Quantum ML Notes

VT NSI

Alex Klee

July 26, 2024

1 Introduction

This document describes exploratory research conducted this summer in two new areas of quantum technology: entropy-based quantum optimization systems and uniform quantum random number generators (uQRNG). My work focused on evaluating the performance and applicability of these systems for a variety of problems, including machine learning tasks, cryptographic computations, and statistical randomness testing.

To do this, I had cloud-based access to entropy quantum computers (Dirac-1 and Dirac-3 from Quantum Computing Inc.), and these systems were tested on several optimization-based tasks, usually developed in Python. The goal was to investigate how specific classical problems could be mapped to quantum formulations and whether these mappings provided performance or accuracy benefits. Meanwhile, the uQRNG system was analyzed for randomness quality and considered for use in cryptographic applications where unpredictability is essential.

This report is a compilation of some of my notes and research in more interesting and promising applications of these systems.

2 Applications for Dirac: Machine Learning

The Dirac hardware offers a versatile set of tools and techniques applicable across different domains. This section explores some of the applications where Dirac can be applied, with some cases being more useful than others. Machine learning models typically involves algorithms and complex statistical problems to achieve their goal. However, many machine learning problems, especially those involving large-scale data, can be computationally intensive and challenging for classical computers. The Dirac systems can significantly contribute to some problems within machine learning, providing more efficient and potentially scalable solutions. Some of these solutions have already been explored, as seen on the QCi knowledge base, while many possibilities are still untouched.

2.0.1 Classification

One area where the Dirac computers fit more naturally is classification, due to the nature of some classification models and their use of optimization problems. Boosting is one method that was already implemented by QCi. In the eqc models package being currently developed, there is a base class for a classifier, and I used this to implement a version of a support vector machine, loosely based on the classifierqsvm class in the package. I tested both the QBoost model and the qSVM model vs two classical versions of the same algorithms, with both SVMs using a RBF kernel. The data is from the Penn machine learning benchmark.

Accuracy of Classifiers	No imbalance		Low imbalance		More imbalance	
	Test	Train	Test	Train	Test	Train
QSVM	92.3%	100%	96.9%	95.3%	81.8%	84.5%
scikit-learn SVM	73.1%	69.6%	84.4%	92.2%	90.2%	89.3%
QBoost	84.6%	88.2%	93.8%	99.2%	77.2%	100%
AdaBoost	76.9%	72.5%	93.8%	89.8%	77.2%	100%

While these results are promising, I could only test it on the smaller datasets from the PMLB list. This is mainly due to the large portion of additional polynomial terms needed to satisfy the lengthy constraints for both the SVM and the boosting formulation. For example, the hamiltonian for a dataset with only 120 data points could have over 3000 terms, despite only having two features. This is because of the strict summation constraint on Dirac-3, as opposed to the natural inequality constraints of the support vector machine algorithm.

2.0.2 Singular Value Decomposition

Singular Value Decomposition is a type matrix decomposition approach that is useful in matrix reduction by generalizing the eigendecomposition of a square matrix (same number of columns and rows) to any matrix. It will help us to simplify matrix calculations. Mathematically, it looks like

$$X = U\Sigma V^T$$

An estimation of the first principal component has a natural mapping to the Dirac-1 QUBO formulation, as the equation already takes the form

$$\min_q -q^T G q$$

With this, singular value decomposition problems can easily be solved on Dirac-1, with a similar example shown on the QCi knowledge base under feature selection.

2.0.3 Tensor Decomposition

A tensor is a multidimensional array. More formally, an Nway or Nth-order tensor is an element of the tensor product of N vector spaces, each of which has its own coordinate system. A first-order tensor is a vector, a second-order tensor is a matrix, and tensors of order three or higher are called higher-order tensors. The order of a tensor is the number of dimensions, also known as ways or modes.

Tensor decomposition is a mathematical technique used to break down a tensor into simpler, more manageable components. A three dimensional example approximating $\boldsymbol{\mathcal{X}}$ could look like

$$\boldsymbol{\mathcal{X}} \approx \hat{\boldsymbol{\mathcal{X}}} = \sum_{r=1}^{R} u_r \circ v_r \circ w_r$$

where u, v, and w are vectors and $\circ$ is the outer product. This is called the CANDECOMP (or Parafac) decomposition of a tensor, and can be solved using Alternating Least Squares. Alternating Least Square (ALS) is a matrix factorization algorithm that normally runs in a parallel fashion. The goal of ALS given a matrix R of size $m \times n$ is to factor it into U of size $m \times k$ and V of size $n \times k$. This can also be thought of as an optimization problem:

$$\min_{U,V} \|R - UV^T\|_F^2$$

where $\|v\|_F$ is the Frobenius norm. Due to the optimization portion here, I attempted to find a mapping of this type of problem to the hamiltonian required for Dirac-3. It comes pretty close to resembling a base formulation of the Hamiltonian for Dirac-3, however the math I was doing ended up with terms with 6th order interactivity, one too many for Dirac-3. With that being said, I do believe it is possible to find that mapping.

