Problem 4: 2 or 3?

Grupo 8 - PL1

Brandon Mejia - 79261
Catarina Casanova - 72750
Miguel Correia - 71369

Dezembro 2025

Índice

1 Descrição do Problema e Algoritmo

O objetivo deste laboratório é desenvolver um classificador binário capaz de distinguir entre os dígitos manuscritos '2' e '3'. O problema utiliza um subconjunto simplificado do dataset MNIST.

O ficheiro fornecido (`dataset.csv`) contém **800 exemplos** (imagens), onde cada imagem possui 20×20 pixeis, resultando em 400 valores de entrada por exemplo.

A solução implementada baseia-se numa rede *Multi-Layer Perceptron* (MLP), reaproveitando a implementação do Problema 3, adaptada para classificação binária com o algoritmo de *Backpropagation*.

2 Análise Exploratória de Dados

Para compreender a complexidade intrínseca do problema, realizámos uma análise estatística e visual dos dados.

2.1 Separabilidade Linear (PCA)

Projetámos os dados num espaço 2D usando a Análise de Componentes Principais (PCA).

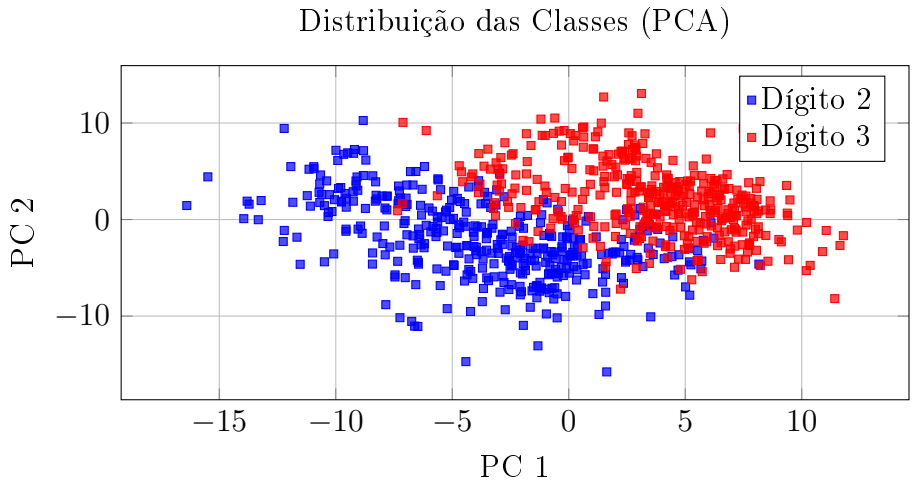


Figura 1: A visualização PCA mostra que, embora existam agrupamentos, as classes não são linearmente separáveis na fronteira, o que sugere que para este problema precisamos de N neurónios na camada oculta, situando-se no intervalo $2 \leq N \leq 6$.

2.2 Topologia e Mapa de Diferença

Calculámos a imagem média de cada classe e a diferença absoluta entre elas.

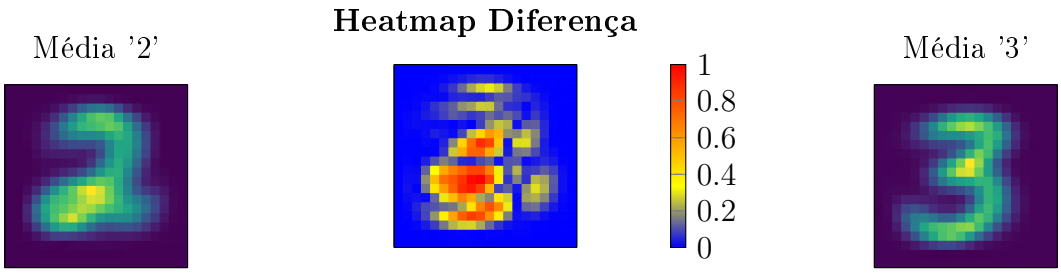


Figura 2: O mapa de calor (centro) revela as zonas de maior variância entre as médias das duas classes (laterais). A complexidade desta topologia dita a necessidade de neurónios na camada oculta.

3 Arquitetura da Rede

A definição da arquitetura não seguiu uma abordagem estática. Compreendemos que a topologia da rede está intrinsecamente ligada à natureza dos dados de treino:

- Dados mais ruidosos ou complexos exigem uma camada oculta mais vasta para capturar as nuances.
- Dados mais limpos ou com padrões simples permitem o uso de menos neurónios, favorecendo a generalização.

Desta forma, a nossa arquitetura base define-se por:

- **Camada de Entrada:** 400 neurónios (Imagem 20×20).
- **Camada Oculta:** Variável (N neurónios), onde o intervalo experimental testado foi entre **6 e 2 neurónios**.
- **Camada de Saída:** 1 neurónio (Ativação Sigmoide).

