UC Aprendizagem Profunda

Trabalho prático em grupo - Módulo 1

César Cardoso, Gonçalo Brandão, Guilherme Rio, LingYun Zhu, and Gustavo Gomes

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {pg57870,pg57874,pg57875,pg57885,pg58105}@alunos.uminho.pt

1 Introdução

Este projeto prático consiste no desenvolvimento de modelos de Machine/ Deep Learning para identificar textos gerados por Inteligência Artificial e textos escritos por seres humanos. O presente relatório visa descrever o processo de desenvolvimento, abordando a metodologia adotada para a construção dos dados de treino, o desenvolvimento de modelos de implementação própria e por Tensorflow e a análise dos resultados obtidos.

2 Tarefa 1: Construção dos Datasets

A uma primeira fase a escolha dos dados utilizados para treinar os nossos modelos.

Para tal tarefa decidimos utilizar os datasetes fornecidos pela equipa docente de modo a criarmos o Detetor de Textos AI ou Human de forma clara.

Criamos um notebook, dataset_standerizer.ipynb que transforma os datasets na forma stander de Id, text e Label. Também será importante referir que decidimos colocar todos os Exemple Text dentro de aspas para garantir uma boa definição de inicio e de fim de frase.

ID	text	Label
D(ID Dataset) - (Linha Dataset)	"EXEMPLO TEXT"	Human or AI

3 Tarefa 2: Modelos com Implementação Própria/Numpy

Nesta tarefa foram implementados modelos utilizando exclusivamente o numpy, sem a utilização de bibliotecas de machine learning ou deep learning como scikit-learn, TensorFlow/Keras ou PyTorch.

3.1 Deep Neural Networks (DNNs)

A estrutura da rede é definida por de uma lista de tamanhos de camadas (layer_sizes), na qual cada elemento especifica o número de neurônios de uma camada das correspondente. As funções de ativação associadas a cada camada são determinadas por outra lista separada (activation_functions), com suporte a funções como ReLU, Sigmoid, Tanh e Softmax. A inicialização dos pesos é realizada utilizando o método de He, o qual favorece a convergência em redes profundas, enquanto os bias são inicializados com vetores de zeros.

Implementamos diferentes loss functions, como Binary Crossentropy, Mean Squared Error (MSE) e Categorical Crossentropy, adequadas para classificação binária e regressão. Para mitigar o problema de overfitting, implementamos técnicas de regularização, como:

- **Dropout**: Técnica que desativa aleatoriamente neurônios durante o treinamento, contribuindo para a generalização do modelo.
- **Regularização L2**: Método que penaliza pesos excessivamente grandes, promovendo a estabilidade do modelo.

Implementamos tambem dois optimizadores:

- **Gradiente Descendente Estocástico (SGD):** Método clássico de otimização.
- **Adam**: Otimizador que utiliza momentos de primeira e segunda ordem para acelerar a convergência, incluindo correções de viés para garantir atualizações mais precisas.

O treino é realizado utilizando mini-batches, com o shufle dos dados a cada époc. Durante a forward propagation, as ativações de cada camada são calculadas. Em seguida, na fase backpropagation, os gradientes dos pesos e bias são computados em relação à função de perda, permitindo a atualização dos parâmetros conforme o otimizador selecionado. Adicionalmente, a implementação possui um Early Stopping, que interrompe o treino caso validação não apresente melhoria após um número predefinido de épocas consecutivas. Durante o treino, métricas como perda e accuracy são registradas para os conjuntos de treinamento e validação, possibilitando uma monitorização contínua do modelo.

