UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

SISTEMAS DE INFORMAÇÃO

THOMAZ BARBOSA MACIEL

Simulando perceptrons quânticos em problemas de classificação de imagens da base Quickdraw

Recife

2022

THOMAZ BARBOSA MACIEL

Simulando perceptrons quânticos em problemas de classificação de imagens da base Quickdraw

Trabalho apresentado ao Programa de Graduação em Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Fernando Maciano de Paula Neto

Recife

2022

THOMAZ BARBOSA MACIEL

Simulando perceptrons quânticos em problemas de classificação de imagens da base Quickdraw

Trabalho apresentado ao Programa de Graduação em Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Recife, 19 de Maio de 2022

BANCA EXAMINADORA

_____

Prof. Fernando Maciano de Paula Neto (Orientador)

UNIVERSIDADE FEDERAL DE PERNAMBUCO

_____

Prof. Stefan Michael Blawid (2º membro da banca)

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Agradecimentos

Meu total agradecimento a Deus, criador dos céus e da terra, cuja bondade e misericórdia permitiram que eu estivesse aqui hoje, fazendo com que este e qualquer outro trabalho feito por este discente pudesse ser realizado.

Meus sinceros agradecimentos a minha família por me incentivar e me guiar pelo caminho do estudo e do conhecimento, como também a todos aqueles que me ajudaram até este dia.

Também agradeço ao professor Fernando Maciano, orientador deste trabalho, e ao Centro de Informática, por todo o aparato técnico, científico e educacional disponibilizado.

A estes, meu muito obrigado!

"Eu, porém, cantarei a tua força; pela manhã louvarei com alegria a tua misericórdia; porquanto tu foste o meu alto refúgio, e proteção no dia da minha angústia.”

Salmos 59:16

Resumo

Modelos quânticos de aprendizagem máquina têm sido propostos para unir as vantagens da computação quântica com a adaptabilidade e generalização de comportamento dos modelos inteligentes. Uma das implementações mais simples e aplicáveis é o neurônio "Perceptron", desenvolvido por Frank Rosenblatt, cuja versão quântica foi proposta por Francesco Tacchino et. al. Há também uma extensão desse trabalho que permite construir o neurônio que suporta entradas e pesos com valores reais, proposto por Mangini et. al. Este trabalho propõe a análise da execução e treinamento da implementação de tal modelo, avaliando a precisão desse neurônio na base pública Quickdraw.

Palavras-chave: Perceptron, Quântico, Quickdraw.

ABSTRACT

Quantum machine learning models have been proposed to combine the advantages of quantum computing with the adaptability and generalization of behavior of intelligent models. One of the simplest and most applicable implementations is the “Perceptron” neuron, developed by Frank Rosenblatt, whose quantum version was proposed by Francesco Tacchino et. al. There is also an extension of this work that allows building the neuron that supports inputs and weights with real values, proposed by Mangini et. al. This work proposes the analysis of the execution and training of the implementation of such a model, evaluating the quality of this neuron in the Quickdraw public database.

Keywords: Perceptron, Quantum, Quickdraw.

Lista de Ilustrações

SUMÁRIO

1 INTRODUÇÃO

Como uma das áreas mais proeminentes no estudo da computação, a Computação Quântica tem sido um dos campos mais estudados e investidos nos últimos anos. O uso de modelos computacionais com base nas leis probabilísticas da física quântica tem trazido avanços para diversos setores como aprendizagem de máquina e inteligência artificial. Sendo desenvolvido entre 1958-1959, o *Perceptron* é um modelo desenvolvido por Frank Rosenblatt, que funciona como um "classificador linear", usado para classificar os dados de entrada fornecidos a ele e fornecer uma saída binária baseada nessa mesma classificação[1].

Em 2019, Francesco Tacchino e colaboradores introduziram o seu modelo de implementação do Perceptron de Rosenblatt. Sua implementação mostra um algoritmo que assim como sua saída, recebe valores binários como entrada, tendo resultados satisfatórios o suficiente para que seu autor considere sua implementação, em uma tradução livre, como "um primeiro passo para redes neurais quânticas práticas implementadas eficientemente em hardware de processamento quântico de curto prazo"[2].

