Review of QAOA

13.0

Quantum Approximate Optimization for constrained optimization:

Find $z \in \{0,1\}^n$ to minimize: $C(z) = \sum_i C_i(z)$ where

$$C_i(z) = \begin{cases} 1 & \text{if } z \text{ satisfies constraint } C_i \\ 0 & \text{otherwise} \end{cases}$$

Ex MAXCUT is an instance of a constrained optimization problem.

Define a graph G s.t. $i \in G$ is a vertex of the graph and

$(i,j) \in E(G)$ is an edge from vertex i to j in the graph

Let $C(z) = \langle z | \hat{C} | z \rangle$ where $\hat{C} = \sum_{(i,j)\in E(G)} \frac{1}{2}(\mathbb{1} - Z_i Z_j)$

↑ the "Cost Hamiltonian"

QAOA Algorithm

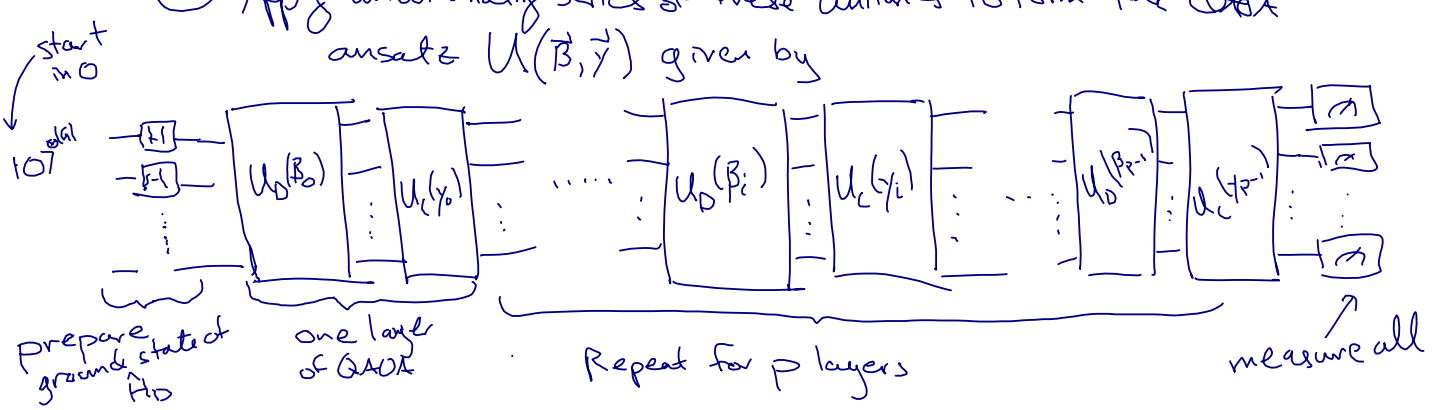
(1) Pick a "driver Hamiltonian" whose ground state is easy to prepare on your quantum computer. e.g.

$\hat{H}_D = \sum_{i \in G} X_i$ which has ground state $|H_D^0\rangle = \frac{1}{\sqrt{2^{|G|}}} \sum_{z \in \{0,1\}^{|G|}} |z\rangle$

(2) Construct parameterized unitaries corresponding to the cost and driver Hamiltonians

$$U_D(\beta) = e^{-i\beta \hat{H}_D} \qquad U_C(\gamma) = e^{-i\gamma \hat{C}}$$

(3) Apply an alternating series of these unitaries to form the QAOA ansatz $U(\vec{\beta}, \vec{\gamma})$ given by



(4) Sample a set of bitstrings $Z(\vec{\beta},\vec{\gamma})$ from running $U(\vec{\beta},\vec{\gamma})$.

Calculate the score as the expected value

$$E(\vec{\beta},\vec{\gamma}) = \frac{1}{|Z(\vec{\beta},\vec{\gamma})|}\sum_{a\in Z(\vec{\beta},\vec{\gamma})} \langle a|\hat{C}|a\rangle$$

(5) Run this in a minimization loop to choose a new $\vec{\beta}',\vec{\gamma}'$ to decrease $E(\vec{\beta},\vec{\gamma})$. Then repeat from #3

* Note it is also a good idea to store the "best" bitstring z that has been sampled so far.