3.2 Recurrent neural networks(RNNs)

A classe RNN definida em Numpy recebe parâmetros como o número de características de entrada, input_size, unidades na camada oculta ,hidden_size, unidades na saída output_size, pré_definida para 1 pois estamos perante um problema de classificação binária. A taxa de aprendizado (lr) e taxa de dropout (dropout_rate). Os pesos e bias são inicializados aleatoriamente, com os pesos conectados á entrada à camada oculta, as conexões recorrentes e a ligação da camada oculta à saída. Durante o forward pass, as entradas são processadas sequencialmente com a função *tanh* para os estados ocultos e *sigmoid* para a saída, aplicamos o dropout para prevenir overfitting. Já no backward pass, utiliza-se o método de backpropagation through time (BPTT) para acumulação dos gradientes, e o treino é conduzido com entropia cruzada binária e gradiente descendente. Na fase de previsao, o método predict desativa o dropout para gerar previsões consistentes.

O pipeline de treino, no notebook rnn_pipelineipynb, inicia com o carregamento dos conjuntos de dados de treino, teste e validação, seguido pelo pré-processamento utilizando a abordagem de *unigram* que transforma os textos em vetores numéricos por e o mapeamento dos rótulos (AI para 1 e Human para 0). A função train_evaluate_rnn reestrutura os dados para o formato esperado pela rede, treina a RNN por 100 epochs com parâmetros como hidden_size=64, dropout_rate=0.2 e lr=0.08, e exibe métricas de classificação e a matriz de confusão. Ao final, o modelo treinado e o vetorizador são salvos para uso futuro na inferência.

4 Tarefa 3: Modelos com Implementação em Tensorflow e Large Language Models

4.1 DNN Tabular

No notebook "train_dnn_tabular.ipynb" implementamos os primeiros modelos em keras. Usamos a camada TextVectorization para experimentar diversas estratégias de bag of words. Comparamos 3 estratégias e uniformizou-se removendo a pontuação e transformando em minúsculas.

- Unigrams com multi hot encoding
- Bigrams com multi hot encoding
- Bigrams com normalização tf-idf

Esscolhemos uma arquitetura de DNN que produziu um treino adequado para as 3 estratégias, para tal analisamos o gráfico de Training e Validation Loss e experimentamos até ambas estabilizarem. Os melhores parâmetros que encontramos foram 3 camadas com as respectivas dimensões 64, 32, 16, "relu" de ativação e um dropout de 0.2. A ultima camada é de um neurónio com ativação sigmoid. Usamos também a técnica de early stopping para prevenir overfitting.

Conclusões

As 3 abordagens têm resultados muito semelhantes. Apesar de terem bons resultados nos dados de teste. No dataset de avaliação (30 amostras) os resultados não se mantêm e têm uma performance fraca. Explorou-se também o tamanho do vocabulário das 3 estratégias com o argumento MAX_TOKENS. Dos valores experimentados 500,1000,5000,10000,20000 5000 têve os melhores resultados no dataset de avaliação. Para 10000,20000 os modelos facilmente davam overfit e classificavam tudo com a mesma classe. Para 500,1000 deu piores resultados que 5000, parece ser um vocabulário demasiado curto para representar o problema, levando a demasiada perda de informação e portanto, underfit.

4.2 RNN Simples, LSTM, GRU

RNN Simples

No notebook de RNN, utilizamos a camada SimpleRNN do Keras para implementar uma rede neural recorrente simples. A arquitetura consiste em uma única camada SimpleRNN com 64 unidades, ativação tanh e dropout de 20% para regularização. A saída da camada recorrente é ligada a uma camada Dense com uma única unidade e ativação sigmoid para realizar a classificação binária. O otimizador utilizado é o SGD com uma taxa de aprendizagem de 0.08, e a função de perda é a entropia cruzada binária. O modelo é treinado por 100 épocas, com suporte opcional para validação.

LSTM

No notebook de LSTM, utilizamos a camada LSTM do Keras, que é uma variante mais avançada de RNN, projetada para lidar com dependências de longo prazo. A arquitetura inclui uma camada LSTM com 64 unidades, ativação padrão, dropout de 20% e recurrent dropout de 20% para regularização adicional. Assim como no RNN, a saída da camada recorrente é conectada a uma camada Dense com ativação sigmoid. O otimizador utilizado é o Adam com uma taxa de aprendizagem de 0.001, e o treinamento inclui o uso de EarlyStopping para interromper o treinamento caso a perda de validação não melhore após 5 épocas consecutivas.

