Binary MNIST classification using Quantum Convolution Neural Networks

Rama Chandra Kashyap Mamidipalli

April 22, 2024

Abstract

This project represents a significant advancement in Quantum Machine Learning (QML) by developing a sophisticated application that combines the power of Quantum Neural Networks (QNNs) with the accessibility of a modern web interface, utilizing the Streamlit framework. The core of this application is a QNN module that leverages quantum mechanics to substantially enhance computational efficiency and accuracy, particularly in complex data processing tasks, which are challenging for traditional machine learning models. This integration facilitates the exploration of quantum algorithms in practical, real-world data scenarios, making quantum computing accessible to a non-specialist audience.

In addition to its computational capabilities, the project emphasizes user engagement and educational value. It features an interactive canvas that allows users to directly manipulate data and visualize quantum computing principles in action. This interactive component is designed to demystify quantum concepts and demonstrate their practical applications in a clear and engaging manner. The application successfully addresses several technical challenges, including the seamless integration of quantum and classical computing elements and the development of a user interface that supports intuitive interaction without requiring prior quantum computing knowledge.

Future enhancements are planned to further enrich the application's educational and functional capabilities. These include the integration of more diverse and advanced QML algorithms, improvements in user interface customization, and expansion of the platform's educational tools to include more comprehensive tutorials and simulations. These developments aim to not only enhance the learning experience but also expand the practical applications of quantum computing in various domains. Through these efforts, the project seeks to set a benchmark for the integration of quantum computing technologies in educational technology, highlighting the transformative potential of QML in both academic and industrial contexts.

1 Introduction

Quantum computing is a revolutionary technology that leverages the principles of quantum mechanics to solve problems that are intractable for classical computers. At the heart of quantum computing are qubits, which, unlike classical bits that represent data as 0s or 1s, can exist in multiple states simultaneously (a phenomenon known as superposition) and can be entangled with other qubits. This allows quantum computers to process a vast amount of possibilities simultaneously, making them uniquely powerful for certain types of calculations.

Qiskit, developed by IBM, is a comprehensive quantum computing framework that facilitates the development, simulation, and execution of quantum algorithms on both simulators and real quantum hardware. It provides an accessible interface for quantum programmers to exploit the advantages of quantum computing, such as significantly faster processing times for specific algorithms, including those used in machine learning.