Apesar da comprovação da sua eficiente implementação, o modelo apresentado por Tacchino et al em 2019 ainda tinha sua limitação em relação aos valores de entrada, sendo capaz apenas de aceitar valores binários para sua classificação. Com o intuito de criar uma generalização no uso do algoritmo apresentado por Tacchino et al e aumentar assim a usabilidade do modelo, Stefano Mangini et al propôs em 2020, em conjunto com o próprio Tacchino e outros, uma extensão do modelo apresentado anteriormente, onde agora, a implementação receberia um vetor de valores discretos. O vetor de entrada seria processado juntamente com um vetor de pesos, e seria classificado utilizando o mesmo número de qbits, fator cujo o autor menciona em uma tradução livre, ser crucial para a aplicação direta de um "procedimento de aprendizado automático de diferenciação"[3].

1.1 OBJETIVOS

1.1.1 Gerais

O objetivo geral deste trabalho é desenvolver uma implementação prática do modelo de neurônio quântico proposto por Stefano Mangini et al em [3], executando uma aplicação em uma base de dados real (QuickDraw) para fins de classificação de imagens, analisando assim sua precisão e acurácia.

1.1.2 Específicos

Este trabalho tem como objetivos específicos:

A. Detalhar a implementação prática do modelo proposto por Mangini utilizando a linguagem de programação Python e bibliotecas como Qiskit e NumPy;
B. Demonstrar a utilização do modelo com dados reais utilizando a base de dados pública do QuickDraw;
C. Analisar os resultados do modelo na classificação de imagens;
D. Identificar potenciais melhorias e expansões a implementação apresentada neste trabalho através dos resultados obtidos.

Estaremos abordando na seção 1.2 deste trabalho alguns conceitos básicos da "Computação Quântica" considerados necessários para a compreensão do conteúdo abordado. No capítulo 2, serão apresentados os modelos de Neurônios artificiais entregues por Tacchino e Mangini. No capítulo 3, abordaremos os experimentos realizados sobre a base de dados do QuickDraw, utilizando o modelo apresentado por Mangini. No capítulo 4, estaremos expondo as conclusões obtidas com este trabalho, como também as possibilidades de expansão do mesmo.

1.2 COMPUTAÇÃO QUÂNTICA

Considerando a performance dos computadores clássicos em relação a problemas complexos, Richard Feynman propôs, no início da década de 80, os conceitos iniciais de um computador quântico[4]. Um computador capaz de executar cálculos usando as regras da mecânica quântica, possuindo assim um paradigma de funcionamento e de transformação de informações completamente diferente dos computadores clássicos[5]. Da mesma forma que os computadores clássicos funcionam através de circuitos elétricos e da manipulação de bits, os computadores quânticos utilizam circuitos quânticos, com sua transformação feita através de portas lógicas e tendo como sua unidade principal o "qbit".

1.2.1 Qbit

Assim como os computadores clássicos funcionam através de "bits", unidades fundamentais que podem assumir os valores 1 ou 0, os computadores quânticos possuem os qbits ou "bits quânticos". Os qbits são unidades de informação que, assim como o bit clássico, podem assumir valores 1 ou 0, porém, ao contrário da unidade clássica, os qbits podem assumir um estado de "superposição", que consiste em uma combinação linear entre os dois estados. Uma outra propriedade dos qbits derivada das regras quânticas é o "emaranhamento", um estado onde um qbits é descrito em referência a outra unidade. Considerando sua característica probabilística, pode-se afirmar que o simples ato de medição de um qbit pode alterar seu estado. Podemos representar os vetores da base computacional de um Qbit através da seguinte notação matricial[5]:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Para representar matematicamente um estado quântico de maneira simples, podemos utilizar dois vetores das bases ortonormais $|0\rangle$ e $|1\rangle$:

$$|\psi\rangle = a|0\rangle + b|1\rangle$$

Chamamos a e b de amplitudes de probabilidade de respectivamente $|0\rangle$ e $|1\rangle$. O quadrado da norma desse valor pode ser entendido como a probabilidade do estado quântico colapsar para determinado valor clássico quando for medido. E por ser probabilístico, então:

$$|a|^2 + |b|^2 = 1$$

Além da sua representação vetorial, também podemos ter a representação gráfica de um qbit através da Esfera de Bloch(figura 1.1), onde realizamos a parametrização de $|\psi\rangle$ utilizando a seguinte fórmula:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle$$

