We consider the problem of optimizing a quantum circuit to solve a certain problem. The idea is that we will fix the "architecture" of the quantum circuit. For our purposes, an architecture is an ordered list of quantum gates, where we only specify which qubits each gate applies to, but not what the gates themselves actually are. An architecture looks like a quantum circuit with none of the unitary gate boxes "filled in".

More formally, we define a *n-qubit Quantum Circuit Architecture* (n-qubit QCA for short) to be a tuple $\mathcal{A} = (A_1, \ldots, A_L)$ of subsets of $\{1, \ldots, n\}$; each A_i represents the set of qubits the corresponding quantum gate operates on.

We define an *assignment* for our QCA $\mathcal{A} = (A_1, \ldots, A_L)$ to be a choice of L unitary matrices $\mathcal{U} = (U_1, \ldots, U_L)$, such that each U_i acts on a space of size $2^{|A_i|}$. An assignment transforms our QCA into a real quantum circuit in the obvious way: for each placeholder A_i, one fills it in with the $|A_i|$-qubit quantum gate defined by U_i. Each U_i can be tensored with identity matrices on the qubits it does not operate on, in order to obtain a larger unitary $\widehat{U_i} \in U(2^n)$ describing its effect on the entire set of n qubits. Such a tensor product is cumbersome to write (after all, the qubits of U_i may not even be continuous), so we will hide behind this notation, and simply use $\widehat{U_i}$ to refer to this larger unitary. Thus, the behaviour of a QCA $\mathcal{A}$ with an assigment $\mathcal{U}$ can be described by the unitary $U := \widehat{U_1} \cdots \widehat{U_L}$.

For now, we will focus on the problem of binary classification: we have a set of n-bit strings, each labelled as true/false, and we wish to train a QCA that can classify them as accurately as possible. For now, we ignore the problem of coming up with a suitable QCA in the first place, and assume that we have both a QCA $\mathcal{A} = (A_1, \ldots, A_L)$, and that we also have decided on a measurement for taking the output from the QCA and deciding whether it's a true or false output. Any such measurement amounts to choosing orthogonal vectors $|\varphi_T\rangle$ and $|\varphi_F\rangle$, measuring the output state $|\varphi\rangle$ from our QCA, and outputting "True" with probability $\langle \varphi_T | \varphi \rangle$ and "False" with probability $\langle \varphi_F | \varphi \rangle$.

Thus, if we have an assignment $\mathcal{U}$ for our QCA, for a particular input $|\psi\rangle$ (say, with truth value "True"), our chance of being wrong is:

$$|\langle \varphi_F | U | \psi \rangle|^2 = |\langle \varphi_F | \widehat{U_1} \cdots \widehat{U_n} | \psi \rangle|^2$$

We wish to minimize this error chance (or perhaps, its mean-squared error taken over all or some of the training data). To this extent, we wish to use gradient descent, with the assignment unitaries $U_1, \ldots, U_L$ as the variable, to find an optimal (or at least high-quality) assignment. In order to do so, we wish to compute the gradient of our loss function, with respect to each unitary. Similarly to the case of training classical neural networks, we compute the loss gradient for one specific training sample; for this next part, we will fix an input $|\psi\rangle$, which without loss of generality we assume is a True case.

Let us fix some $i \in \{1, \ldots, L\}$, and look at U_i. For simplicity, let's assume that U_i operates on the first d qubits, i.e., $A_i = \{1, \ldots, d\}$. Now, since we are optimizing over the manifold $U(2^d)$, some care is needed: if we were to compute the best direction to move U_i in and step in that direction, U_i would leave the manifold. Instead, we will parameterize $U(2^d)$ in the following way. From Lie theory, we know that the lie algebra of $U(2^d)$ is $u(2^d)$, the space of skew-hermitian matrices:

$$u(2^d) = \{X \in M_{2^d}(\mathbb{C}) : X = -X^*\}$$

And we know that $\exp : u(2^d) \to U(2^d)$ is a smooth, surjective mapping. Thus, given our current U_i, we can find a matrix $X_i \in u(2^d)$ such that $\exp(X_i) = U_i$; we can then optimize X_i itself instead. Since $u(2^d)$ is itself a subspace, this eliminates the above problem.

And so, we are thinking of our loss function f as a function that takes in X_i, and tells us what error it eventually causes us to have. Let $C_1 := \widehat{U_1} \cdots \widehat{U_{i-1}}$ and $C_2 := \widehat{U_{i+1}} \cdots \widehat{U_L}$ represent the parts of the circuit before and after U_i, respectively. Then, we have

$$\begin{aligned} f(X_i) &= |\langle \varphi_F | \widehat{U_1} \cdots \widehat{U_n} | \psi \rangle|^2 \\ &= |\langle \varphi_F | C_1 (U_i \otimes I) C_2 | \psi \rangle|^2 \\ &= |\langle \varphi_F | C_1 (\exp(X_i) \otimes I) C_2 | \psi \rangle|^2 \end{aligned}$$

And so we need to compute $f'(X_i)$. It will be useful to split f into 3 parts $f_1 \circ f_2 \circ f_3$, where $f_3 = \exp$, $f_1 = |\cdot|^2$, and

$$\begin{aligned} f_2 : U(2^d) &\to \mathbb{C} \\ U &\mapsto \langle \varphi_F | C_1 (U \otimes I) C_2 | \psi \rangle \end{aligned}$$

Where I here is the 2^{n-d}-row identity matrix, acting on the qubits unused by U_i. Fortunately, f_2 is quite easy to differentiate, as it's a linear function. After some wrestling with notation, its derivative (at every point X in its domain) is:

$$f_2'(X) = \mathrm{Tr}_Y \left[C_2^* | \varphi_F \rangle \langle \psi | C_1^* \right]$$

Where $\mathrm{Tr}_Y = I \otimes \mathrm{Tr}$ is the partial trace operator, where I has 2^d rows. We note 3 computationally important things:

1. $\langle \psi | C_1^*$ is the conjugate transpose of the result of running the circuit on $|\psi\rangle$ until U_i.
2. $C_2^* | \varphi_F \rangle$ is the result of running the circuit in reverse on input φ_F, until U_i.
3. The partial trace of this matrix can be computed efficiently when d is small; in particular, $C_2^* | \varphi_F \rangle \langle \psi | C_1^*$ is a very large matrix, but we do not need to actually compute it.

Putting it all together via chain rule, our loss gradient is:

$$\begin{aligned} \frac{\partial f}{\partial X_i} &= f_3'(f_2(U_i)) \circ f_2'(U_i) \circ \exp'(X_i) \\ &= f_3'(f_2(U_i)) \circ \mathrm{Tr}_Y \left[C_2^* | \varphi_F \rangle \langle \psi | C_1^* \right] \circ \exp'(X_i) \end{aligned}$$

We can average this gradient over several training samples, and then move X_i in that direction, obtaining a new matrix X_i' and a new corresponding quantum gate U_i. We can do this for all gates in our assignment, and iterate the process in the usual way for gradient descent.

Main TODOs are to compute the above components of the gradient cleanly. In particular:

- Efficiently compute the partial trace of a matrix of the form uv^* for long vectors u, v.
- Compute the derivative of exp; however, is there a way to avoid this? All we eventually need is $\exp'(X_i)$ composed with a low-rank linear map, so in principle we only need one dimension of the range of $\exp'(X_i)$, and may not need to compute the entire superoperator.
- How will the derivative of $|\cdot|^2$ work anyway? It's weird because we're representing the vectors as complex vectors, but we're kinda actually working over $\mathbb{R}$ when taking the derivatives. . .