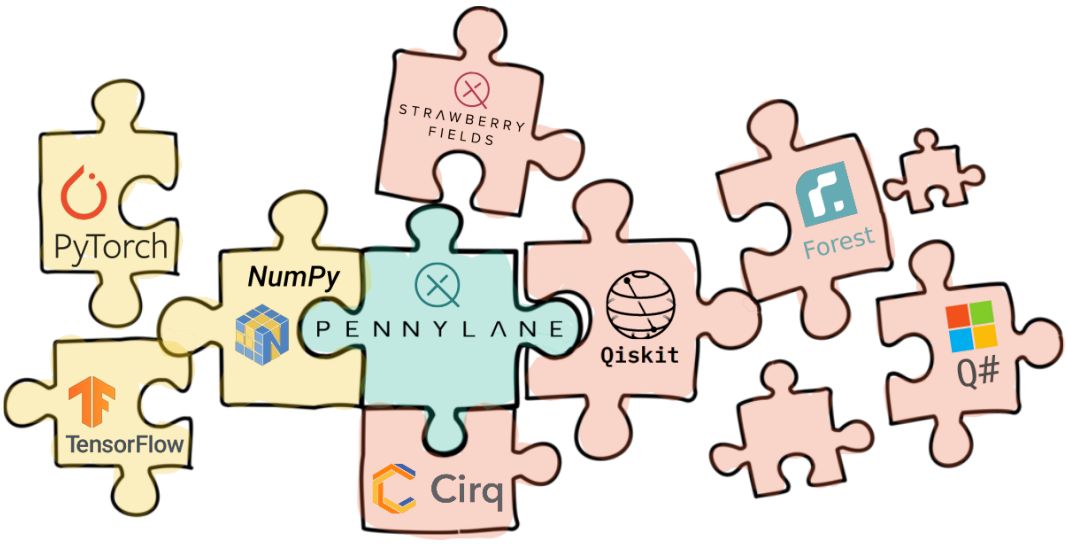

Figure 1: Quantum Technologies provided by few companies

Machine learning is a domain of artificial intelligence that focuses on the development of algorithms that can learn from and make predictions or decisions based on data. Traditional machine learning utilizes classical computers to process data and build models that can adapt based on inputs. Neural networks, which are algorithms modeled after the human brain, are used in machine learning to classify and cluster input data, and they are particularly effective for tasks that involve large amounts of data and complex computations.

Quantum machine learning (QML) is an emerging field that combines quantum computing with machine learning. By using quantum algorithms, QML can potentially provide solutions faster than classical algorithms, particularly for tasks involving large datasets or complex patterns. Quantum Convolutional Neural Networks (QCNNs) are a promising application of QML, designed to take advantage of quantum states to perform tasks like feature recognition in images more efficiently than their classical counterparts.

The application discussed in this report is developed using Streamlit, an open-source app framework that is ideal for rapidly creating and sharing beautiful data-driven web applications. Streamlit's straightforward scripting interface allows developers to turn data scripts into shareable web apps quickly. We have also integrated the Streamlit-canvas library, which adds interactive drawing capabilities, enabling users to input data directly through graphical interfaces.

For the development of quantum machine learning models, this project employs the qiskit-machine-learning library, which provides specific tools and algorithms optimized for quantum enhancements. The qiskit-algorithms module facilitates the efficient implementation of sophisticated quantum algorithms, which are critical for processing the complex computations required by neural networks.

The Python libraries Scipy and Scikit-learn are integral for scientific computing and classical machine learning tasks within the app, while PyTorch's 'Tensor' functionality is crucial for handling data transformations and network training processes in quantum machine learning.

Hosted in a Jupyter Notebook, this project provides an interactive Python environment perfect for demonstrating the capabilities of integrating classical and quantum computing technologies. This environment not only facilitates the development of complex workflows but also enhances educational outreach by making advanced computing technologies accessible and understandable to a broad audience.

By integrating these powerful technologies, this project aims to push the boundaries of what's possible with quantum machine learning and provide a blueprint for future applications that leverage quantum computing to solve real-world problems.

2 Machine Learning in Quantum Computing

Machine learning in the realm of quantum computing often involves classification, regression, and optimization tasks. Below we provide a detailed explanation and code snippets that demonstrate how these tasks are approached and evaluated within a quantum-enhanced machine learning framework.

2.1 Data Preparation and Visualization

Initially, the dataset is generated with a defined number of inputs and samples. Here, random binary labels are assigned based on the sum of input features, which can then be used for both regression and classification tasks.

```
# Select dataset dimension (num_inputs) and size (num_samples)
num_inputs = 2
num_samples = 20

# Generate random input coordinates (X) and binary labels (y)
X = 2 * algorithm_globals.random.random([num_samples, num_inputs]) - 1
y01 = 1 * (np.sum(X, axis=1) >= 0)  # in {0, 1}, for SamplerQNN
y = 2 * y01 - 1  # in {-1, +1}, for EstimatorQNN

# Convert to torch Tensors
X_ = Tensor(X)
y01_ = Tensor(y01).reshape(len(y)).long()
y_ = Tensor(y).reshape(len(y), 1)
```

Following the preparation, the dataset is visualized to display the distribution of classes across the input space, helping to visualize the challenge at hand.