(6) Return the best bitstring after some number of iterations

Implementing exponentiations of Paulis

In order to program this ansatz we need a way to turn the cost and driver Hamiltonians into quantum gates from our gate set. Both $\hat{H}_D$ and $\hat{C}$ are given as sums of products of Pauli matrices.

If we can exponentiate sums of products of Paulis then we can exponentiate any Hermitian matrix. We will show a technique to do this.

Case (i) $\hat{H} = Z_j$ then $U = e^{-i\alpha Z_j} = \underbrace{I\otimes\cdots\otimes}_{\text{qubits } j \text{ to } N}\begin{bmatrix} e^{-i\alpha} & 0 \\ 0 & e^{i\alpha}\end{bmatrix}\underbrace{\otimes\cdots\otimes I}_{\text{qubits } 0 \text{ to } j} = RZ(2\alpha)_j$

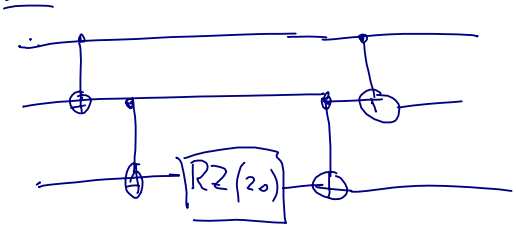
The exponent of a diagonal matrix is easy

up to a global phase

Case (ii) $\hat{H} = \alpha\bigotimes_{j\in M} Z_j$ some $|M|$ products of Z. We saw when studying error correction that this operator labels the parity of the qubits. Thus:

$U \Rightarrow$ for i in enumerate(M[:-1]):
    CNOT(M[i], M[i+1])
RZ(α) M[-1]
for i in enumerate[reverse(M[:-1])]:
    CNOT(M[i], M[i+1])

Ex $Z_0 Z_1 Z_2$



Case (iii) $H = \alpha \prod_{j \in M} P_j$ where P_j is one of the Pauli matrices.

Here we know that we can transform any Pauli into Z via unitaries e.g.

$$Z = HXH$$

and vice versa any Z can be turned into another Pauli e.g.

$$HZH = X$$

Thus we reduce to case(ii) and apply pre and post rotations to qubit j if $P_j \neq Z_j$

Case (iv) $H = H_1 + H_2$ This is where we can apply the Trotterization from the previous lecture. When H_1 and H_2 commute this is trivial $\Rightarrow$ we just do e^{-iH_1} then e^{-iH_2}.

When they do not commute we use the Trotter Suzuki formula to alternate them.

$$e^{i(H_1+H_2)} = \left(e^{iH_1/m} e^{iH_2/m} \right)^m .$$

These cases taken together suffice for any Hermitian H as it can be written as a sum of products of Paulis.

Let's look at how this is implemented in pyQuil.

Notebook

Intro to Quantum Machine Learning

A binary quantum classifier (Havlicek 2018)

Defn Classification Problem

Let training set T and test set S be sets of vectors $\vec{t}, \vec{s} \in \mathbb{R}^d$

Assume a labeling of both given by

$$m : T \cup S \to \{+1, -1\}$$

The training algorithm is only given the restriction of m to T

$$m_{|T} : T \to \{\pm 1\}$$
$$\vec{t} \mapsto m(\vec{t})$$

e.g the labelings of the training set.

The GOAL is to infer a map on the test set S,

$$\hat{m} : S \to \{\pm 1\}$$

That has the highest probability of agreement with $m_{|S}$, i.e. m on the test data.

For example this could be represented by an objective function

$$O(\hat{m}) = \frac{1}{|S|} \sum_{\vec{s}} \frac{1}{2}\left(1 - m(\vec{s})\hat{m}(\vec{s})\right)$$

which is to be minimized.