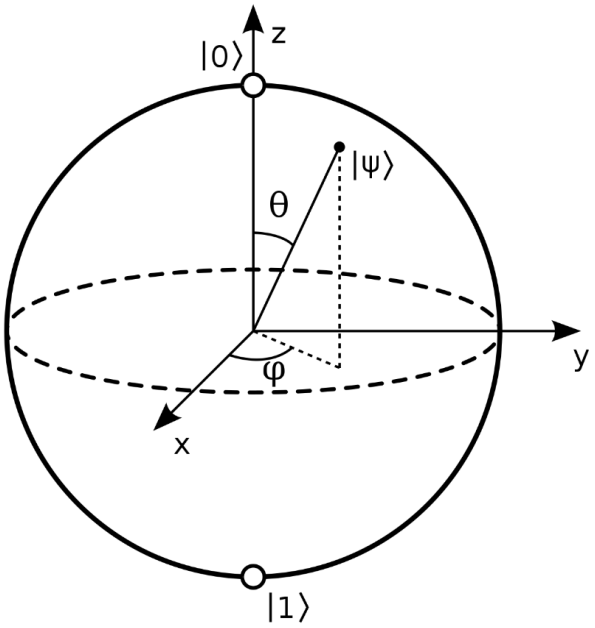


Figura 1.1: Esfera de Bloch

1.2.2 Registrador Quântico

Registradores quânticos são conjuntos de qbits utilizados por computadores quânticos para representar estados e informações. Os cálculos e transformações da informação são realizados através da manipulação dos qbits que compõem um registrador utilizando "Operadores Quânticos".

1.2.3 Operadores Quânticos

Considerando a identidade probabilística dos qbits e dos registradores, os operadores quânticos, também chamados de portas, são matrizes de transformação aplicadas aos registradores quânticos através de um produto tensorial[5]. Tais matrizes são sempre unitárias, garantindo que um resultado da operação seja sempre reversível utilizando um operador/matriz inversa. Tendo em vista o seu uso na execução deste trabalho, serão apresentadas portas comuns, que foram utilizadas para a implementação do modelo.

Pauli X

A porta Pauli X, também conhecida como porta "X" ou porta "NOT", realiza operação similar a porta clássica Not, onde ela executa uma rotação de 180 graus sobre o eixo X,invertendo as amplitudes de 0 e 1. A porta Pauli X tem sua definição pela Matriz:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle$$
$$X\,|\psi\rangle = a\,|1\rangle + b\,|0\rangle$$

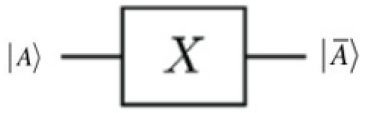


Figura 1.2: Representação da porta Pauli X

Pauli Z

A porta Pauli Z gira os qbits em 180 graus em torno do eixo Z, negando a amplitude do estado 1, sem interferir com o estado 0. Podemos definir a matriz da porta Z como:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle$$

$$Z\,|\psi\rangle = a\,|1\rangle - b\,|1\rangle$$

Figura 1.3: Representação da porta Pauli Z

Hadamard

Quando aplicada em um registrador quântico, esta porta gera uma superposição idêntica para ambos estados(0 e 1), fazendo com que a probabilidade de ambos se torne idêntica. A porta de Hadamard é definida pela Matriz:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H\,|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$H\,|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Figura 1.4: Representação da porta Hadamard

Para um estudo aprofundado de mais operadores quânticos, o leitor pode ler em [9].

1.2.4 Circuito Quântico

Circuitos quânticos são modelos universais de computação para computação quântica, similares aos circuitos clássicos. Circuitos quânticos possuem portas quânticas e "fios" quânticos, que representam o transporte do qbit através do circuito. Com sua leitura iniciada da esquerda para a direita, os fios simples representam os qbits, enquanto os fios duplos representam os bits clássicos. Os itens conectados aos fios, podendo ser portas ou registradores de medida, servem para representar as operações realizadas nos qbits.

Figura 1.5: Representação de um circuito quântico

Na figura 1.5 acima, podemos ver uma representação simples de um circuito quântico; possuindo apenas dois registradores, um registrador quântico no seu primeiro Fio simples, e um registrador clássico sendo mostrado com fios duplos logo abaixo. Podemos ver inicialmente a aplicação da porta de Hadamard no início do circuito, criando uma superposição equilibrada entre as amplitudes quânticas. Logo após isso, uma porta Pauli X é aplicada, gerando assim a negação das suas amplitudes. Considerando que o resultado da primeira porta de hadamard seria uma superposição equilibrada, podemos afirmar que o resultado da porta X sobre essa probabilidade, seria a mesma superposição equilibrada, onde $|a|^2 = 0{,}5$ e $|b|^2 = 0.5$. Após a aplicação das duas portas, seguimos com a medição do nosso registrador quântico para o nosso registrador clássico, resultando assim em uma saída binária relativa a amplitude gerada pelo circuito.