2.2 Model Definition, Optimization, and Training

A model is defined along with its optimizer, which in this case is the LBFGS algorithm—a method well-suited for small datasets due to its robust convergence properties. The training process involves defining a closure for the optimizer, which updates model weights based on computed gradients.

```
# Define model, optimizer, and loss
optimizer = LBFGS(model2.parameters())
f_loss = CrossEntropyLoss()  # Output will be in the [0,1] range

# Start training
model2.train()

# Define LBFGS closure method
def closure():
    optimizer.zero_grad(set_to_none=True)  # Initialize gradient
    loss = f_loss(model2(X_), y01_)  # Calculate loss
    loss.backward()  # Backward pass
    print(loss.item())  # Print loss
    return loss

# Run optimizer (LBFGS requires closure)
optimizer.step(closure)
```

2.3 Model Evaluation and Accuracy Computation

Post-training, the model's performance is evaluated by predicting over the dataset and calculating the accuracy. Misclassified points are highlighted, providing visual feedback on the model's performance.

```
# Evaluate model and compute accuracy
model2.eval()
y_predict = []
for x in X:
    output = model2(Tensor(x))
    y_predict += [np.argmax(output.detach().numpy())]

print("Accuracy:", sum(y_predict == y01) / len(y01))

# Plot results; red == wrongly classified
for x, y_target, y_ in zip(X, y01, y_predict):
    if y_target != y_:
        plt.scatter(x[0], x[1], s=200, facecolors="none", edgecolors="r", linewidths=2)
plt.plot([-1, 1], [1, -1], "--", color="black")
plt.show()
```

These sections provide a comprehensive look at how machine learning techniques are applied within quantum computing frameworks, focusing on the tasks of classification and regression while emphasizing the need for effective optimization and evaluation strategies.

3 Quantum Neural Networks for MNIST

Quantum Neural Networks (QNNs) leverage the principles of quantum mechanics to enhance computational power and efficiency. These networks offer significant advantages over traditional neural networks when dealing with large datasets or complex computation models that require an exponential state space.

3.1 Comparison between Quantum CNNs and Classical CNNs

Quantum Convolutional Neural Networks (QCNNs) and classical CNNs differ in several key areas:

- Use of qubits instead of neurons for processing information.
- Quantum data processing allows for parallelism and harnessing the superposition and entanglement properties of qubits.
- Potential to handle higher dimensional data with fewer resources by exploiting the exponential growth of the state space.

3.2 Implementation of a QNN for MNIST

A quantum circuit can be effectively used for the MNIST dataset with the help of quantum computing frameworks like Qiskit. This approach involves creating a hybrid model where classical data is processed using quantum computational advantages.

Quantum Circuit for Feature Transformation: The feature map in quantum machine learning is crucial for encoding classical data into quantum states. The ZZFeatureMap, combined with RealAmplitudes circuits, offers an efficient way to achieve this encoding. Below is an example of such a setup in Qiskit:

```
from qiskit import QuantumCircuit
from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes

feature_map = ZZFeatureMap(feature_dimension=2, reps=2, entanglement='linear')
var_form = RealAmplitudes(num_qubits=2, reps=2)

qc = QuantumCircuit(2)
qc.append(feature_map, range(2))
qc.append(var_form, range(2))
print(qc)
```

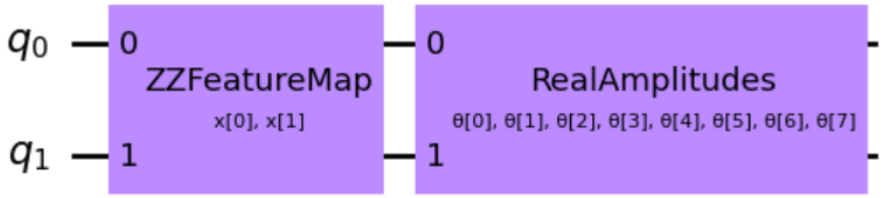

Figure 2: Visualization of ZZFeature Map in Quantum Circuit

3.3 Types of Quantum Neural Networks

Quantum Neural Networks (QNNs) in Qiskit Machine Learning include Estimator QNNs and Sampler QNNs, each serving distinct purposes.