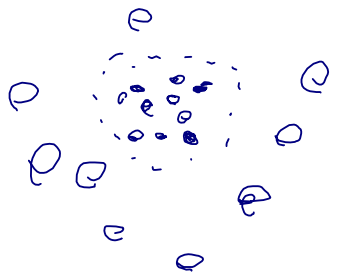
Support Vector Machines

Are a classical approach to this problem. Find the seperating "hyper-plane" that gives the best score. For example $d = 2$

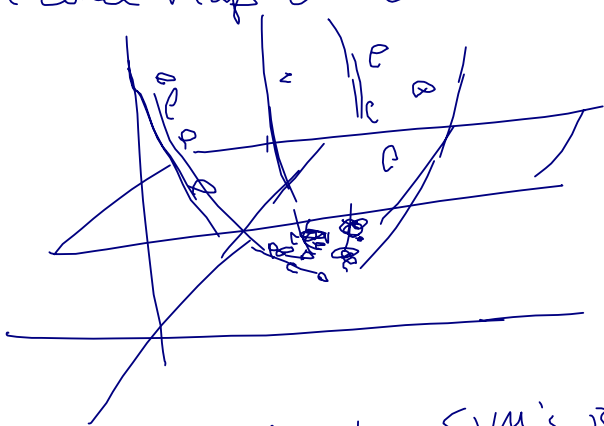
Simple example of an SVM where we choose some optimal

$$mx + b$$

However not all data can be split into a nice linear separation. For example:



Has no linear separator. However if we do a "kernal" transformation and maps our data into a higher dimensional space



← we can find a separating hyperplane.

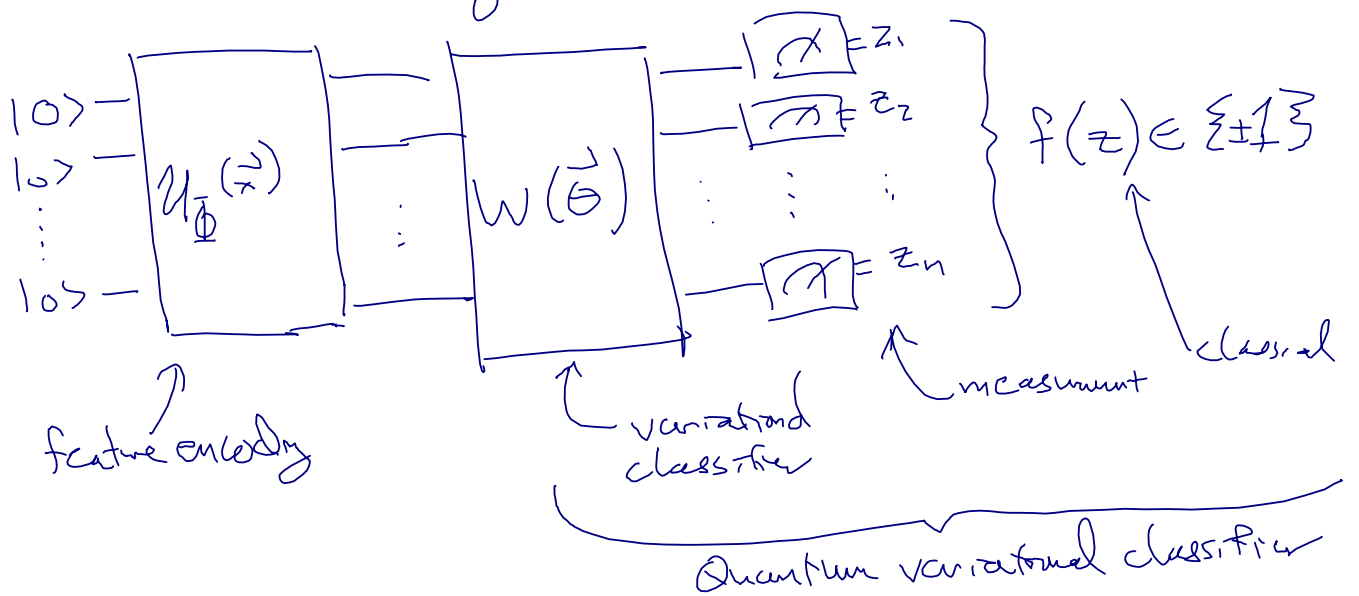
The idea behind many Quantum SVM's is to use the large vector space on a QPU to act as the kernal in a large vector space. This means we can compute new non-linear classifications.

