Universidade do Algarve

Faculdade de Ciências e Tecnologias

Engenharia Informática

0 or 1 ?

Autor(s)

Márcio Felício a76999
Maria Anjos 79768
Miguel Rosa 76936

IA2425P03G03

Docente(s)

José Valente de Oliveira
Pedro Martins

Relatório apresentado em cumprimento dos requesitos da
Universidade do Algarve na cadeira de Inteligência Artificial para o grau de licenciatura
em *Engenharia Informática*

30 de Novembro, 2024

Indíce

Capítulo 1

Introdução

1.1 Descrição do Problema

Este trabalho aborda o reconhecimento de dígitos manuscritos, com foco específico na distinção entre os dígitos "0"e "1". O problema é uma aplicação clássica de aprendizado de máquina na área de reconhecimento de padrões, utilizando redes neurais artificiais. Para isso, foi utilizado um conjunto de dados simplificado do famoso dataset MNIST, contendo imagens de dígitos em escala de cinza, com resolução de 20x20 pixels.

Cada imagem no conjunto de dados é representada por 400 valores que correspondem à intensidade dos pixels, normalizados para o intervalo [0, 1]. Os rótulos indicam os dígitos associados a cada imagem (0 ou 1). O objetivo principal é treinar uma rede neural capaz de classificar corretamente os dígitos apresentados na entrada, com base nas imagens fornecidas.

1.2 Algoritmos e Estratégias Utilizadas

Para resolver o problema, foi implementada uma rede neural de múltiplas camadas com função de ativação sigmoide, projetada para gerar saídas no intervalo [0, 1]. Essa configuração facilita a interpretação probabilística das classificações. A arquitetura foi baseada no repositório GitHub de Kim Feichtinger, com modificações para atender às especificações do problema. Foram realizadas diversas decisões de projeto para otimizar o desempenho e garantir a generalização:

1.2.1 Estrutura da Rede Neural

Número de neurônios e camadas: A rede foi configurada com:

- 400 neurônios de entrada, correspondendo aos 400 valores dos pixels da imagem.
- 10 neurônios em uma camada oculta, escolhidos para equilibrar desempenho e custo computacional, conforme análises experimentais.
- 1 neurônio de saída, que retorna uma probabilidade para a classificação como "0"ou "1".

Função de ativação: A função sigmoide foi escolhida para todos os neurônios, garantindo que as saídas permaneçam no intervalo [0, 1].

1.2.2 Pré-processamento dos Dados

O pré-processamento dos dados foi realizado pela classe `DataPreprocessor`, com as seguintes etapas:

- **Carregamento e validação dos dados:** Os dados foram carregados a partir dos arquivos `dataset.csv` e `labels.csv`. O programa valida que cada linha de `dataset.csv` contém exatamente 400 valores e que os rótulos em `labels.csv` são binários (0 ou 1).
- **Normalização:** Todos os valores de intensidade dos pixels foram escalados para o intervalo [0, 1], dividindo-os por 255.0.
- **Divisão do conjunto de dados:** O conjunto de dados foi dividido em 80% para treinamento e 20% para teste, garantindo uma avaliação robusta.

1.2.3 Treinamento e Validação

O treinamento e a validação do modelo envolveram as seguintes etapas:

- **Erro médio quadrático (MSE):** O MSE foi utilizado como métrica de desempenho durante o treinamento, sendo monitorado tanto no conjunto de treinamento quanto no de validação.
- **Early Stopping:** Foi implementado um mecanismo de parada antecipada para evitar *overfitting*, interrompendo o treinamento caso o erro de validação não melhorasse após um número definido de iterações (*paciência*).
- **Histórico de treinamento:** Os valores do MSE para treinamento e validação foram registrados e exportados para um arquivo CSV, permitindo a visualização gráfica da evolução do aprendizado.

1.2.4 Avaliação

A avaliação do modelo foi realizada utilizando o conjunto de teste. A classe `DigitTrainer` calcula a acurácia final e exibe os erros de predição. A rede neural foi avaliada com base em:

- **Acurácia:** Percentual de previsões corretas no conjunto de teste.
- **Análise de erros:** Relatório detalhado das previsões incorretas.