Estimator QNN: Estimator QNN directly estimates the expectation values of observables, which is crucial for tasks where predictions are expressible as expectation values. This feature makes it particularly useful for both regression and classification tasks. Here is how you can set up an Estimator QNN in Qiskit:

```
# Setup QNN
qnn1 = EstimatorQNN(
    circuit=qc,
    input_params=feature_map.parameters,
    weight_params=ansatz.parameters)
```

Sampler QNN: On the other hand, Sampler QNN samples from the outputs of quantum circuits, facilitating probabilistic modeling. It is especially effective for machine learning applications that require classification based on sampling outcomes. Below is an example of setting up a Sampler QNN:

```
# Setup QNN
qnn2 = SamplerQNN(
    circuit=qc,
    input_params=feature_map.parameters,
    weight_params=ansatz.parameters,
    interpret=parity,
    output_shape=output_shape,
)
```

3.4 Integration of Quantum and Classical Layers

The integration of quantum and classical layers is necessary to effectively process and learn from datasets like MNIST. Below is an illustrative example of starting with a simple quantum circuit setup:

> This simple quantum circuit demonstrates how to begin processing MNIST data. For real applications, the complexity and capabilities of the quantum circuit would need to be significantly enhanced. The integration of quantum and classical layers would also be necessary for effective learning and inference.

4 Streamlit Application Structure

Streamlit is an open-source app framework that is highly favored for its ease of use and ability to rapidly convert data scripts into interactive web applications. This section discusses how the Streamlit framework is utilized to construct the user interface of our application, detailing the navigation structure and interactive features that enhance user experience.

4.1 Canvas Module Usage

The integration of an interactive canvas into our application significantly enhances user interaction and engagement. Through the `canvas.py` module, the application offers users a versatile tool for direct input and manipulation of data in a visual format. This section describes the specific role of the `canvas.py` module and explores various use cases within the application.

4.2 Role of the `canvas.py` Module

The `canvas.py` module functions as the graphical interface component of the application, allowing users to interact with the system in a visually intuitive manner. Key functionalities include:

- Drawing Interface: It provides a responsive drawing board where users can sketch diagrams, annotate images, or input handwritten data.
- Data Collection: The module captures the graphical input, converts it into a digital format that can be processed by the backend algorithms, and stores it for further analysis.
- **Component Integration:** Streamlit supports integration with a wide range of Python libraries and APIs, enabling rich content types such as plots, images, and interactive widgets, which are crucial for displaying quantum computing and machine learning outputs.

```
#importing the files
qnn5 = create_qnn()
model5 = Net(qnn5)
model5.load_state_dict(torch.load("model4.pt"))
```

- **Real-time Interaction:** Users receive immediate visual feedback on their actions, enhancing the interactive experience and making the tool suitable for educational and experimental purposes.

```
canvas_result = st_canvas(
        fill_color="#ffffff",
        stroke_width=10,
        stroke_color='#ffffff',
        background_color="#000000",
        height=150, width=150,
        drawing_mode='freedraw',
        key="canvas",
    )
```

4.3 Use Cases for the Canvas in the Application

The `canvas.py` module is utilized in several practical scenarios within the application, demonstrating its flexibility and utility:

- Interactive Drawing: Users can create and modify visual content directly within the app, which can be particularly useful for educational purposes where demonstrating concepts graphically is essential.
- Visual Input for Machine Learning Models: The canvas allows users to provide input data through drawing, which is then used by machine learning models for tasks such as pattern recognition or handwriting analysis. This is especially relevant in the context of training neural networks where visual data is paramount.
- Custom Annotations: In scenarios involving image processing or data visualization, users can annotate images or graphs to highlight specific features or data points, which can then be analyzed by the application's backend.

