cost function

Both of the above have the option to run using a 'Quantum-Hard' Kernel function

To run the, PyQuil is needed, plus a user code which can be gotten by signing up to use the Rigetti simulator online:

Follow instructions online: `http://docs.rigetti.com/en/stable/start.html` This will allow you to download the Rigetti SDK which includes a compiler and a Quantum Virtual Machine

Also the standard packages, numpy, matplotlib, etc.

In the current form of the above codes, it is necessary to run some functions in file_operations_out.py to generate the data and pre-compute the quantum kernel etc.

MAXIMUM MEAN DISCREPANCY

INSTRUCTIONS FOR USE

run using:

```
python run_and_compare.py input.txt
```

Where input.txt should look like:

N_epochs N_qubits N_trials learning_rate_one learning_rate_two

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Namespace Documentation

3.1 auxiliary_functions Namespace Reference

some additional useful functions

Functions

- def **ConvertToString** (index, N_qubits)
- def StringToList (string)
- def SampleListToArray (original_samples_list, N_qubits)

  Convert list to array.
- def SampleArrayToList (sample_array)
- def EmpiricalDist (samples, N_v, arg)
- def TotalVariationCost (dict_one, dict_two)
- def ConvertStringToVector (string)
- def L2NormForStrings (string1, string2)
- def TrainTestPartition (samples)

  This function partitions an array of samples into a training array and a test array.
- def MiniBatchSplit (samples, batch_size)

3.1.1 Detailed Description

some additional useful functions

A collection of sever additional function useful during the running of the code.

3.1.2 Function Documentation

3.1.2.1 ConvertStringToVector()

```
def auxiliary_functions.ConvertStringToVector (
              string )
```

```
This function converts a string to a np array
```

3.1.2.2 EmpiricalDist()

```
def auxiliary_functions.EmpiricalDist (
              samples,
              N_v,
              arg )
```

```
This method outputs the empirical probability distribution given samples in a numpy array
as a dictionary, with keys as outcomes, and values as probabilities
```

3.1.2.3 L2NormForStrings()

```
def auxiliary_functions.L2NormForStrings (
              string1,
              string2 )
```

```
This function computes the L2 norm between two strings
```

3.1.2.4 MiniBatchSplit()

```
def auxiliary_functions.MiniBatchSplit (
              samples,
              batch_size )
```

```
This function takes the first \'batch_size\' samples out of the full sample set
```

3.1.2.5 SampleArrayToList()

```
def auxiliary_functions.SampleArrayToList (
              sample_array )
```

```
This function converts a np.array where rows are samples
    into a list of length N_samples
```

3.1.2.6 SampleListToArray()

```
def auxiliary_functions.SampleListToArray (
              original_samples_list,
              N_qubits )
```

Convert list to array.

Parameters

in	*original_samples_list*	The original list
in	*N_qubits*	The number of qubits
out	*sample_array*	The list converted into an array

return Converted list

```
This function converts a list of strings, into a numpy array, where
    each [i,j] element of the new array is the jth bit of the ith string
```

3.1.2.7 StringToList()

```
def auxiliary_functions.StringToList (
              string )
```

```
This kernel converts a binary string to a list of integers
```

3.1.2.8 TotalVariationCost()

```
def auxiliary_functions.TotalVariationCost (
              dict_one,
              dict_two )
```

```
This Function computes the variation distace between two distributions
```

3.1.2.9 TrainTestPartition()

```
def auxiliary_functions.TrainTestPartition (
              samples )
```

This function partitions an array of samples into a training array and a test array.