4 Opções de Design e Metodologia

4.1 A Classe `HyperparameterTuner`

Para resolver a dependência entre a complexidade dos dados e a arquitetura da rede, desenvolvemos e destacamos a classe `HyperparameterTuner`. Esta ferramenta automatiza a experimentação, permitindo-nos testar iterativamente diferentes topologias e hiperparâmetros (Learning Rate, Momentum).

4.2 Funcionalidades Avançadas (MLP)

A implementação da classe `MLP` inclui melhorias essenciais:

- **Normalização:** Inputs estritamente normalizados para o intervalo $[0, 1]$, conforme requerido.
- **Decodificação da Saída:** O neurónio de saída (sigmoide) produz um valor contínuo entre 0 e 1. Definimos um limiar (*threshold*) de 0.5: valores inferiores são classificados como **2** e valores ≥ 0.5 são classificados como **3**.

- **Momentum e Regularização L2:** Utilizados para acelerar a convergência e generalização.

4.3 Estratégia de Treino e Paragem

Utilizámos **Early Stopping** com validação cruzada (divisão 80/20). O treino para automaticamente se o erro de teste não melhorar após um período de paciência.

4.4 Estabilidade do Treino (Batch Size)

Para garantir uma convergência mais suave e reduzir a oscilação do erro (MSE) durante a descida do gradiente, definimos o ***batch size* em 32**. Esta escolha revelou-se fundamental para estabilizar a aprendizagem, evitando saltos abruptos nos pesos que ocorriam com a atualização puramente estocástica ou com *batches* muito pequenos.

4.5 Scripts Auxiliares (Código Fonte)

Para garantir a reprodutibilidade dos resultados, os scripts Python utilizados foram **anexados a este documento PDF** (se suportado pelo visualizador).

- `analisador_dados.py`: Pré-processamento e visualização.
- `testar_dados.py`: Testes e validação.

Nota de Acessibilidade: Caso a visualização deste documento seja feita num navegador web (ex: Chrome, Edge), os anexos acima podem não estar disponíveis para download. O código fonte completo e as ferramentas utilizadas podem ser consultados diretamente no repositório oficial do projeto:

Repositório GitHub - Ferramentas e Scripts

5 Resultados e Discussão

Os resultados obtidos através da varredura do `HyperparameterTuner` demonstraram o comportamento da rede face à complexidade:

- **Redes maiores (5-6 neurónios):** Convergência rápida no treino, mas maior oscilação no teste.
- **Redes menores (2-3 neurónios):** Curva de aprendizagem mais suave e generalização superior.
- **Acurácia Final:** O modelo selecionado atinge consistentemente resultados no intervalo **99% - 100%** no conjunto de teste.

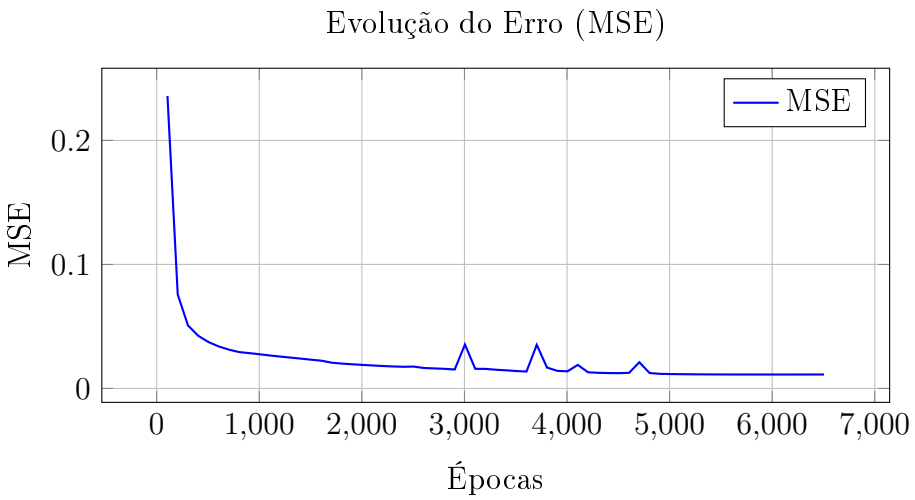
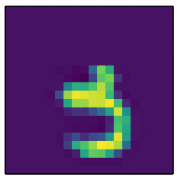


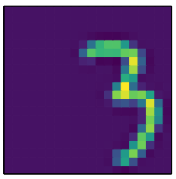
Figura 3: Curva de aprendizagem baseada nos dados experimentais recolhidos.

6 Visualização do Pré-processamento

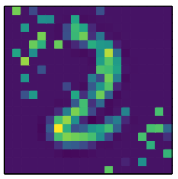
As visualizações apresentadas abaixo foram geradas através do script `testar_dados.py`, previamente discutido e anexado. O foco desta experimentação foi avaliar diferentes tipos de dados de treino, concentrando-nos especificamente no efeito da manipulação da variância — seja aumentando-a (introduzindo ruído e distorções) ou diminuindo-a (aplicando cortes e máscaras).