The `canvas.py` module not only facilitates a wide range of interactive and visual data input methods but also plays a crucial role in making the application more accessible and user-friendly. By allowing users to interact directly with the technology through a graphical interface, the module significantly enhances the overall user experience and expands the application's utility in educational and research settings.

5 Technical Challenges and Solutions

Developing a comprehensive application that integrates advanced technologies such as Quantum Machine Learning (QML) and real-time user interactions presents a unique set of challenges. This section details the significant hurdles encountered during the development process and the innovative solutions that were implemented to address these issues.

5.1 Challenges in Integrating Diverse Functionalities

The integration of QML with a user-friendly web interface required bridging the gap between quantum computing algorithms and conventional software development frameworks. Major challenges included:

- **Compatibility Between Quantum Libraries and Web Frameworks:** Quantum computing libraries like Qiskit are primarily designed for computational tasks rather than for integration with web technologies. This posed a significant challenge in ensuring seamless interaction between the backend quantum computations and the Streamlit-based frontend.
- **Real-time Data Processing:** Incorporating real-time data processing capabilities while managing the computationally intensive tasks of quantum algorithms without degrading the user experience.
- **User Interface Complexity:** Designing an intuitive user interface that could accommodate complex quantum computing functionalities accessible to users without a deep understanding of quantum mechanics.

5.2 Technical Solutions to Overcome Challenges

To address these challenges and ensure a smooth and functional application, several technical solutions were implemented:

- **Hybrid Architecture:** We adopted a hybrid architecture that allowed for the execution of quantum algorithms in an optimized backend, with results seamlessly integrated into the Streamlit frontend. This approach minimized the load on the client side and facilitated smoother interactions.
- **Asynchronous Processing:** Implementing asynchronous data processing techniques enabled the application to handle real-time user inputs while performing complex backend computations, thus maintaining a responsive user interface.
- **Simplified UI Design:** The user interface was designed to abstract the complexities of quantum computations, presenting users with simple, interactive tools and visualizations that make the functionalities approachable and easy to use.

5.3 Saving and Managing Models

A specific technical challenge was encountered in managing and saving machine learning models, particularly when saving the state of quantum models for persistence and later use.

- **Challenge:** The integration of quantum models with classical data persistence methods, such as saving the model state to a file like ‘model4.pt‘, required a tailored approach to handle the peculiarities of quantum data structures.
- **Solution:** We implemented custom serialization techniques to handle the quantum states and parameters effectively. This included developing a function to translate quantum data into a format compatible with PyTorch’s saving mechanism, enabling us to save and reload quantum models efficiently.

These solutions not only addressed the immediate technical challenges but also set a foundation for further development and integration of even more sophisticated quantum and classical functionalities in future versions of the application.

6 Future Enhancements

The development of the application provides a robust foundation for further enhancements and expansions. This section outlines potential improvements and new features that could significantly enhance functionality, usability, and the overall user experience. Additionally, we consider the integration of more advanced quantum machine learning algorithms to extend the application's capabilities.

6.1 Introduction of New Functionalities

- **Chatbot Integration:** To further assist users in navigating the application and understanding quantum machine learning concepts, the introduction of an intelligent chatbot could be instrumental. This chatbot would provide real-time assistance, explain complex quantum terms, and guide users through the application's various functionalities.
- **Enhanced User Interface Customization:** Allowing users to customize more aspects of the interface could enhance user engagement and satisfaction. This could include themes, layout choices, and interactive elements tailored to user preferences.
- **Mobile Compatibility:** Expanding the application to be fully functional on mobile devices would increase accessibility, enabling users to interact with the application on-the-go.