GRU

No notebook de GRU, utilizamos a camada GRU do Keras, que é uma alternativa mais eficiente ao LSTM, com menos parâmetros e desempenho comparável. A arquitetura é semelhante à do LSTM, com uma camada GRU de 64 unidades, dropout de 20% e recurrent dropout de 20%. A saída da camada GRU é conectada a uma camada Dense com ativação sigmoid. O otimizador utilizado é o Adam, e o treinamento também inclui EarlyStopping para evitar overfitting, interrompendo o treinamento quando a perda de validação não melhora.

Comparação das Estratégias

As três abordagens compartilham uma estrutura básica semelhante, mas diferem na camada recorrente utilizada. O RNN simples é mais básico e pode sofrer com problemas de gradientes desaparecendo, enquanto o LSTM e o GRU são projetados para lidar melhor com dependências de longo prazo. O LSTM é mais robusto, mas tem mais parâmetros, enquanto o GRU é mais leve e eficiente, sendo uma boa alternativa as LSTM.

4.3 Zero Shot

Para experimentar LLMs começamos pela abordagem mais simples, zero shot. Para tal, começamos por realizar uma experiência comparativa entre os fornecedores de LLMs mais relevantes: OpenAI, Anthropic e Deepseek. Os resultados de cada uma foram obtidos manualmente no website ou app de cada e a experiência está documentada no notebook inference_zeroshot_manual_experiment.ipynb.

Resumidamente, os modelos normais da OpenAI e Deepseek não tiveram bons resultados e foram utilizados os modelos de reason. Quanto à Anthropic, o modelo Claude Sonnet teve melhores resultados que o modelo mais complexo: Opus. Foi utilizado o dataset inicial de avaliação de 30 samples. O OpenAI reason teve 67% de acurácia, Deepseek R1 90% e Claude Sonnet 100%.

Para a submissão 2, utilizou-se o melhor modelo desse teste, Claude Sonnet, "submissao2-grupo008.ipynb". Na submissão interagimos com o Claude através da API.

Para a submissão 3, utilizamos outra vez zero shot no notebook “submissao3-grupo008-s1.ipynb”. No notebook, para além da submissão tem um estudo comparativo dos modelos mais recentes da Google que saíram na altura. Comparou-se com o modelo mais simples da google, o mais equilibrado e o mais complexo. Tiveram os respectivos resultados de acurácia: 56%, 65% e 59%. Enquanto o Claude têve 87%. Para este teste já foi utilizado um dataset maior “dataset2_disclosed_complete.csv”.

4.4 Few Shot

A estratégia de Few Shot foi implementada no notebook "submissao3-grupo008-s2.ipynb".

Como base de dados vectorial foi utilizada a chromaDB através de uma wrapper classe disponibilizada pela langchain, "langchain_chroma". Esta facilita a interacção com a base de dados. Como embedding foi utilizado um embedding apropriado para pesquisa semântica disponibilizado pelo HuggingFace : https://huggingface.co/sentence-transformers/all-mpnet-base-v2 . A base de dados foi preenchida com múltiplos datasets obtidos na tarefa 1.

Experimentamos 4 combinações possíveis de few shot. Para cada estratégia experimentamos os valores 1 e 3 de **K** (exemplos).

- Estratégia 1: consiste em: para cada amostra a classificar fornecer os **K** exemplos semanticamente mais parecidos.

– Estratégia 2 : semelhante, mas para cada amostra foram fornecidos os **K** exemplos mais parecidos da classe Humana e os **K** mais parecidos da classe AI.

Os resultados de acurácia podemos ver na tabela 1 e no notebook.

	K = 1	**k** = 3
E1	86%	86%
E2	90%	87%

Table 1. Acurácia de estratégias de Few Shot

Conclusões

Os resultados são semelhantes. A estratégia 2 resultou melhor em ambos valores de **K** e como têve melhor em **K** = 1, usou-se esses parâmetros para a submissão 3.