1.2.5 Classificação de Novas Entradas

A classe `DigitClassifier` foi projetada para carregar a rede neural treinada e realizar a predição em novas entradas. Para cada linha de entrada contendo 400 valores de pixels, o programa normaliza os dados, processa-os pela rede neural e retorna a classificação final (0 ou 1).

Capítulo 2

Testes Unitários

Os testes unitários desenvolvidos para este projeto encontram-se no diretório `src/main/java/UnitTests` em anexo. Eles foram projetados para verificar a funcionalidade e a confiabilidade de todas as etapas do sistema, garantindo que cada componente do projeto opere conforme esperado. A seguir, destacam-se as principais categorias de testes realizados:

2.1 Testes da Classe `DataPreprocessor`

- **Carregamento de dados:** Verifica se os dados são corretamente carregados a partir dos arquivos `dataset.csv` e `labels.csv`, incluindo a normalização para o intervalo [0, 1] e a validação do número de pixels e rótulos binários (0 ou 1).
- **Validação de formatos:** Testa a detecção de inconsistências nos arquivos, como número incorreto de pixels por linha ou rótulos inválidos.
- **Divisão do conjunto de dados:** Avalia a correta separação dos dados em conjuntos de treinamento e teste, considerando diferentes proporções (ex.: 80% treino e 20% teste).

2.2 Testes da Classe `DigitClassifier`

- **Classificação de entrada válida:** Verifica se a rede neural treinada pode classificar corretamente entradas válidas com 400 valores.
- **Detecção de erros:** Testa o comportamento do sistema ao lidar com entradas inválidas, como número incorreto de pixels ou valores fora do intervalo esperado.
- **Falha ao carregar pesos:** Simula cenários onde o arquivo de pesos da rede neural está ausente ou corrompido, verificando a robustez do sistema.

2.3 Testes da Classe `DigitTrainer`

- **Cálculo do erro médio quadrático (MSE):** Avalia se o MSE é calculado corretamente para um conjunto de dados dado.
- **Treinamento da rede neural:** Testa o processo de treinamento, incluindo a atualização dos pesos e a convergência do MSE.
- **Avaliação do modelo:** Verifica a precisão da rede neural no conjunto de teste, garantindo que os resultados sejam consistentes.

2.4 Testes da Classe `NeuralNetwork`

- **Inicialização:** Testa a criação da rede neural, incluindo a correta inicialização de pesos, vieses e dimensões.
- **Propagação direta (forward propagation):** Verifica se as saídas da rede estão no intervalo esperado [0, 1] devido à função de ativação sigmoide.
- **Treinamento e atualização de pesos:** Avalia se os pesos são corretamente ajustados durante o treinamento.
- **Mutação e fusão:** Testa a capacidade da rede de aplicar mutações nos pesos e de combinar duas redes neurais.
- **Robustez:** Verifica o comportamento da rede ao receber entradas de tamanho incorreto, garantindo que exceções apropriadas sejam lançadas.

Esses testes asseguram a confiabilidade do sistema e contribuem para um projeto mais robusto e fácil de manter.

Capítulo 3

Opções de Projeto Tomadas

3.1 Padrão de Projeto

As decisões para o desenvolvimento deste trabalho foram:

- **Rede neural customizável**: A arquitetura foi baseada no repositório GitHub de Kim Feichtinger, permitindo o ajuste dos hiperparâmetros, como o número de camadas ocultas, neurônios por camada e taxa de aprendizado.
- **Modularidade**: O projeto foi estruturado de forma modular, separando as responsabilidades em diferentes componentes. Essa abordagem facilita tanto a manutenção quanto a expansão futura.
- **Simplicidade computacional**: A rede neural foi configurada com 400 neurônios de entrada, 10 neurônios na camada oculta e 1 neurônio na camada de saída, alcançando um equilíbrio entre desempenho e custo computacional.
- **Prevenção de overfitting**: O conjunto de dados foi dividido em 80% para treinamento e 20% para teste. A técnica de *Early Stopping* foi utilizada para interromper o treinamento caso o erro de validação não apresentasse melhorias significativas.
- **Monitoramento de métricas**: Durante o treinamento, o erro médio quadrático (MSE) foi registrado e exportado para arquivos CSV, permitindo uma análise gráfica detalhada da evolução do aprendizado.