The last 20% of the original set is used for testing

Parameters

in	*samples*	A list of samples
out	*train_test*	array of lists

return Split array

3.2 file_operations_in Namespace Reference

import functions

Functions

- def **FileLoad** (file)
- def DataDictFromFile (N_qubits, N_samples)

  Reads data dictionary from file.
- def DataImport (approx, N_qubits, N_data_samples, stein_approx)

  Returns relevant data.
- def KernelDictFromFile (N_qubits, N_samples, kernel_choice)

  Reads kernel dictionary from file.
- def **ParamsFromFile** (N_qubits)

3.2.1 Detailed Description

import functions

A collection of functions for imported pre-computed data

3.2.2 Function Documentation

3.2.2.1 DataDictFromFile()

```
def file_operations_in.DataDictFromFile (
              N_qubits,
              N_samples )
```

Reads data dictionary from file.

Parameters

in	*N_qubits*	The number of qubits
in	*N_samples*	The number of samples

Returns

A dictionary containing the appropriate data

3.2.2.2 DataImport()

```
def file_operations_in.DataImport (
              approx,
              N_qubits,
              N_data_samples,
              stein_approx )
```

Returns relevant data.

Parameters

in	*approx*	The approximation type
in	*N_qubits*	The number of qubits
in	*N_data_samples*	The number of data samples
in	*stein_approx*	The approximation type
out	*data_samples*	The requested list of samples
out	*data_exact_dict*	The requested dictionary of exact samples

Returns

Requested data

3.3 param_init Namespace Reference

Initialise some inputted variables.

Functions

- def **HadamardToAll** (prog, N_qubits)
- def NetworkParams (N_qubits)

  Initialise weights and biases as random.
- def StateInit (N_qubits, circuit_params, p, q, r, s, circuit_choice, control, sign)

3.3.1 Detailed Description

Initialise some inputted variables.

3.3.2 Function Documentation

3.3.2.1 NetworkParams()

```
def param_init.NetworkParams (
              N_qubits )
```

Initialise weights and biases as random.

This function computes the initial parameter values, J, b randomly chosen on interval [0, pi/4], gamma_x, gamma_y set to constant = pi/4 if untrained

Parameters

in	*N_qubits*	The number of qubits

Returns

initialised parameters

3.3.2.2 StateInit()

```
def param_init.StateInit (
        N_qubits,
        circuit_params,
        p,
        q,
        r,
        s,
        circuit_choice,
        control,
        sign )
```

```
This function computes the state produced after the given circuit, either QAOA, IQP, or IQPy,
depending on the value of circuit_choice.
```

3.4 run_and_compare Namespace Reference

This is the main module for this project.

Functions

- def get_inputs (file_name)

  This function gathers inputs from file.
- def **SaveAnimation** (N_trials, framespersec, fig, N_epochs, N_qubits, learning_rate, N_born_samples, cost_func, kernel_type, approx, data_exact_dict1, data_exact_dict2, born_probs_list1, axs)
- def **PlotAnimate** (N_trials, N_qubits, N_epochs, learning_rate, N_born_samples, cost_func, kernel_type, approx, data_exact_dict1, data_exact_dict2)
- def **animate** (i, N_trials, N_qubits, learning_rate, N_born_samples, kernel_type, approx, data_exact_dict1, born_probs_list1, axs)
- def main ()

  This is the main function.

Variables

- list plot_colour = [ ]

  If kernel is to be computed exactly set N_kernel_samples = 'infinite'.

3.4.1 Detailed Description

This is the main module for this project.

More details.

3.4.2 Function Documentation

3.4.2.1 get_inputs()

```
def run_and_compare.get_inputs (
              file_name )
```

This function gathers inputs from file.

Parameters

in	*file_name*	name of file to gather inputs from
out	*N_epochs*	number of epochs
out	*N_qubits*	number of qubits
out	*N_trials*	number of trials
out	*learning_rate_one*	first learning rate
out	*learning_rate_two*	second learning rate

Returns

listed parameters

3.4.3 Variable Documentation

3.4.3.1 plot_colour

```
list run_and_compare.plot_colour = [ ]
```

If kernel is to be computed exactly set N_kernel_samples = 'infinite'.

list of colours

Index