Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática

Redes Neuronais Artificiais

Classificação Emotiva de Texto

Grupo de trabalho:
André Almeida Gonçalves, A75625
Miguel Dias Miranda, A74726
Rogério Gomes Lopes Moreira, A74634
Tiago Filipe Oliveira Sá, A71835

Braga, 8 de Abril de 2018

Conteúdo

Resumo

Com o massivo volume de publicações que diariamente se realizam nas diversas redes sociais, torna-se relevante realizar uma análise de alto nível do seu conteúdo. Aliando esta fonte de dados à atual capacidade de processamento que se consegue obter em métodos de *Machine Learning* é assim viável a exploração destes dados com base nestas técnicas, como forma de análise dos mesmos.

Neste sentido, o presente projeto prático pretende explorar a utilização de *Redes Neuronais Artificiais* (*RNAs*) num problema de classificação emocional de texto, através de publicações na rede social *Twitter*.

Ao longo deste relatório é apresentado todo o processo desenvolvido como forma de analisar o conjunto de dados em bruto, o pré processamento do mesmo, o desenvolvimento da arquitetura da RNA e o processo de treino e teste da mesma.

Por fim, são apresentados os resultados obtidos, focando os padrões de treino que obtêm melhores resultados de classificação.

1. Introdução

Redes neuronais artificias apresentam-se como um sistema conexionista, fortemente inspirado nas características do sistema nervoso central do ser humano. Apesar das RNAs serem um modelo simplificado, a sua arquitetura extremamente interligada de unidades de processamento permite que estas sejam capazes de generalizar e adquirir conhecimento, através de um processo de aprendizagem.

Tirando partidos das suas características, vários algoritmos de *Machine Learning* recaem sobre estas estruturas de aprendizagem, devido às suas capacidades de classificação e previsão em qualquer um dos paradigmas de aprendizagem.

Neste projeto procura-se assim recorrer às capacidades de uma RNA, criada e modelada ao contexto do problema, para realizar a classificação do nível emocional de publicações da rede social *Twitter*.

No sentido de introduzir e uniformizar os conceitos em torno das RNAs, o presente relatório fornece inicialmente um descrição teórica dos principais temas associados com a modelação de uma rede neuronal artificial. Posteriormente, são descritas todas as etapas desenvolvidas ao longo do projeto e as decisões tomadas no decorrer deste.

Como estrutura do presente documento: o capítulo 2 apresenta uma descrição breve daquilo que atualmente se entende por uma rede neuronal artificial, focando com relevo as características das suas unidades de processamento (neurónios), as arquiteturas existentes e o algoritmo de aprendizagem de *Back-Propagation*; o capítulo 3 apresenta uma descrição da fase de pré processamento dos dados, focando a eliminação de ruído, seleção da informação útil e criação de novos atributos.

De seguida, numa vertente mais prática, os capítulos 4 e 5 apresentam, respetivamente, todas as considerações no processo de desenvolvimento e análise de resultados, ao longo dos diferentes cenários de teste realizados.

Por fim, o capítulo 7 apresenta as conclusões relativamente ao projeto e o trabalho futuro.

2. Redes Neuronais Artificiais

2.1 Descrição geral

Apresentando-se como uma abordagem de *machine learning*, os algoritmos de redes neuronais artificias (RNAs) baseiam-se num sistema conexionista, inspirado no funcionamento da estrutura cerebral humana.

Estas técnicas recorrem a uma topologia em rede para receber informações que, após processadas, influenciam o estado interno de cada neurónio da rede. O conhecimento é assim adquirido através do ambiente por um processo iterativo de aprendizagem. Este conhecimento é armazenado nas conexões da rede, através de uma adaptação dos pesos dos neurónios -ou nodos- ao longo de um processo de aprendizagem.

Independente do contexto, uma rede neuronal apresenta uma arquitetura genérica (Figura 2.2), normalmente composta pelas seguintes camadas [7, 15]:

- **Camada de entrada**, na qual é recebida a informação a analisar. Apesar dos dados serem normalmente obtidos através de um dataset pré preparado, esta camada pode ser vista como a comunicação da rede com o ambiente, como forma de percecionar e obter dados.