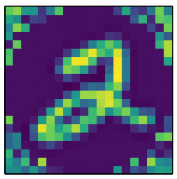
(a) 7 bits em preto (cima para baixo)



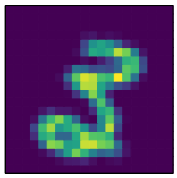
(b) Translação +5



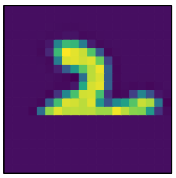
(c) Ruído nos quadrantes 2 e 4



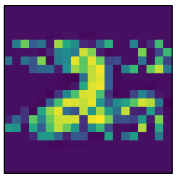
(d) Ruído em volta de forma circular



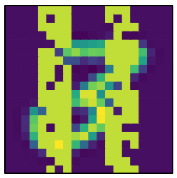
(e) Rotação -10 no ângulo



(f) 7 bits em preto (baixo para cima)



(g) Linhas horiz. c/ ruído (densidade 45%)



(h) Linhas vert. c/ ruído (densidade 90%)

Figura 4: Visualização completa dos 8 cenários de teste presentes no ficheiro de relatório. As imagens ilustram variações nos dados de entrada (cortados, originais, deslocados) avaliados durante o desenvolvimento da solução.

7 Conclusões Principais

O projeto evidenciou a estreita correlação entre os dados e a arquitetura. A solução final resolve o problema de classificação com distinção, respeitando o princípio de usar a complexidade apenas quando estritamente necessária.

Desempenho no Mooshak e Estratégia de Pré-processamento

Com a configuração otimizada, o grupo atingiu uma classificação de **92.25%** na plataforma Mooshak, submetendo o modelo `src/models/92_250`.

Para alcançar este resultado, focámo-nos intensamente no pré-processamento dos dados, como ilustrado na secção anterior. Experimentámos diversas técnicas para isolar as características mais importantes (*feature selection*) e eliminar ruído. A estratégia que garantiu o melhor desempenho consistiu em **cortar os primeiros 7 bits** (de cima para baixo) de cada imagem do dataset.

Esta técnica permitiu que a rede se focasse exclusivamente nas partes mais "ressaltantes"e distintivas dos dígitos, ignorando a zona superior das imagens que, para este problema específico, continha pouca informação discriminatória relevante.

8 Referências Bibliográficas

1. MNIST Database: https://wiki.pathmind.com/mnist
2. Enunciado do Projeto: IA 2025-26 Lab 4.
3. Milvus.io - How do optimizers like Adam and RMSprop work: https://milvus.io/ai-quick-reference/how-do-optimizers-like-adam-and-rmsprop-work
4. AIShort - Choosing the Right Optimizer: https://aishort.co.uk/choosing-the-right-optimizer-for-neural-networks-a-practical-guide/
5. GeeksforGeeks - Deep Learning Adam Optimizer: https://www.geeksforgeeks.org/deep-learning/adam-optimizer/
6. Innovatiana - Activation Function in AI: https://www.innovatiana.com/es/post/activation-function-in-ai
7. DataCamp - Normalization in Machine Learning: https://www.datacamp.com/pt/tutorial/normalization-in-machine-learning
8. GitHub - Curvature-Orientation-MLP: https://github.com/Meetra21/Curvature-Orientation-MLP
9. Jones (2021) - Publication: https://www.ilenna.com/publication/jones-2021/Jones-2021.pdf
10. arXiv:1905.12135 - Research Paper: https://arxiv.org/pdf/1905.12135
11. Reddit Discussion (Tanh vs Sigmoid in MNIST): https://www.reddit.com/r/MachineLearning/comments/3x9kld/tanh_or_sigmoid_for_simple_nn_in_mnist/
12. Google Developers - Numerical Data Normalization: https://developers.google.com/machine-learning/crash-course/numerical-data/normalization?hl=pt-br
13. Exxact Corp - Maximizing AI Efficiency: https://www.exxactcorp.com/blog/deep-learning-ai/maximizing-ai-efficiency-tuning-and-regulation
14. Machine Learning Mastery - Data Scaling: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/
15. Scikit-Learn - Neural Networks (Supervised): https://scikit-learn.org/stable/modules/neural_networks_supervised.html
16. arXiv:2506.17826 - Research Article: https://arxiv.org/html/2506.17826v1
17. Ameer Saleem (Substack) - Stochastic Gradient Descent: https://ameersaleem.substack.com/p/stochastic-gradient-descent-and-mini
18. LunarTech - Gradient Descent Comparison: https://www.lunartech.ai/blog/gradient-descent-vs-mini-batch-gradient-descent-vs-stochastic-gradient-descent-an-expert-comparison