Essas escolhas foram tomadas com o objetivo de garantir um projeto eficiente, organizado e com resultados claros e reprodutíveis.

Capítulo 4

Diagrama de Classes UML 0 or 1 ?

4.1 Apresentação do Diagrama

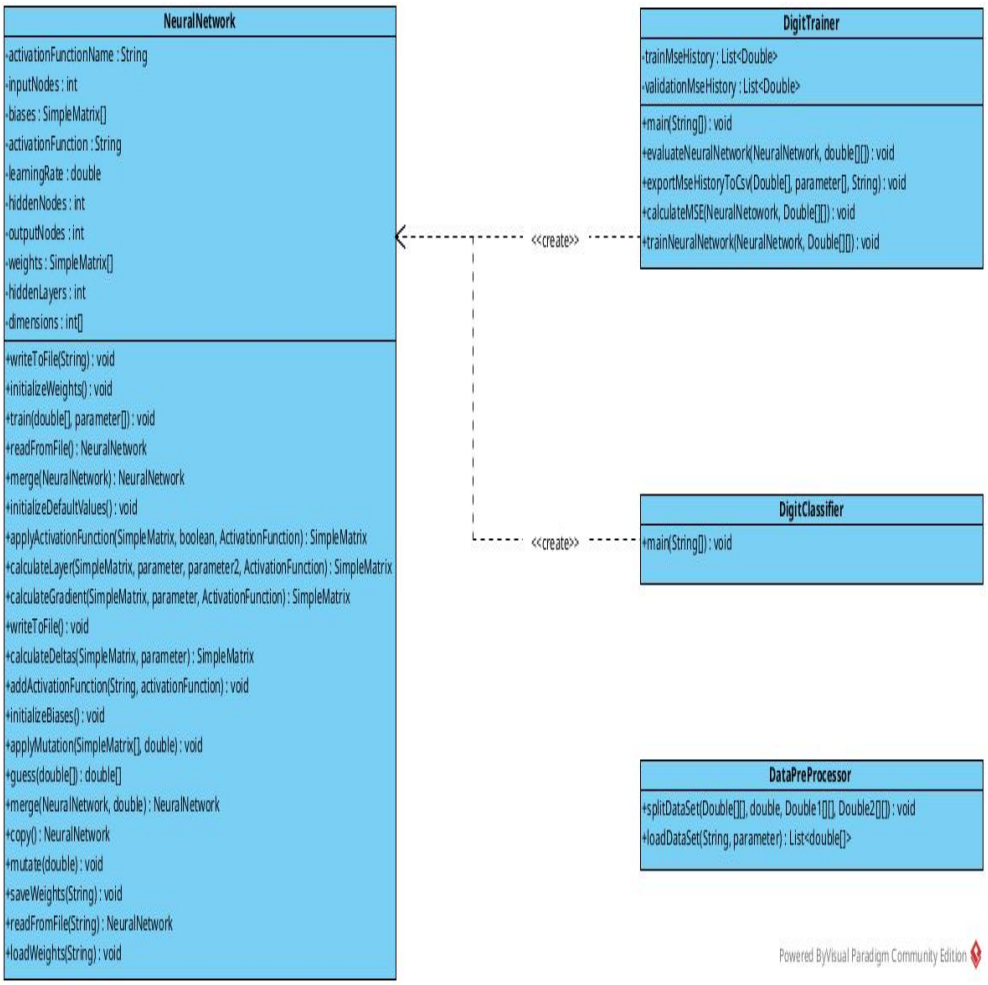

Figure 4.1: Diagrama UML do projeto 0 or 1 ? (Para melhor visualização da imagem, a mesma encontra-se na pasta src/main/java/UML)

Capítulo 5

Resultados, Análises e Discussões

5.1 Resultados

O conjunto de dados utilizado tal como referido acima foi uma versão simplificada do famoso dataset MNIST, contendo 800 imagens de dígitos manuscritos "0"e "1"em escala de cinza, com resolução de 20x20 pixels. Cada imagem é representada por 400 valores correspondentes às intensidades dos pixels.