  É assim uma camada sem antecessor, servindo como uma interface para alimenta a rede com nova informação.

- **Camadas intermédias**, que desempenham a tarefa de processamento dos dados de entrada, com base num determinado paradigma de aprendizagem.

- **Camada de saída**, onde se devolvem os resultados do processo de aprendizagem da rede.

  O número de nodos de saída está diretamente relacionado com o número de parâmetros que são previsto pela rede num determinado contexto.

Alguns dos benefícios das redes neuronais, comparativamente a outros processos de *Machine Learning*, estão associados ao seu puder de processamento. Esta capacidade deve-se essencialmente à sua estrutura, que permite executar um ambiente extremamente paralelo. Além deste fator, apresentam ainda outras características de relevo, nomeadamente [15] [7] [9]:

- **Generalização**: No final do processo de aprendizagem, a rede é capaz de descrever um universo completo, através de uma aprendizagem prévia com algumas das suas partes. Além disto, devido à sua robustez, as RNAs são ainda capazes de processar ruído ou informação incompleta, conseguindo generalizar a informação que analisam na fase de treino para novos casos futuros.

- **Não linearidade**: Além de problemas simples, estes algoritmos apresentam-se capazes de analisar características não lineares, através da interconexão de diferentes neurónios e uso de diferentes funções de ativação;
- **Relação entre input-output**: Quando aplicadas num contexto de aprendizagem supervisionada, com um conjunto de input de treino e um dataset de output esperado, as redes neuronais são capazes de gerar resultados satisfatórios de classificação;
- **Adaptação**: Uma das principais características das RNAs reside na sua capacidade de adaptação, útil em ambientes com dados desconhecidos ou em falta.

  Os pesos atribuídos a cada neurónio são capazes de ser adaptar, conforme o domínio sobre a qual a rede se encontra a ser treinada. Alem disto, a sua topologia pode também alterar-se de acordo com as mudanças do ambiente, criando ou reduzindo o número de ligações entre nodos.

  Desta forma, no final do processo de aprendizagem, uma RNA é capaz de generalizar novos casos à sua capacidade de raciocínio.
- **Tolerância a falhas**: Devido à sua topologia, as RNAs permitem um elevada tolerância a falhas. A falha de um neurónio pode ser atenuada pela marcação do peso dos seus axónios como zero, simulando assim o desaparecimento das ligações para aquele neurónio.

  A falha de um neurónio leva ao reajustamento dos restantes, levando a uma degradação gradual e suave da rede, em caso de desativação de algumas conexões entre neurónios.

2.2 Neurónio Artificial

Cada camada de uma RNA é composta por um conjunto de neurónios, identificados pelos nodos da topologia da rede. Um neurónio pode ser visto como uma unidade de processamento da rede, recebendo ligações de outro neurónios e unindo-se a outros nodos da rede semelhantes a si.

As ligações entre neurónios são designadas por axónios, tal como na estrutura biológica do cérebro e, têm a si associadas um peso. Este peso pode, numa interpretação simples, ser visto como a importância da informação que aquele axónio envia a um neurónio. O peso de cada axónio é regulado ao longo do processo de aprendizagem, levando a uma generalização da topologia da rede aos dados, até se obter um resultado coerente com os esperados no contexto. A transmissão de informação entre os nodos é conseguida através de sinapses, novamente seguindo a nomenclatura biológica.