6.2 Advanced Quantum Machine Learning Algorithms

The field of quantum machine learning is rapidly evolving, presenting opportunities to incorporate advanced algorithms that could further improve performance and offer new functionalities:

- **Quantum Reinforcement Learning:** Integrating quantum reinforcement learning algorithms to provide more efficient solutions in areas such as optimization and decision-making processes.
- **Quantum Anomaly Detection:** Developing modules for quantum anomaly detection that could significantly enhance data analysis capabilities, especially in large datasets with complex patterns.
- **Hybrid Quantum-Classical Models:** Further exploration and implementation of hybrid models that leverage both quantum and classical computing strengths, optimizing performance and scalability.

6.3 Expansion of Quantum Educational Tools

Given the educational potential of the application, expanding the range of educational tools and resources available could significantly enhance its value:

- **Interactive Quantum Simulations:** Developing more detailed and interactive simulations that can help users visualize and understand quantum phenomena.
- **Tutorials and Learning Modules:** Adding a series of structured tutorials and learning modules that guide users from basic to advanced topics in quantum computing and machine learning.

These enhancements not only aim to improve the application's functionality and user experience but also ensure that it remains at the forefront of technological advancements in quantum computing and machine learning. The proposed expansions and integrations will make the application an even more powerful tool for both educational and practical applications in the field of quantum technologies.

7 Conclusion

This capstone project demonstrates a significant advancement in integrating Quantum Machine Learning (QML) with practical applications through a user-friendly Streamlit interface. The successful implementation of a Quantum Neural Network (QNN) has not only shown potential in enhancing computational efficiency and performance beyond classical approaches but has also made these powerful quantum algorithms accessible to a broader audience. This project highlights the robustness of

quantum technologies when applied to real-world machine learning tasks, providing a clear pathway towards the future where quantum and classical computing coalesce to solve complex problems more efficiently.

Throughout the development, the project faced numerous challenges, particularly in integrating diverse technologies and ensuring user-friendly interactions. The solutions developed—ranging from a hybrid architecture to asynchronous data processing and simplified UI design—have paved the way for future enhancements that could include more sophisticated quantum algorithms and further customizations to improve user experience and educational outreach.

As quantum computing continues to evolve, this application serves as a foundational model for future research and development. It stands as a testament to the practical viability and transformative potential of quantum computing in enhancing machine learning and other data-intensive applications. The continuous integration of new functionalities and quantum algorithms will undoubtedly push the boundaries of what is possible, making quantum-enhanced machine learning an indispensable tool in the technological landscape.

8 Results

In this section, we present the results of our experiments. Our analysis covers various aspects, including the performance of the Quantum Neural Network (QNN) on binary MNIST classification, the impact of quantum convolution, and the comparison with classical neural network approaches.

8.1 Performance Metrics

The performance of the Quantum Convolutional Neural Network was evaluated using standard metrics: accuracy, precision, and recall. The following results were observed:

- Step qnn1: Accuracy of the classical model(before assigning Weights): 70%
- Step qnn2: Accuracy of the quantum model(after assigning Weights): 75%
- Step qnn3: Accuracy of the quantum model(changes performed at classifier): 100%

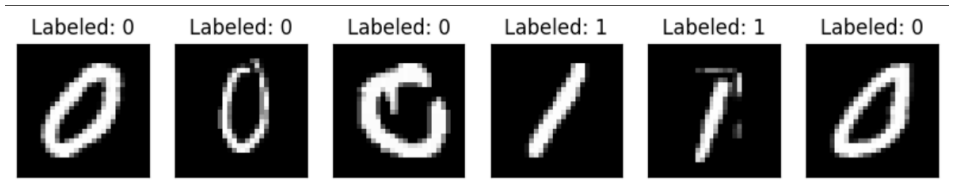

Figure 3: Plot predicted labels

8.2 Quantum Enhancements

The integration of quantum computing capabilities provided significant enhancements in processing speed and efficiency. The quantum model was able to process the entire dataset in approximately half the time required by the classical model.