2 PERCEPTRON QUÂNTICO

Introduzido em 1958 por Frank Rosenblatt[1], o Perceptron é um modelo de neurônio artificial utilizado para resolver diversos problemas através de várias entradas que são fornecidas ao neurônio, onde são processadas e tem-se o retorno de uma saída binária.

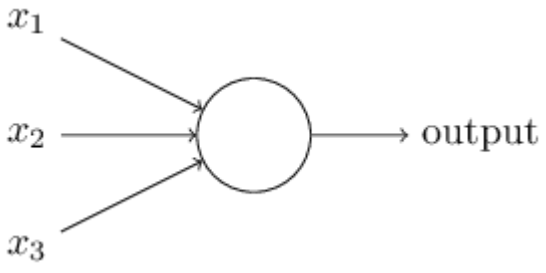


Figura 2.1: Representação gráfica do perceptron quântico

O perceptron utiliza-se de pesos, números reais utilizados para representar a importância de cada entrada, e de um valor limiar para determinar o seu valor de saída. A saída é determinada pela soma ponderada dos seus valores, sendo maior ou maior que o valor limiar, tendo assim a sua saída binária.

$$0 \quad \text{if } \sum_j w_j x_j \leq \text{threshold}$$
$$1 \quad \text{if } \sum_j w_j x_j > \text{threshold}$$

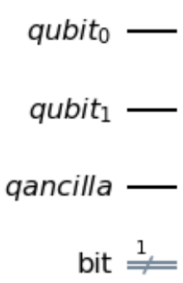
Na equação acima, temos a representação matemática do modelo proposto, onde temos, assim como descrito, a soma ponderada dos valores sendo comparada com o valor limiar(threshold).

2.1 CONTINUOUSLY VALUED QUANTUM NEURON

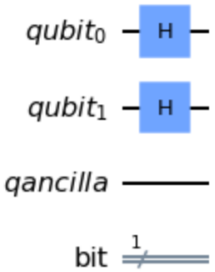
Em seu trabalho de 2020[3], Mangini et al propõem uma extensão ao modelo proposto por Tacchino et al em seu trabalho[2]. O modelo do neurônio quântico proposto nesta feita estende a sua proposta que antes apenas aceitava valores binários em seus vetores, para agora ser capaz de aceitar valores discretos em seus vetores de entrada e pesos. Sua função de saída modela um produto interno entre os dois vetores de entrada e peso de tamanho N, necessitando $n = log_2 N$ qubits para armazená-los. O produto interno é realizado em uma espaço diferente, conforme podemos ver na equação abaixo e trata-se da probabilidade de medir o estado quântico |1> no qubit auxiliar:

$$f(\theta, \phi) = |\frac{1}{N}\sum_{k=0}^{N-1} e^{i(\theta_k - \phi_k)}|^2$$

onde $\theta = [\theta_0, \theta_1 \ldots, \theta_{N-1}]$ e $\phi = [\phi_0, \phi_1 \ldots, \phi_{N-1}]$ são os vetores de entrada e peso respectivamente Criamos assim um circuito contendo um registrador quântico contendo n qbits, sendo acompanhado por um registrador quântico auxiliar, e um registrador clássico para a medição. Considerando um vetor de entrada com N = 4, obtemos o seguinte circuito:



Após a criação do circuito, iniciamos a operação pela aplicação de portões de Hadamard pelo seus qbits, ignorando por hora o nosso qbit auxiliar, criando assim uma superposição uniforme entre suas probabilidades.



Assim, o modelo segue sua aplicação mirando em cada elemento dos seus vetores fornecidos, e aplicando neles as suas fases específicas. A mudança de fase específica definida por Mangini et al[3], é equivalente a uma combinação de portões X, e de portas CRZ multi controladas, onde os valores dos vetores são os valores das rotações desta porta. A construção da estrutura para o vetor de pesos é feita da mesma maneira que o vetor de entrada, com a mesma combinação de portas:

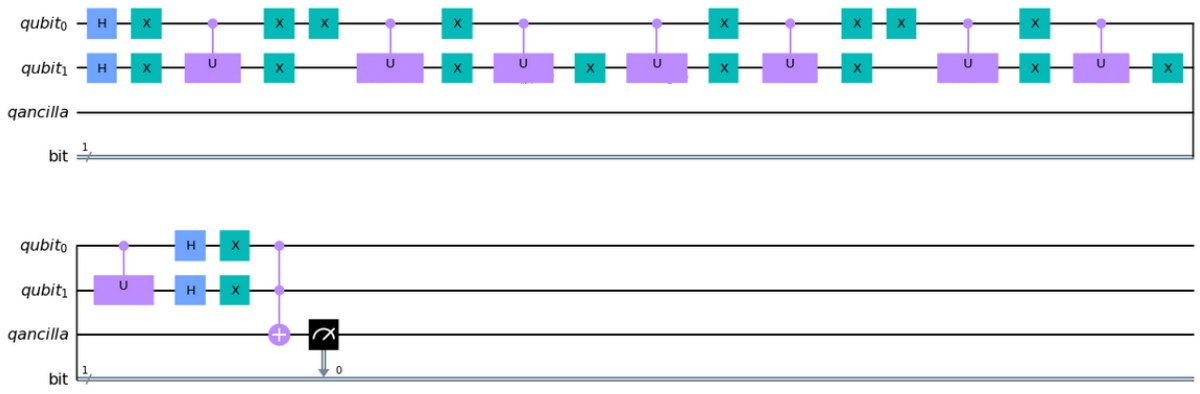


Figura 2.2: Representação do circuito proposto por Mangini et al

Para extrair o resultado final, como mostrado acima, um último grupo de portões H e X são aplicados em todos os registradores quânticos, e após isso, um portão X multi-controlado é aplicado para carregar o resultado no qbit auxiliar. Por fim, a medição do qbit auxiliar se encarrega de calcular o produto interno como amplitude de probabilidade do estado da base |1> do qubit auxiliar. Desta forma, o produto interno pode ser medido indiretamente por várias medições deste circuito, gerando um histograma de probabilidade. O

valor aproximado desse produto interno está em função da probabilidade do estado |1> ser lido.

3 EXPERIMENTO

Nesta seção, iremos apresentar as ferramentas utilizadas para a realização do experimento de classificação usando o modelo proposto por Mangini e colaboradores, bem como também os resultados obtidos na sua execução.

3.1 Metodologia

3.1.1 Linguagem

Para facilitar a implementação e auxiliar na execução do experimento, utilizamos a linguagem "Python" para este trabalho. Além de ser uma linguagem de fácil leitura e entendimento, acelerando o processo de desenvolvimento, Python também conta com uma série de ferramentas pré-implementadas que auxiliam no desenvolvimento de algoritmos na área de computação Quântica. A versão utilizada para o desenvolvimento foi a 3.9.7.

3.1.2 Anaconda

Anaconda é uma plataforma de código aberto criada para auxiliar o desenvolvimento de código relacionado à aprendizagem de máquina e ciência de dados. A plataforma possui diversas bibliotecas prontas para serem instaladas e utilizadas com apenas um clique, e foi utilizada neste experimento para a configuração do ambiente de desenvolvimento.

3.1.3 Jupyter Notebook

Jupyter Notebook é uma aplicação web de código aberto interativa, feita para desenvolvimento de código e análise de dados. Seu uso permite o desenvolvimento, gerenciamento e organização do fluxo de dados de forma facilitada, auxiliando assim não só a criação de código, mas também a análise dos dados que estão sendo executados em tempo real. Nele foram executados todos os algoritmos desenvolvidos neste trabalho, com a plataforma sendo executada em conjunto da plataforma Anaconda mencionada anteriormente.

3.1.4 Qiskit

Qiskit é um kit de desenvolvimento de software de código aberto criado para trabalhar com computadores quânticos. Ele permite a criação e manipulação de programas quânticos e a sua execução em protótipos quânticos através da nuvem pela plataforma “IBM Quantum Experience” ou a sua simulação local em um computador clássico.

3.2 Base de Dados

Para execução da implementação, foi escolhida a base de dados do Quickdraw. Criado como um jogo online, “Quick, Draw!” É uma aplicação criada pela Google que desafia seus jogadores a desenharem um objeto ou ideia que seja informado na tela. Logo em seguida, o jogo utiliza uma rede neural para "adivinhar" o que está sendo desenhado. Cada desenho feito pelos usuários é salvo na sua base de dados que é utilizado para alimentar a sua rede neural, tornando-a mais inteligente com cada interação. Hoje em dia, a base de dados do QuickDraw está disponível publicamente em seu github [6], podendo ser baixada por qualquer pessoa que tenha interesse. A base de dados é separada por categorias de objetos/imagens que foram desenhadas pelos usuários.

