Ryan Lin

A Comparison of Classical and Quantum Machine Learning Models: Quantum Embedding

Image Credit: Created with NightCafe AI Art Generator https://creator.nightcafe.studio/

Introduction

In recent years, we have seen the field of artificial intelligence (AI) grow at an astonishing rate. It has revolutionized all types of industries, whether it be healthcare or finance. Indeed, developing machine learning models can help automate tasks, improve decision-making, or improve the efficiency of various processes. Because of its potential, businesses across the globe are investing heavily in AI techniques to gain competitive advantages in their fields.

As the field of AI advances, we are beginning to see traditional computing techniques hit their limits in terms of processing power and speed. The computational power required to train and execute models at a large scale takes on the order of years. As a result, researchers have begun incorporating quantum techniques into our models in hopes that the properties of quantum computing can allow us to break through the barriers that classical machine learning models face today. The goal of quantum AI is to create models that can either process datasets at a faster rate or make more accurate predictions.

In this paper, I will first outline a popular classical machine learning model, the Convolutional Neural Network (CNN), and describe its architecture. I will then highlight why the structure of a quantum circuit can be seamlessly integrated into a classification model like a CNN. With the help of the Python libraries TensorFlow and Pennylane, I will also share a practical implementation of a hybrid quantum/classical model that I developed to make a binary classification of the Iris dataset. Finally, I will give a brief overview of the performance of this model and what this entails for the growing field of AI, as well as its potential intersection with quantum techniques.

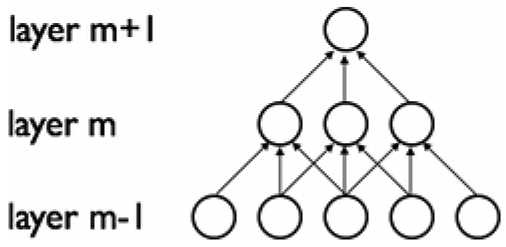
Classical Machine Learning

Machine learning today is typically aimed at solving two types of problems: classification and prediction. This section will describe the architecture of one of the most commonly used classification/detection models, the Convolutional Neural Network (CNN).

Convolutional Neural Networks (CNNs)

The CNN was developed based on the animal visual cortex. Every animal with a visual cortex takes in a “grid” of visual data and makes high-complexity classifications based on the input. The goal of the CNN is to understand the “metric” by which an animal brain can distinguish between two similar objects, like a bear or a dog.

As one would expect, the CNN shares a structure analogous to that of an animal’s neural pathways. It consists of a tree/graph with **neurons**, which “add” inputs and apply a particular activation function, and **synapses**, which apply “weights” (trained through techniques like gradient descent) to the provided inputs. The CNN represents spatially-local correlation by having connectivity between the adjacent layers of neurons.

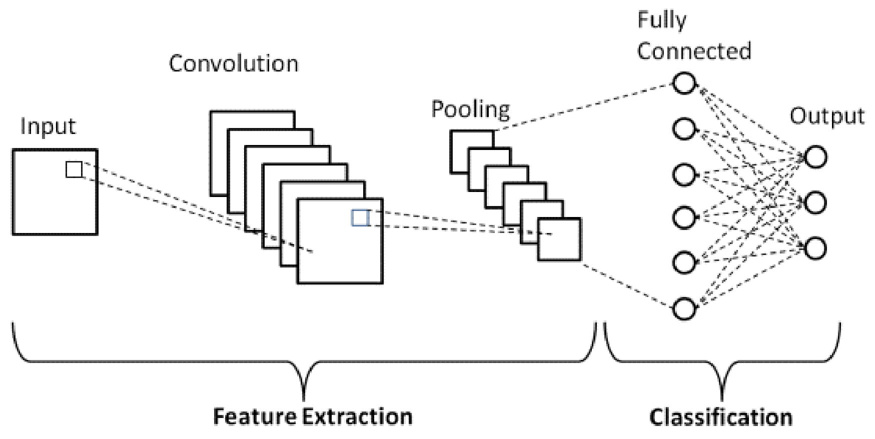


Consider this simple three-layer CNN. Let us take the *m-1* layer as the retina - it receives a large input and ultimately feeds it to the end layer, which is typically a simple classifier (ex. linear or logarithmic classifier). The neurons in layer *m* have receptive fields of width three with respect to the retina, meaning three neurons from the *m-1* layer feed into one neuron in the *m* layer. Similarly, the receptive field of layer *m+1* with respect to the input is five. Note that each of these “branches” only responds to variations within their receptive fields. However, by having numerous layers, we can make the filters produced by the various layers become global. What this means is that one unit in *m+1* can encode a non-linear feature described by the five units in layer *m-1*.