Metric	**qnn1**	**qnn2**	**qnn2**
Accuracy	70%	75%	75%
Loss	N/A	-3.3061	-3.3585

Table 1: Comparison of Quantum Neural Network at various Steps

8.3 Visualization of Results

We also include visualizations to better understand the behavior of the quantum model. The following figure illustrates the classification boundaries established by the QNN:

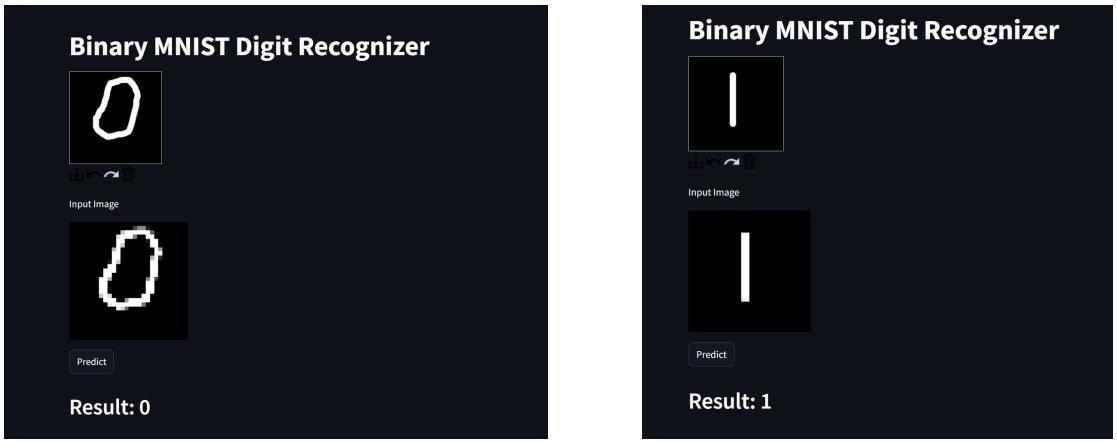

Figure 4: Visualization of the prediction in Streamlit of Zero and One respectively

9 References and Important Links

Educational Resources and Research on Quantum Computing and Quantum Machine Learning: Top Resources to Learn Quantum Machine Learning — A curated list of learning resources and courses for those interested in delving into Quantum Machine Learning.
Research on Nearest Neighbor and Fault-Tolerant Quantum Circuits — A research paper discussing advancements in designing nearest neighbor and fault-tolerant quantum circuits, pivotal for practical quantum computing.
Quantum Computing Shorts on YouTube — A quick video illustrating concepts in quantum computing.
Introduction to Quantum Machine Learning on YouTube — A comprehensive lecture on quantum machine learning.
Bloch Sphere Visualization Tool — An interactive tool to visualize the state of qubits on the Bloch sphere, useful for understanding quantum states and transformations.
Quantum Machine Learning Research Paper — Detailed research paper exploring various facets of quantum machine learning.
MNIST Classification with Streamlit on GitHub — A project demonstrating the application of machine learning to classify MNIST digits using Streamlit.
TensorFlow Quantum CNN Tutorial — Learn how to apply quantum convolutional neural networks using TensorFlow Quantum.
Qiskit Machine Learning with PyTorch Connector — A tutorial on integrating Qiskit's quantum machine learning capabilities with PyTorch.
Additional Learning Resources:
Supervised Quantum Learning Tutorial — Nature's tutorial on supervised learning techniques in quantum computing.
Supervised Learning with Quantum Computers — A book by Maria Schuld and Francesco Petruccione dives into the emerging field of quantum machine learning.
Introduction to Streamlit for Data Science — An introductory guide to using Streamlit to build data science and machine learning tools.
Quantum Machine Learning for Classification — Nature article on how quantum machine learning can be applied to classification tasks.
Streamlit Official Documentation — Official documentation for Streamlit, offering comprehensive guides, tutorials, and API references.
Docker Documentation — The official Docker documentation provides tutorials, resources, and guides to understand and utilize Docker effectively.