3.3 Implementação

Utilizando as tecnologias mencionadas na seção anterior e o modelo proposto por Mangini, seguimos o início da implementação como é demonstrado no código anexado(apêndice A):

```
def geraNeuronio(entrada, peso):
    #Definir quantidade de qBits
    qtdQubit = math.ceil(math.log(len(entrada),2))
    #Criação do registrador quântico
    quantumRegister = QuantumRegister(qtdQubit,'qubit')
    #Criação dos registradores auxiliar e clássico para medição
    quantumAncilla = QuantumRegister(1,'qancilla')
    classicalRegister = ClassicalRegister(1, 'bit')
    #Criação do circuito quântico
    circuito = QuantumCircuit(quantumRegister,quantumAncilla, classicalRegister)
```

A implementação inicia-se calculando o número de qbits baseado no tamanho do vetor de entrada e a criação dos registradores necessários para o circuito. São criados um registrador quântico para cada número de qbits, e mais um registrador quântico auxiliar para ajudar na medição.

```
#Aplicação da porta H por todo registrador quântico
for i in range(qtdQubit):
    circuito.h(quantumRegister[i])
```

Após a criação de todos os componentes do circuito, iniciamos o uso das portas aplicando a porta de Hadamard nos nossos registradores quânticos, ignorando o registrador auxiliar por enquanto.

```
#Execução da lógica do modelo para entrada e pesos
for i in range(len(entrada)):
    binario = str(bin(i))[2:].zfill(qtdQubit)
    for x in range(len(binario)):
        if binario[x] == '0':
            circuito.x(quantumRegister[x])
    circuito.mcrz(entrada[i], quantumRegister[:qtdQubit-1],quantumRegister[qtdQubit-1:][0])
    for x in range(len(binario)):
        if binario[x] == '0':
            circuito.x(quantumRegister[x])
for i in range(len(peso)):
    binario = str(bin(i))[2:].zfill(qtdQubit)
    for x in range(len(binario)):
        if binario[x] == '0':
            circuito.x(quantumRegister[x])
    circuito.mcrz(peso[i], quantumRegister[:qtdQubit-1],quantumRegister[qtdQubit-1:][0])
    for x in range(len(binario)):
        if binario[x] == '0':
            circuito.x(quantumRegister[x])
```

Seguindo com a implementação, seguimos para a parte principal da lógica do algoritmo. Considerando o vetor de entrada, extraímos o valor binário da “posição” de cada elemento do vetor, aplicando a porta X nos caracteres do binário que forem iguais a ‘0’. Após isso, aplicamos a porta mcrz no nosso circuito, e repetimos o processo de aplicar a porta X nos valores ‘0’ do binário. Tendo feito isso, executamos o mesmo processo para o nosso vetor de pesos, seguindo o mesmo passo a passo efetuado para o vetor de entrada.

```
#Aplicação da porta H por todo registrador quântico
for i in range(qtdQubit):
    circuito.h(quantumRegister[i])
#Aplicação da porta X por todo registrador quântico
for i in range(qtdQubit):
    circuito.x(quantumRegister[i])
```

Tendo completado os passos para os vetores de entrada e pesos, encerramos o processo aplicando novamente a porta de Hadamard nos nossos registradores quânticos, e logo em seguida aplicamos a porta X, também nos mesmos registradores.

```
#Aplicação da porta X multicontrolada
circuito.mcx(quantumRegister, quantumAncilla)
#Medição para registrador clássico
circuito.measure(quantumAncilla,classicalRegister)
return circuito
```

Nesse momento, efetuamos a primeira operação no nosso registrador auxiliar, aplicando a porta mcx passando nossos registradores quânticos como qbits de controle, tendo nosso qbit auxiliar como qbit alvo. Por fim, executamos a função de medição passando como parâmetro nosso qbit auxiliar e nosso registrador clássico.