Tucker decomposition is another form of tensor decomposition that can be thought of as a type of higher-order SVD/PCA in which $\boldsymbol{\mathcal{X}}$ is decomposed into a core tensor $\boldsymbol{\mathcal{G}}$ that is transformed by a matrix along each dimension. It decomposes a tensor into a core tensor multiplied by a matrix along each mode. In the three-way case where $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$, we have

$$\boldsymbol{\mathcal{X}} \approx \hat{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr}\, \mathbf{a}_p \circ \mathbf{b}_q \circ \mathbf{c}_r$$

where $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are the orthogonal factor matrices, and $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{P \times Q \times R}$ is the core tensor, containing important information about interactivity. The Tucker decomposition of a tensor can be solved as a non-convex optimization problem by finding the difference between the tensor and its approximation:

$$\min \|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}}\|_F^2$$

where F indicates the Frobenius norm. Unfortunately, I had the same difficulty here as I did with the CP/Parafac decomposition, the mathematical representation entails a sixth order term. However, due to its similarity to the Hamiltonian for Dirac-3, I do believe it is also possible to find a mapping to allow this type of problem to be solved using the Dirac-3.

2.1 Cryptograhpy

Outside of machine learning, I researched some applications of Dirac in cryptography, mostly in algebraic settings. Shor's algorithm is famous for factoring prime numbers on a theoretical quantum system, although the qubit manipulation required by Shor's algorithm is not feasible for the Dirac systems. However, Schnorr's algorithm, detailed here, actually demonstrates a different method for factoring primes that has the potential to run on the Dirac-3. A group of researchers in China in this paper demonstrated this algorithm's potential to be sped up by quantum technology, using QAOA, a quantum integer factorization algorithm. The main step in this algorithm, a lattice reduction, takes the form of a Hamiltonian

$$H_c = \left\| \mathbf{t} - \sum_{i=1}^{n} \hat{x}_i \mathbf{d}_i - \mathbf{b}_{op} \right\|^2 = \sum_{j=1}^{n+1} \left| t_j - \sum_{i=1}^{n} \hat{x}_i d_{i,j} - b_{op}^j \right|^2$$

This problem is solvable on Dirac-3, however, in my testing I did not find any significant advantage to this algorithm over its classical counterparts, and did not find any sort of scalability remotely close to the research paper utilizing QAOA.

2.2 Inequality constraint for Dirac-1

QCi's Dirac-1 solves unconstrained as well as linearly constrained binary optimization problems. However, these are only strict constraints. Some problems including a common knapsack problem are constrained by inequalities.

```
QT = np.pad(Q, ((0, N), (0, N)))  # adding the extra slack variables at the end of the Q matrix
n_qubits = len(QT)
lambd = 2  # choose a lambda parameter enough large for the constraint to always be fulfilled
# Adding the terms for the penalty term
for i in range(len(QT)):
    QT[i, i] += lambd * weights[i] * (weights[i] - 2 * maximum_weight)
    for j in range(i + 1, len(QT)):
        QT[i, j] += lambd * weights[i] * weights[j]
    for j in range(i + 1, len(QT)):
        QT[j, i] += lambd * weights[i] * weights[j]
offset = lambd * maximum_weight**2
```

The code above allows a problem with an inequality summation constraint to be run on Dirac-1, with the job type sample-qubo. In this code, the matrix Q contains the coefficients from the problem, and the 'weights' matrix contains the coefficient of the variable in the constraint. An example optimization problem with this formulation could be

$$E = 3x_1 + 5x_2 + 8x_3 \quad \text{s.t.} \quad 5x_1 + x_3 \leq 10$$

This type of problem is naturally suited for Dirac-1, and performs well.

3 Testing and applications of uQRNG

As opposed to my explorative approach in researching Dirac-1 and Dirac-3, my experience with the uQRNG was more analytical.

3.1 Testing

I tested the quantum random numbers with both the NIST test suite and the Dieharder tests, comparing them to multiple PRNGs and other quantum random numbers.

CryptoRandom is a popular cryptographically-secure PRNG, qRandom is a set of random numbers generated using IBM's qiskit superconducting quantum computer. The QRNG4 denotes the 4th run of QCi's uQRNG, 1 million random bits. The results from the NIST tests are as follows, with the *proportion* category representing the number of test failures out of 100 runs.

	CryptoRandom (example prng)		**qRandom** (IBM Qiskit)		**QRNG4** (QCi)	
	P-Value	Proportion	P-Value	Proportion	P-Value	Proportion
Frequency	0.595549	99	0.153763	100	0.171867	100
BlockFrequency	0.224821	99	0.514124	98	0.719747	98
CumulativeSums	0.883171	99	0.946308	100	0.759756	100
CumulativeSums	0.102526	98	0.657933	100	0.350485	100
Runs	0.935716	99	0.037566	100	0.987896	100
LongestRun	0.678686	97	0.319084	100	0.851383	99
FFT	0.494392	98	0.037566	98	0.514124	100
Serial	0.534146	100	0.637119	99	0.037566	100
Serial	0.699313	98	0.062821	100	0.037566	100

As for the Dieharder tests, I was unable to run all of them due to the billions of bits required by some of the later tests, and many tests actually rewound the random number up to 50 times. Even with the rewinding, most of the random numbers still passed the tests. The uQRNG passed 55 out of 56 tests, with weak results for 3 of the passing runs. These results show clearly that the numbers are truly random, and suitable for cryptography and other applications.