Foram realizados experimentos utilizando duas proporções distintas para a divisão do conjunto de dados em treinamento e teste: 80% para treinamento e 20% para teste, e 60% para treinamento e 40% para teste. A tabela a seguir resume os resultados obtidos em cada cenário:

Proporção Treino/Teste	Acurácia no Teste	Tempo de Execução (s)	Amostras de Teste
80% Treino / 20% Teste	100%	74.400	160
60% Treino / 40% Teste	99.69%	62.446	320

Table 5.1: Resultados obtidos com diferentes proporções de treinamento e teste.

Em ambos os cenários, a rede neural apresentou um alto nível de acurácia no conjunto de teste. Além disso, foram gerados gráficos mostrando a evolução do erro médio quadrático (MSE) ao longo das iterações para os conjuntos de treinamento e validação em cada proporção de divisão. Esses gráficos estão apresentados nas Figuras 5.1 e 5.2.

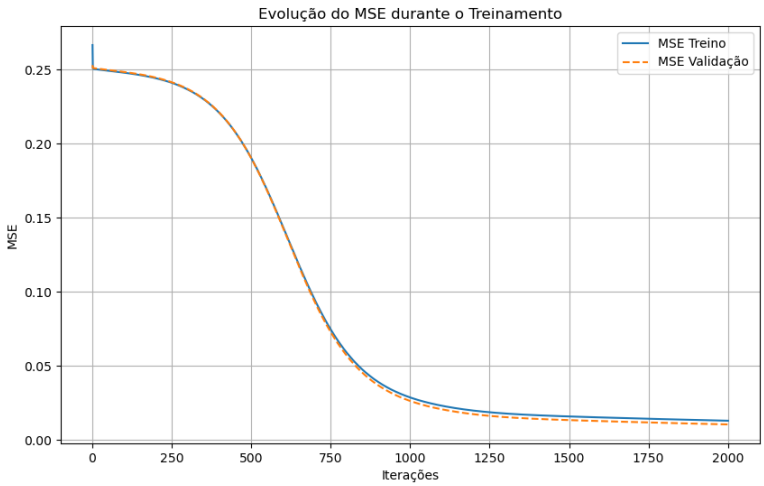

Figure 5.1: Evolução do MSE durante o treinamento para a divisão 80% treinamento e 20% teste.

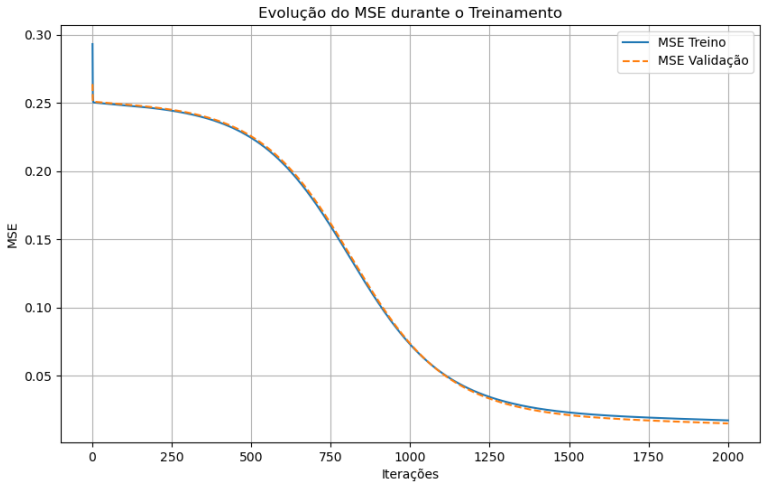

Figure 5.2: Evolução do MSE durante o treinamento para a divisão 60% treinamento e 40% teste.