3.4 Execução

Para execução do experimento, utilizamos dois conjuntos de imagens específicos vindos da base de dados do QuickDraw: Airplane e Boomerang, que são equivalentes a imagens de aviões e boomerangs desenhadas pelos usuários da aplicação. Foram analisados 30 itens de cada conjunto para a classificação. Cada imagem é recebida da base de dados como um vetor de pixels de tamanho 28x28 com o desenho feito em escala de cinza. Estes pixels, por sua vez, possuem valores que vão de 0 a 255, valores que são normalizados para entre 0 a π/2 antes da execução, valor recomendado na proposta do modelo[3]. A função usada para normalização foi a seguinte:

$$\frac{I}{255} \cdot \frac{\pi}{2}$$

Considerando I como um valor do vetor de entrada, que representa um pixel no intervalo de 0 e 255, dividimos I por 255 e multiplicamos o resultado por π/2, para assim conseguir um resultado no intervalo desejado. Para o vetor de pesos, foi utilizada a função "np.random.uniform()", para gerar vetores de pesos aleatórios do mesmo tamanho do vetor de entrada. Os parâmetros passados para a criação dos pesos foram: size=784, gerando um vetor de tamanho equivalente a um vetor de imagens 28x28 com tamanho igual as amostras, e high=pi/2, definindo o valor máximo dos pesos para igualar o valor máximo alcançado na normalização dos dados da base. Foram gerados 100 vetores de pesos aleatórios para serem executados junto com os 30 conjuntos de imagens, totalizando 3000 execuções com os seguintes valores:

	Avião(Airplane)	**Boomerang**
Min	0.5203981288284556	0.4539011998109918
Max	0.6910185700859761	0.7472270524795972
Média(Average)	0.6027689006642348	0.6348093837716354

Como demonstrado na tabela acima, os tidos como resultado podem ser considerados bem próximos. Considerando que os desenhos são registrados em escala de cinza, os valores dos seus pixels, de maioria preta, acabam sendo bem similares, sendo diferenciados principalmente pela posição de cada pixel e pelo formato do desenho. Podemos observar que o valor mínimo do Boomerang é menor que o do Avião e que ao mesmo tempo o seu número máximo é maior, tendo uma média que fica acima da média do avião. Como valor de limiar(threshold), utilizamos um valor definido manualmente, para observar os diferentes resultados possíveis da classificação.

	Avião(Airplane)	**Boomerang**
Acertos com limiar 0.600	1389	2697
Acertos com limiar 0.700	3000	97
Acertos com limiar 0.620	2275	2276

Na tabela acima, podemos ver a análise dos resultados da execução quando comparados valor limiar(threshold). Com 30 imagens analisadas em cada conjunto de imagens, considerando suas 100 execuções com vetores de pesos diferentes, foi possível obter diversas combinações se comparados a limiares diferentes. Encontrou-se um equilíbrio maior ao utilizar o valor 0.620 como limiar, onde o número de acertos entre os resultados se diferenciou em apenas um, enquanto limiares maiores ou menos mostravam tendências para uma determinada classe. Pode-se observar um número de acertos superior a 2000 considerando o limiar mencionado, enquanto outros valores limiares mostram maiores taxas de acerto para uma classe enquanto sacrifica-se o acerto da outra. A execução foi feita em um computador clássico, com uma CPU Ryzen 5 1600, 32 gigabytes de memória RAM e uma GPU RX 570.

4 CONCLUSÃO

4.1 Considerações Finais

O modelo de neurônio artificial proposto por Mangini et al. permite computar uma operação com vetores de entrada com valores contínuos[3]. Este modelo é um passo crucial em direção à aplicação direta do neurônio artificial em diferentes tarefas e processos de aprendizagem. Dito isto, ainda é cedo para afirmar a sua total aplicabilidade, considerando que a complexidade do algoritmo está longe de ser inexistente. Até mesmo em cálculos de imagens redimensionadas, a execução ainda se mostra custosa para simulação em computadores clássicos, sendo necessário em momentos da implementação, validar a sua lógica através de sua execução matemática para poupar tempo. Em relação a sua precisão, pode-se afirmar que o resultado de mais de 2000 acertos, em 3000 testes para cada classe, foi relativamente satisfatório, isto levando em consideração as áreas que ainda podem ser melhoradas em comparação com o experimento.