CNNs also utilize shared weights across the entire visual field. In other words, the weights of each edge are shared along the entire visual field. This allows features to be detected even if they are not located in a specific space in the field. The following hypothetical model will illustrate the need for this functionality:

1. Suppose there exists a model that hopes to identify the existence of a red spot on an otherwise fully white background.
2. This model is trained solely on a dataset that has red dots in the center of the image.
3. When encountering test data that has a red dot located at the corner of the image, a model that does not have shared weights would incorrectly determine that there is no red dot in the image.
4. A CNN that utilizes shared weights would be able to detect this red dot regardless of its location

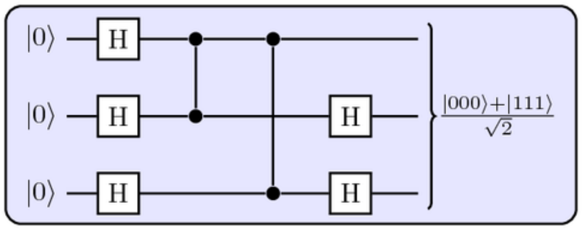
Consider the following visualization of a basic CNN architecture:



The CNN applies a series of filters to the input data, applying the trained weights to a local region of the input and passing this output through an activation function to eventually produce a single output value. By applying many of these filters, a CNN can learn complex, high-level features. The entire set of these features is referred to as a feature map or an embedding. It maps our data into a higher or lower dimensional space and eventually arrives at some form of classifier.

Quantum Embedding

Consider the following example quantum circuit:



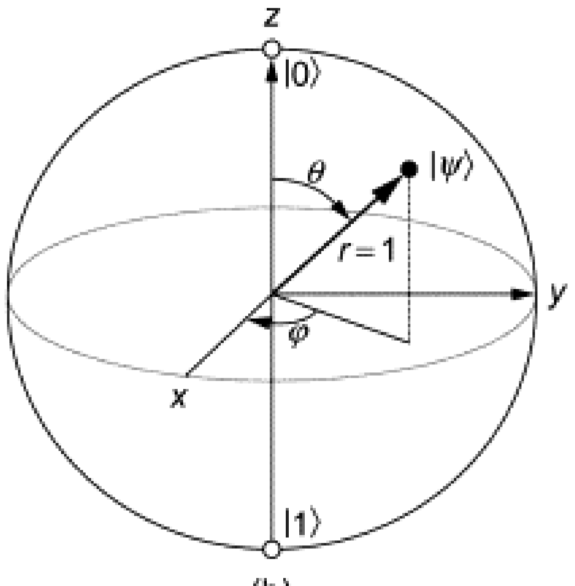
Indeed, we can use a quantum circuit to create a structure similar to the “embedding” in a classical machine learning model. Consider a quantum circuit that features gates with trainable weights. The input would pass through several gates and eventually be measured at the end. Note that one drawback of such an implementation is that back-propagation, a concept that classical models use to “check” their work, is impossible due to the No-Cloning Theorem.

Data Representation

Classical methods embed data points as vectors in a Hilbert space. It then performs numerous linear algebra operations on these vectors to produce an accurate representation of the unknown "metric" that we use to make classifications (ex. what is the difference between a bear and a dog). Quantum computers can model these vectors even more efficiently, as the states of quantum systems are already vectors in a high-dimensional Hilbert space. Instead of preparing the inputs as standard vectors, we can initialize them as state vectors like so:

$$x = 0101 \rightarrow |\psi\rangle = |0101\rangle$$

Because of this, we can perform linear algebraic operations on the space in poly-logarithmic time, which is an exponential speedup when compared to classical operations.



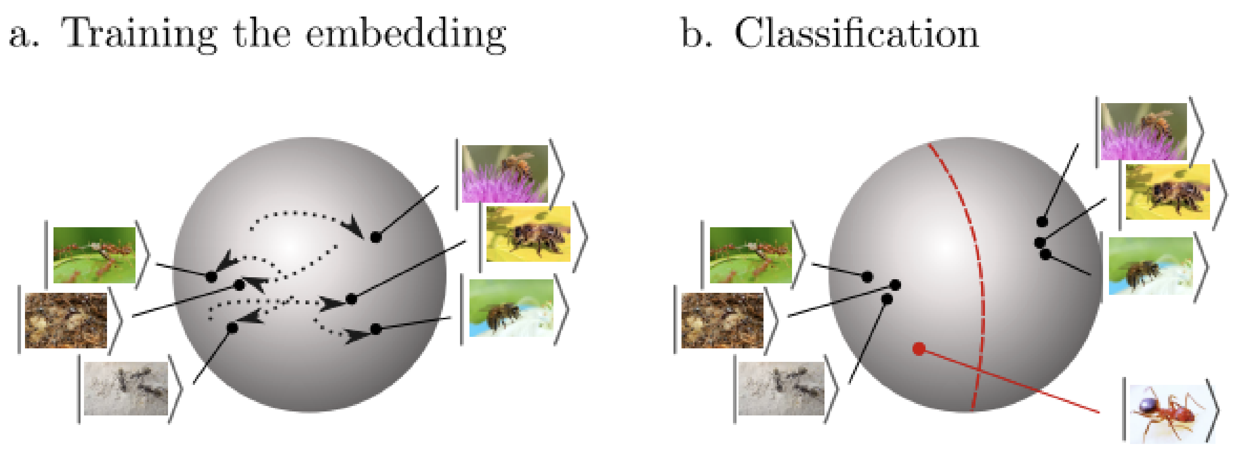
(b)

