**RELATÓRIO DO LABORATÓRIO 07**

**Nome**: André Augusto Bernabé da Costa Marques **RA**: 22001640

1. **Descrição das Arquiteturas**

Este projeto apresenta a implementação de uma rede neural desenvolvida inteiramente sem o uso de bibliotecas específicas de IA, como TensorFlow ou PyTorch. O modelo é voltado para a classificação de imagens em bases de dados 28x28 pixels, sendo adaptável a diversos datasets, como Fashion MNIST e MNIST. Utiliza-se Python com as bibliotecas numpy para manipulação de arrays e matplotlib para visualização de dados.

A rede neural implementada possui operações essenciais como propagação para frente, retropropagação e atualização de pesos e vieses, sendo possível simular o comportamento de uma rede com diferentes configurações de camadas. O modelo é flexível, adaptando-se a outros conjuntos de dados com facilidade.

**Principais Funções**

A classe NeuralNetwork é a responsável pela construção e treinamento da rede, contendo métodos que manipulam pesos, vieses e valores das camadas. Dentre os principais métodos, destacam-se:

* \_\_init\_\_(images\_train, labels\_train, cam\_escondida, cam\_final, index\_image): Inicializa a rede neural, definindo as camadas e associando as imagens de treino.
* initiateNN(): Atribui pesos aleatórios às conexões entre camadas e inicializa valores das camadas.
* fPropagation(): Realiza a propagação para frente aplicando a função sigmoide.
* bPropagation(): Calcula os gradientes de erro e os propaga de volta pela rede.
* updateWeights() e updateBias(): Atualizam pesos e vieses usando gradientes calculados.
* testeModelo(): Avalia o modelo com dados de teste.
* preverImagem(): Faz a previsão de uma imagem específica.
* copy(): Cria uma cópia da rede neural com seus pesos e vieses.
* verCamadaHidden(): Exibe os pesos da camada oculta como imagem

**Como o Modelo Funciona**

O modelo começa inicializando a rede neural com uma camada de entrada, camadas ocultas configuráveis e uma camada de saída. Os pesos são gerados aleatoriamente, e os vieses são inicialmente definidos como zero. Durante a forward propagation, as imagens são processadas, com os valores de entrada sendo multiplicados pelos pesos, somados aos vieses e normalizados pela função sigmoide.

Em seguida, durante a backpropagation, o erro entre os valores previstos e os valores reais é propagado de volta pela rede, ajustando os pesos e vieses com o objetivo de minimizar esse erro. Os gradientes calculados durante a retropropagação são usados para atualizar os pesos e vieses, de acordo com o fator de aprendizado fornecido.

Após o treinamento, o modelo é testado com um conjunto de dados de validação, e a precisão é registrada a cada época. O modelo que obtiver a melhor precisão é salvo para previsões futuras.

**Flexibilidade do Modelo**

Este código foi projetado para ser flexível, permitindo que seja facilmente adaptado para qualquer conjunto de dados de imagens de 28x28 pixels. Basta ajustar os dados de entrada e os rótulos para que a rede neural possa fazer previsões em diferentes cenários.

**Visualização dos Resultados**

A rede neural também oferece ferramentas para visualizar o desempenho do modelo:

**Precisão por Épocas**: O gráfico "Precisão x Épocas" exibe a evolução da precisão do modelo ao longo das épocas de treinamento.

**Visualização dos Pesos:** O método verCamadaHidden permite visualizar os pesos da camada de entrada como imagens, fornecendo uma visão interessante de como os pesos se ajustam durante o treinamento.

**Metodologia dos Testes**

Ambas as redes neurais utilizam a mesma arquitetura básica, personalizada para o projeto, mas com variações de configuração. A criação de uma rede neural inicia-se com a instância da classe NeuralNetwork, que encapsula os métodos necessários para o treinamento. No projeto, foram utilizadas duas configurações principais: uma com 2 camadas ocultas de 16 neurônios cada, e outra com 1 camada oculta de 1000 neurônios.



O treinamento é realizado por uma função dedicada, que retorna as precisões obtidas e seleciona o melhor modelo. Essa função recebe parâmetros como taxa de aprendizado, tamanho do lote, número de épocas e dados de teste, permitindo a adaptação a diferentes cenários e datasets.



Dessa forma, o modelo se torna adaptável a diferentes cenários e datasets, permitindo um aprendizado eficiente e ajustado às necessidades do problema em questão. O código pode ser acessado por completo no link:

<https://github.com/andrecostamarques/mlp-scratch>

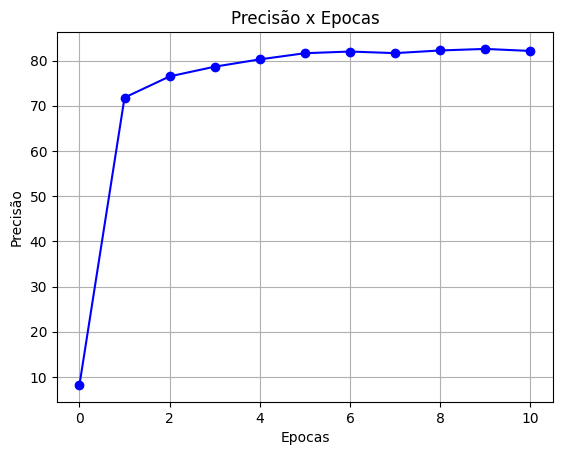
1. **Resultados Alcançados**

Para a Rede Neural com 2 Hidden Layers de 16 neurônios (Vamos chamar de Rede 16, para facilitar) foram definidos como os HiperParâmetros:

* LearningRate = 0.7
* Batch Size = 64
* Epocas = 10

Com essas características, o treinamento retornou o melhor modelo com cerca de 84% de precisão para o Dataset do FashionMnist. Esse mesmo modelo, para o Dataset padrão do Mnist retorna uma precisão de 93%.

Seu crescimento começou a decair aproximadamente na 9ª Época, tendo o seu melhor resultado na 8ª Época.

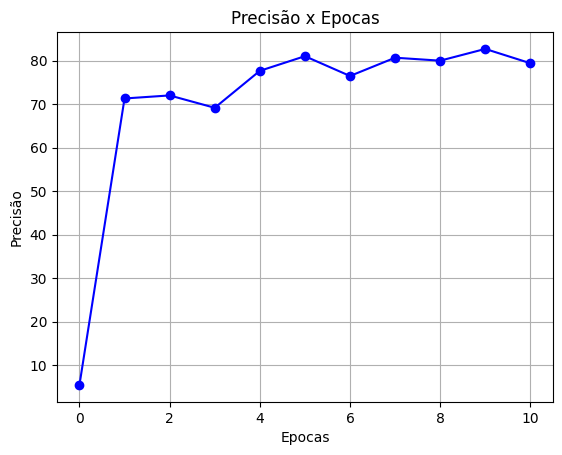


Já a Rede Neural com uma Hidden Layer de 1000 Neurônios (Rede 1000), foram definidos os seguintes hiper parâmetros:

* LearningRate = 0.8
* Batch Size = 32
* Epocas = 10

Com esses hiper parâmetros, o treinamento retornou o melhor modelo com cerca de 82% de precisão para o FashionMnist. Já para o Mnist comum, retornou uma precisão de 95%, superando a Rede16.

Seu crescimento atingiu o pico na 9ª Época, começando a decair a partir da 10ª iteração.



1. **Conclusão**

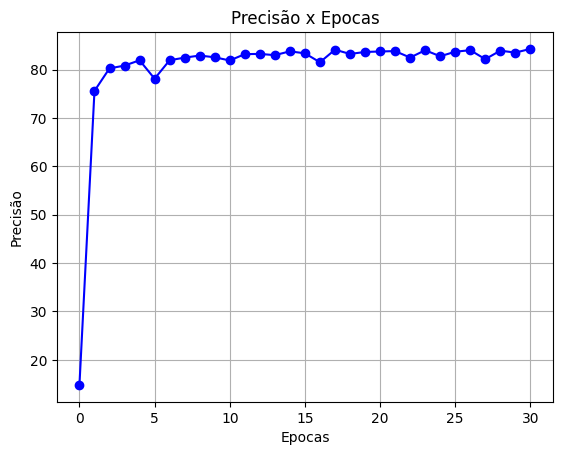
Ao realizar o treinamento para as duas Redes, é possível concluir que ambas atingiram precisões similares, porém, não é correto focar somente nessa característica. O tempo de treinamento para a Rede16 fora de aproximadamente 2min (1min45s+-), enquanto o tempo de treinamento para a Rede1000 fora de aproximadamente 1h50min, ambas para as 10 Épocas.

Portanto, é necessário levar em consideração também, o custo de treinamento para a rede. Além disso, é notável uma pequena diferença enquanto à melhora da precisão, para com, a mudança da estrutura. Ao treinar o modelo com mais enfoque nas camadas e não nos neurônios, é notável uma melhora.

Além disso, há um ponto onde essa mudança se torna negativa também, ao treinar um modelo com 3 Hidden Layers de 16 neurônios, é notável uma perda de precisão, atingindo uma precisão de 80% para o Fashion Mnist.

Fora também testado um modelo com duas Hidden Layers de 32 neurônios, conseguindo atingir uma precisão de 85%, porém, necessitando de mais épocas para o treinamento e custando mais.

Fora também realizado o teste de treinamento em 30 Épocas para a Rede2, todavia, não houve uma mudança significativa da escolha, demonstrando ter atingido o máximo.



Dessa forma, podemos concluir que não é simples decidir a estrutura da rede escolhida, haja visto que, diferentes parâmetros e diferentes escolhas retornaram produtos diferentes, sendo muito difícil qualificar qual é o melhor modelo e melhor estrutura. Porém, podemos confirmar que há sim um *sweet-spot* onde há um custo-benefício entre o tempo de treinamento e o resultado esperado, no contexto desse projeto, fora concluído que o modelo de Rede16 foi o que teve melhor resultado x custo, seguido do modelo Rede32, e por último o modelo de 1000 Neurônios, tendo seu custo reduzindo seu potencial de usabilidade.