Decidimos experimentar esta estratégia para evitar não guiar o modelo na direção errada, havendo exemplos semelhantes de ambas classes. O que demonstrou melhores resultados. Com o aumento de exemplos aumenta o risco de overfit, o que pode explicar a degradação dos modelos para **K**=3.

Outra observação: para estratégia 2 tivemos de utilizar um batch size de perguntas inferior. 5 perguntas em cada prompt em vez de 10. Uma vez que o nosso tier de claude apenas permite um contexto de 20 mil tokens e era ultrapassado. Este limite de contexto é calculado através da soma de tokens de input com output.

5 Tarefa 4: Avaliação dos Modelos

5.1 Resultados Locais

Numpy

Modelo	Resultados Obtidos
DNN - Numpy	60%
RNN - Numpy	53%

Table 2. Tabela Submissão 1

DNN Keras

Estrategia	Resultados Obtidos
Unigram	57%
Bigram	57%
Tf-idf	53%

Table 3. Tabela Submissão 1

RNN Keras

Modelo	Resultados Obtidos
RNN Simples	50%
LSTM	53%
GRU	53%

Table 4. Tabela Submissão 1

5.2 Submissões

Submissão 1

Modelo	Resultados Obtidos
DNN - Numpy	62%
RNN - Numpy	55%

Table 5. Tabela Submissão 1

Submissão 2

Modelo	Resultados Obtidos
DNN - Tabular	64%
Zero Shot - Claude Sonnet	84%

Table 6. Tabela Submissão 2

Submissão 3

Modelo	Resultados Obtidos
Zero Shot	85%
Few Shot	86%

Table 7. Tabela Submissão 3

6 Conclusões

Conclui-se que, para além de proporcionar uma visão aprofundada sobre as diversas técnicas de implementação (com Numpy e com TensorFlow/Keras), o projeto evidenciou importantes insights sobre os resultados obtidos. De forma geral, os modelos implementados apresentaram desempenhos que variaram de forma consistente entre as diferentes abordagens e submissões.

Relativamente aos resultados, as abordagens com implementação própria (utilizando Numpy) obtiveram percentagens moderadas, com o DNN a atingir cerca de 60% e o RNN cerca de 53% em condições locais, e ligeiramente melhores nas submissões (62% para DNN e 55% para RNN). Por outro lado, os modelos desenvolvidos com TensorFlow/Keras revelaram desempenhos semelhantes entre as estratégias de representação de texto – Unigram, Bigram e tf-idf – oscilando entre 53% e 57% para o DNN Tabular, e cerca de 50% a 53% para as variantes de RNN (Simple RNN, LSTM e GRU).

A estratégia Zero Shot mostrou resultados significativamente superiores, nomeadamente com o modelo Claude Sonnet que alcançou 84% na submissão 2 e 85% na submissão 3, evidenciando o potencial dos modelos de Large Language Models para tarefas deste género. A abordagem Few Shot, que envolveu a seleção cuidadosa de exemplos semanticamente semelhantes, demonstrou ligeira melhoria ao atingir 86% na submissão 3. Estes resultados sugerem que, embora as implementações próprias permitam um controlo mais detalhado do processo de treino, os modelos pré-treinados e as abordagens baseadas em exemplificação (Zero e Few Shot) são mais eficazes para a tarefa de deteção de textos gerados por IA.

Assim, os resultados obtidos destacam não só a relevância de uma adequada preparação e normalização dos datasets (com especial atenção ao tamanho do vocabulário, onde um valor em torno de 5000 tokens se mostrou mais adequado), mas também a importância de escolher a abordagem correta para a tarefa, equilibrando complexidade e eficiência. Estes achados fornecem uma base sólida para futuras investigações e ajustes que possam melhorar ainda mais a performance dos modelos no desafio de diferenciar textos de IA e humanos.