Parameters and Training

Recall that the proposed quantum feature map is a quantum circuit with trainable gates. The physical parameters associated with these gates can be trained and adapted via classical optimization methods like gradient descent. An example of such a physical parameter in an optical quantum computer is the angles of beam splitters.

Classification

In the proposed quantum feature map, a fixed quantum circuit with trainable gates is trained to maximize the distance between “clusters” of data points. Each data point is represented as a ket vector at some angle in the Hilbert space. As in the CNN, a simple decision boundary in this Hilbert space can correspond to an extremely complex decision in the original data space.



The final classification is made by applying a threshold to the results of the classifier. This is done by making a measurement:

$$f(x) = \langle x | \mathcal{M} | x \rangle$$

In a binary classification model (i.e. data labeled either -1 or 1) we can set this threshold to be 0.

Evaluating Model Quality

As in a classical machine learning model, we can measure the quality of a classification model through a cost function, which is typically the sum of a loss function and a regularizer

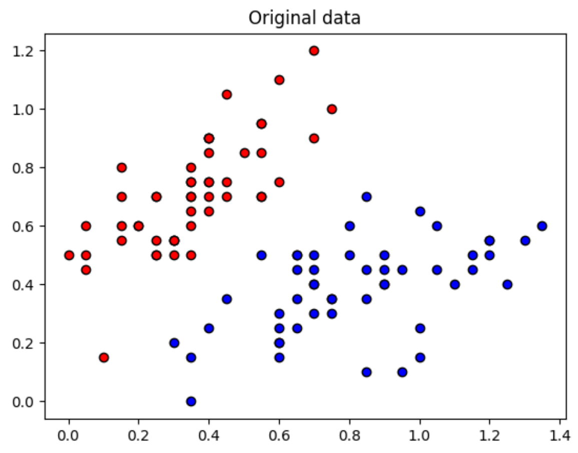
$$\frac{1}{M} \sum_m L(f(x^m), y^m)$$

Practical Implementation

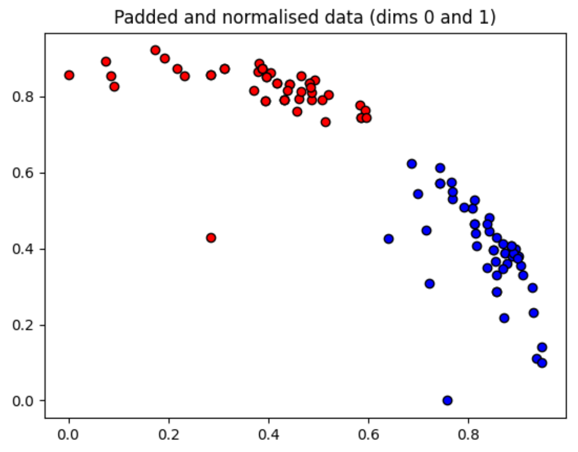
Using Pennylane and TensorFlow, I created a model with both classical and quantum layers. This model was intended to be a proof-of-concept that made a trivial classification on a simple dataset.

Data Preparation

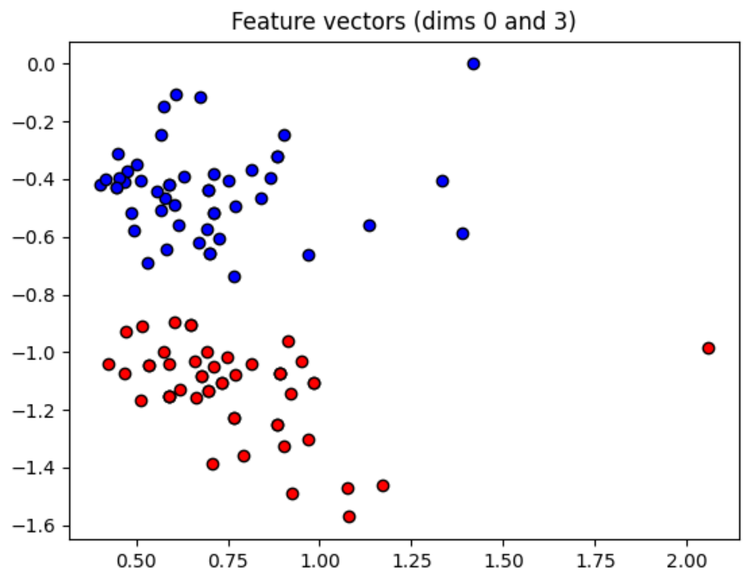
To train and test this model, I used a trimmed-down version of the famous Iris dataset, which is a dataset of iris plants. The original dataset contained three classes of these types of plants, but the trimmed dataset only contains two (to make the classification simpler for the model).