There are many algorithm variants to do this on a QPU.
Today we will cover only one: Quantum Variational Classification

Quantum Variational Classification

Steps (1) Encode classical vectors from $\mathbb{R}^d$ into a wavefunction on N qubits e.g. a vector in $\mathbb{C}^{2^n}$

(2) Train QPU operations (unitaries and measurements) to classify the wavefunctions into labels $\in \{\pm 1\}$

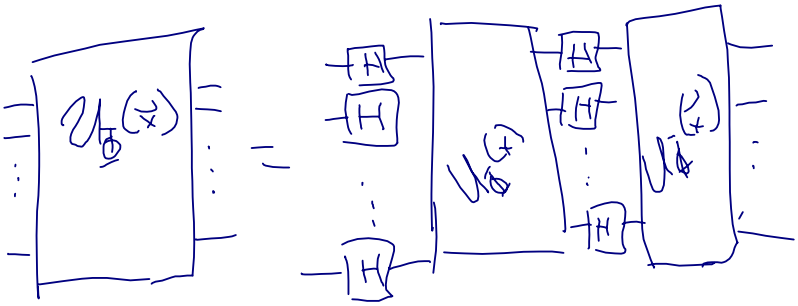


AND a choice of cost function and classical optimizer to find $\vec{\theta}^*$

Feature encoding

We want to do something that can show quantum advantage. Thus we should choose feature encodings that are hard to do classically. We want

$$U_\Phi : TUS \to \mathbb{C}^{2^n}$$
$$\vec{x} \mapsto |\Phi(\vec{x})\rangle = U_\Phi(\vec{x})|0\rangle^{\otimes n}$$



where $U_\Phi(x)$ is a diagonal unitary.

This is a good structure because calculating the inner product between wavefunctions generated by such circuits is in general #P-hard (Goldberg & Guo 2017)

$K(\vec{x},\vec{y}) = |\langle \Phi(\vec{x}) | \Phi(\vec{y}) \rangle|^2$ is #P-hard for this $U_\Phi(\vec{x})$

Ex Let $U_{\Phi(\vec{x})}$ be some circuit

```
for q in qubits:
    RZ(φ_i) q
for q in qubits:
    CPHASE(φ_i+q) q q.next()
```
#pseudocode

This moves us into a high dimensional space that is hard for classical computers to work in.

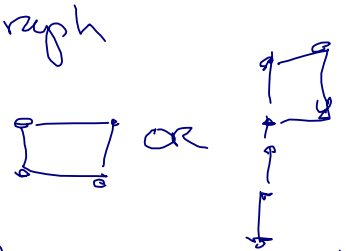
Quantum Variational Classification

Here we choose layers like in a VQE hardware ansatz.

$$W(\vec{\theta}) = U_{loc}^{(d)}(\theta_d) U_{ent} \dots U_{loc}^{(2)}(\theta_2) U_{ent} U_{loc}^{(1)}(\theta_1)$$

$U_{loc}^{(t)}(\theta_t)$ =
```
for q in qubits:
    RZ(θ^z_q,t) q
    RY(θ^y_q,t) q
```

U_{ent} =
```
for q in qubits:
    CZ q q.next()
```
} OR on hardware this can be the connectivity graph



With enough layers this is a universal quantum circuit.

Labeling

For any $\vec{x} \in TUS$ we can now run this to obtain

$$W(\vec{\theta})|\Phi(\vec{x})\rangle$$

Measure all qubits to obtain bitstrings $z_r \in \{0,1\}^N$ for R repetitions. We then apply a labeling function e.g.

$$f(z_r) = \begin{cases} +1 \text{ if } \sum_i^N z_r^{(i)} < N/2 \\ -1 \text{ otherwise} \end{cases}$$

i.e take the majority vote of the qubit outcomes

For some # of repetitions R let $r_+ =$ count of $+1$ labelings s.t. $r_- = R - r_+$

We then take the majority vote of the labels over the R repetitions to get the classification for $\vec{x}$.

Note a meta parameter bias can be introduced here for the final classification e.g.

$$\vec{x} \mapsto \begin{cases} +1 \text{ if } r_+ > r_- + b \\ -1 \text{ otherwise} \end{cases}$$

Prob of error $\Theta(e^{-\sqrt{R}})$

Thus we have

$$LABEL(\vec{\theta}, \vec{x}) \longrightarrow \{\pm 1\}$$

Train $\vec{\theta}$ by labeling over the train set.

Then test by running over the test set

Show cool kernels