4.2 Trabalhos Futuros

Para trabalhos futuros, é útil mencionar o espaço que existe para melhorias na execução e escolha dos pesos. A simulação e execução da classificação com pesos selecionados com base no seu vetor de entrada ao invés de um vetor de pesos gerados aleatoriamente, pode-se provar mais eficaz, trazendo assim melhores resultados ou desempenho. Considerando a recente data de publicação do modelo proposto por Mangini(2020), pode-se esperar melhorias no modelo de execução do neurônio artificial, ou até mesmo melhorias lógicas cuja mudança implicaria em resultados diferentes. No mais, também vale ressaltar que o valor de limiar(threshold) em conjunto com valores de peso cuidadosamente selecionados, também pode acarretar em melhorias na classificação. Diferentes imagens, diferentes conjuntos de dados podem encontrar resultados variados, podendo estar abaixo ou acima da taxa de acerto mencionada na seção anterior. Conforme a tecnologia e o estudo da computação quântica avança, torna-se importante reavaliar este trabalho com as melhorias propostas, para o refinamento dos resultados.

REFERÊNCIAS

[1] Rosenblatt, F. **The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.** Cornell Aeronautical Laboratory, 1958.

[2] Francesco Tacchino. **An artificial neuron implemented on an actual quantum processor.** npj Quantum Inf 5, 26 (2019).

[3] Stefano Mangini. **Quantum computing model of an artificial neuron with continuously valued input data.** Mach. Learn.: Sci. Technol. 1 045008, 2020.

[4] R.P. Feynman. **Simulating physics with computers.** International Journal of Theoretical Physics, 1982.

[5] Débora F. Dias. **Análise de Múltiplas Soluções do Algoritmo de Grover.** Universidade Federal de Pernambuco, 2019.

[6] QuickDraw Database. https://github.com/googlecreativelab/quickdraw-dataset

[7] Numpy package https://numpy.org/

[8] Qiskit SDK https://qiskit.org/

[9] Nielsen, M. A., & Chuang, I. **Quantum computation and quantum information.** Cambridge University Press, 2002.

APÊNDICE A — Algoritmo Mangini

```
import math
from qiskit import QuantumCircuit
from qiskit import QuantumRegister
from qiskit import ClassicalRegister
from qiskit import AncillaRegister

def geraNeuronioMangini(entrada, peso):
    #Definir quantidade de qBits
    qtdQubit = math.ceil(math.log(len(entrada),2))
    #Criação do registrador quântico
    quantumRegister = QuantumRegister(qtdQubit,'qubit')
    #Criação dos registradores auxiliar e clássico para medição
    quantumAncilla = QuantumRegister(1,'qancilla')
    classicalRegister = ClassicalRegister(1, 'bit')
    #Criação do circuito quântico
    circuito = QuantumCircuit(quantumRegister,quantumAncilla, classicalRegister)
    #Aplicação da porta H por todo registrador quântico
    for i in range(qtdQubit):
        circuito.h(quantumRegister[i])
    #Execução da lógica do modelo para entrada e pesos
    for i in range(len(entrada)):
        binario = str(bin(i))[2:].zfill(qtdQubit)
        for x in range(len(binario)):
            if binario[x] == '0':
                circuito.x(quantumRegister[x])
        circuito.mcrz(entrada[i], quantumRegister[:qtdQubit-1],quantumRegister[qtdQubit-1:][0])
        for x in range(len(binario)):
            if binario[x] == '0':
                circuito.x(quantumRegister[x])
    for i in range(len(peso)):
        binario = str(bin(i))[2:].zfill(qtdQubit)
        for x in range(len(binario)):
            if binario[x] == '0':
                circuito.x(quantumRegister[x])
        circuito.mcrz(peso[i], quantumRegister[:qtdQubit-1],quantumRegister[qtdQubit-1:][0])
        for x in range(len(binario)):
            if binario[x] == '0':
                circuito.x(quantumRegister[x])
    #Aplicação da porta H por todo registrador quântico
    for i in range(qtdQubit):
        circuito.h(quantumRegister[i])
    #Aplicação da porta X por todo registrador quântico
    for i in range(qtdQubit):
        circuito.x(quantumRegister[i])
    #Aplicação da porta X multicontrolada
    circuito.mcx(quantumRegister, quantumAncilla)
    #Medição para registrador clássico
    circuito.measure(quantumAncilla,classicalRegister)
    return circuito
```

APÊNDICE B — Resultados da execução

```
def boomePlane(number):
    if number > 0.620:
        return "boome"
    else:
        return "plane"
planeResults = []
boomerangResults = []
for sample in amostrasAirplane:
    planeResults.append(boomePlane(sample))
for sample in amostrasBoomerang:
    boomerangResults.append(boomePlane(sample))

print(planeResults.count("plane"))
print(boomerangResults.count("boome"))
```

```
2275
2276
```