Os gráficos mostram que o MSE diminui ao longo das iterações em ambos os cenários, indicando que o modelo está aprendendo a partir dos dados de treinamento.

5.2 Análises

Analisando os resultados, observa-se que:

- **Acurácia**: Em ambos os casos, a rede neural alcançou uma acurácia muito alta no conjunto de teste. No cenário com 80% de treinamento, a acurácia foi de 100%, enquanto no cenário com 60% de treinamento, a acurácia foi de 99.69%.

- **Quantidade de dados de teste**: Ao aumentar a proporção de dados de teste de 20% para 40%, o número de amostras de teste dobrou (de 160 para 320). Mesmo com mais dados para avaliar, a rede manteve um desempenho excelente.
- **Tempo de execução**: O tempo total de execução foi menor no cenário com 60% de treinamento, devido ao menor número de amostras utilizadas para treinar a rede neural.
- **Evolução do MSE**: Os gráficos do MSE mostram uma convergência similar em ambos os cenários, com o erro diminuindo rapidamente nas primeiras iterações e estabilizando posteriormente.

A pequena redução na acurácia (de 100% para 99.69%) ao utilizar menos dados para treinamento é esperada, pois a rede teve menos exemplos para aprender os padrões dos dígitos. No entanto, essa diferença é mínima e indica que o modelo generaliza bem mesmo com menos dados de treinamento.

5.3 Discussões

Os resultados mostraram que, mesmo com menos dados de treinamento, o modelo manteve uma acurácia muito alta no conjunto de teste (99,69%). Isso indica que o modelo não está sofrendo de *overfitting* significativo, pois consegue generalizar bem para novos dados apesar da redução no conjunto de treinamento.

Considerando que o problema envolve apenas dois dígitos ("0"e "1") e que o conjunto de dados possui 800 amostras, a utilização das proporções 80/20 e 60/40 é apropriada. Os gráficos do MSE em ambos os cenários confirmam que o modelo está convergindo corretamente, sem sinais de *overfitting*.

Concluímos que testar diferentes proporções de divisão foi uma abordagem válida, fornecendo insights valiosos sobre o desempenho e a capacidade de generalização do modelo em condições distintas.

Capítulo 6

Conclusão

Desenvolvemos uma rede neural para distinguir entre os dígitos manuscritos "0"e "1"utilizando uma versão simplificada do dataset MNIST. Após normalizar e pré-processar os dados, treinamos o modelo com proporções de divisão de 80%/20% e 60%/40% entre treinamento e teste.

Os resultados mostraram alta acurácia em ambos os cenários (100% e 99,69%), evidenciando a robustez e a capacidade de generalização do modelo sem sinais significativos de *overfitting*. A redução mínima na acurácia com menos dados de treinamento confirma a eficácia da arquitetura escolhida e das decisões de projeto tomadas.

Concluímos que os objetivos foram plenamente atingidos. O modelo não apenas distingue eficazmente entre "0"e "1", mas também demonstra potencial para adaptações futuras. Para trabalhos subsequentes, seria interessante expandir o modelo para reconhecer todos os dígitos de 0 a 9 e explorar arquiteturas mais complexas ou técnicas adicionais de validação.

Capítulo 7

Bibliografia

LaTeX
Disponível em: https://www.latex-project.org/.

Zotero
Disponível em: https://www.zotero.org/.

Kim-Marcel
"Basic Neural Network." *GitHub*, 3 de dezembro de 2023. Disponível em: https://github.com/kim-marcel/basic_neural_network. Acesso em: 5 de dezembro de 2024.

Russell, Stuart; Norvig, Peter; Negnevitsky, Michael
"Backpropagation (cont.)." Disponível em: https://tutoria.ualg.pt/2024/pluginfile.php/174959/mod_resource/content/1/ia2024-25T17.pdf.

Russell, Stuart; Norvig, Peter; Negnevitsky, Michael
"Backpropagation (Training and Design Issues)." Disponível em: https://tutoria.ualg.pt/2024/pluginfile.php/178130/mod_resource/content/2/ia2024-25T18.pdf.