Of course, this data was first scaled and normalized before being fed into several helper functions to prepare it for the quantum layer.



The first of these functions is for translating every input *x* into a set of angles. The second of these functions feeds these angles into a simple circuit for state preparation. After going through these functions, the input data is ready for the quantum layer. The following graph depicts the data after preparation.



Quantum Layer

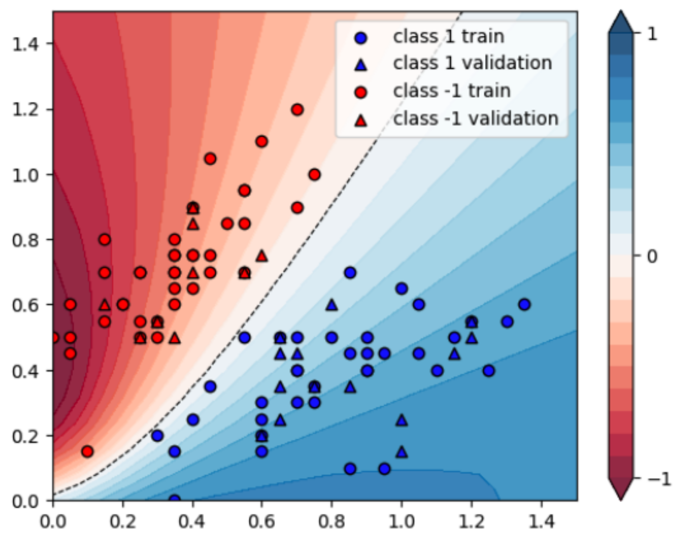
The quantum layer defined for this model was a simple 2-qubit circuit that rotates every qubit and applies a CNOT to entangle the qubit with its neighbor. This is analogous to the connections between layers in the CNN synapse architecture.

Cost and Activation Functions

The cost function used for this model was the squared loss, which is a commonly-used cost function in many classical models. We then used the standard NesterovMomentumOptimizer provided by TensorFlow to train the parameters used in state preparation. Recall that these state preparation parameters are analogous to the weights of a classical machine learning model.

Results

After 60 iterations of training (≈30 minutes of training), the model was able to classify both the training and testing data with 100% accuracy.



Training Accuracy: 1.0000000   Testing Accuracy: 1.0000000

Concluding Remarks

The model described above was used to make a fairly trivial binary classification. A classical model could likely have trained itself for this task in a couple of minutes and achieved fairly high accuracy. Our hybrid model was only able to achieve such an accuracy after 30 minutes of training. The length of this training process implies that such a model is not scalable for an everyday developer. Increasingly complex tasks (tasks with more features or more types of classes) would likely be extremely computationally intensive to train for and difficult to implement with Pennylane alone.

This lengthy training process is likely due to the fact that we cannot utilize back-propagation (due to the No-Cloning Theorem) and the limitations of Pennylane. The library only simulates quantum circuits with classical techniques, so one can imagine that a true quantum circuit would speed up the training process significantly.

Overall, quantum machine learning poses itself as a potentially promising area for further research and a possible future method for creating high-performance models. Despite our hardware limitations, even the proof-of-concept hybrid model was able to achieve high accuracy in a reasonable number of iterations. Perhaps as we develop larger quantum processors, we may see quantum embedding become a commonplace practice in the near future.

Bibliography

"Convolutional Neural Networks (Lenet)." Convolutional Neural Networks (LeNet) - DeepLearning 0.1 Documentation, https://web.archive.org/web/20171228091645/http://deeplearning.net/tutorial/lenet.html.

Lloyd, Seth, et al. "Quantum Embeddings for Machine Learning." ArXiv.org, 10 Feb. 2020, https://arxiv.org/abs/2001.03622.

"Multilayer Perceptron." Multilayer Perceptron - DeepLearning 0.1 Documentation, https://web.archive.org/web/20171228115046/http://www.deeplearning.net/tutorial/mlp.html.

"Pennylane." PennyLane, https://pennylane.ai/.

Skalski, Piotr. "Gentle Dive into Math behind Convolutional Neural Networks." Medium, Towards Data Science, 14 Apr. 2019, https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.