Ao longo do processo de evolução da rede, os neurónios vão atualizando o seu estado, sendo este normalmente representado por um valor numérico entre *[0, 1]*. De uma forma geral, o estado interno de um neurónio é caracterizado por:

- Um **conjunto de ligações**, cada uma com o seu próprio peso. Ao longo da evolução da rede, o sinal propagado por um determinado axónio é multiplicado pelo peso do mesmo, resultando assim na excitação -ou inibição- do nodo;

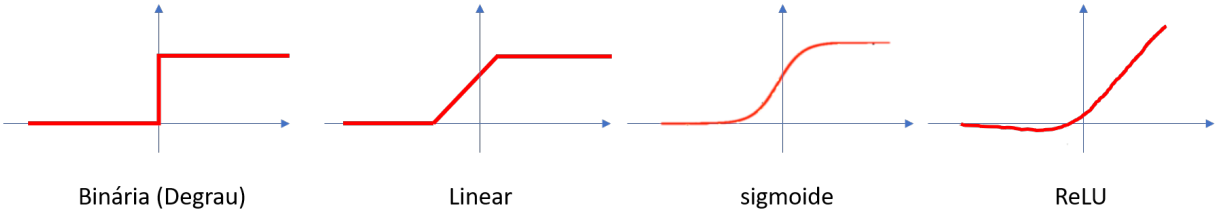


Figura 2.1: Exemplos de diferentes tipos de funções utilizadas como função de ativação dos neurónios.

- Um **Integrador**: Apresenta-se como uma função de redução, que simplifica a influencia das N conexões de entrada de um neurónio para um único valor.

  Geralmente, o integrador realiza apenas o somatório $(\sum)$ dos valores recebidos pelas N ligações de entrada, multiplicados pelo peso da respetiva ligação;

- A **Função de Ativação**: Determina o valor de saída do neurónio, de forma a restringir o valor de saída do neurónio para um determinado valor finito. Este valor é normalmente calculado através do mapeamento do valor do integrador para uma determinada função não linear. De referir que a escolha da função de ativação tem uma influência direta e bastante significativa na capacidade de aprendizagem da rede.

  Algumas das principais funções são a função binária (degrau), linear ou funções do tipo *sigmoide*. [11] Em pesquisas recentes, foi ainda apresentada a função designada por *ReLU*. (Figura 2.1) Esta função, explorada pelos investigadores da *Google Brain Team*, apresenta algumas vantagens face a implementações com a função *sigmoide*.

2.3 Arquiteturas da Rede

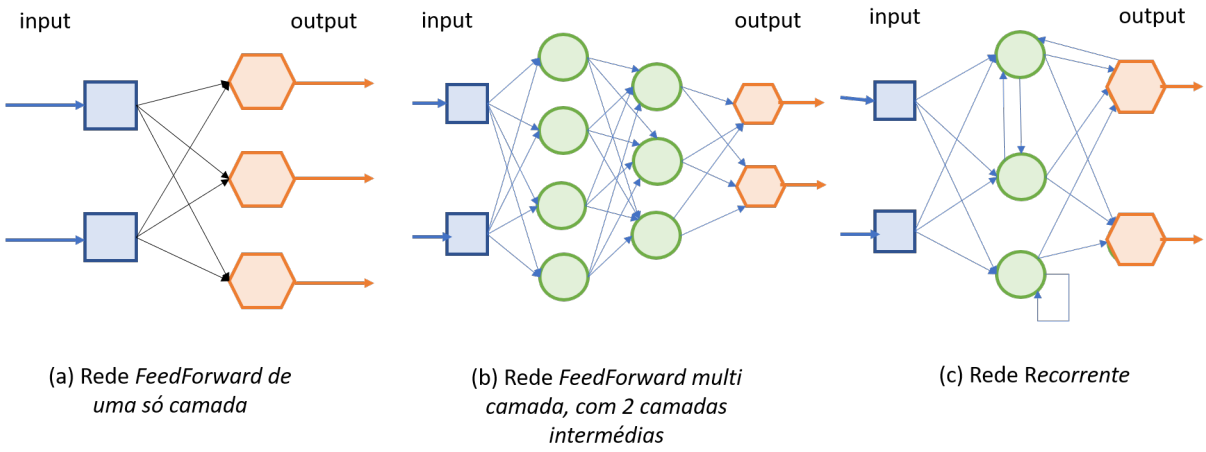
A arquitetura de uma rede neuronal artificial especifica a quantidade de neurónios (nodos) que a mesma possui e, de que formas estes se relacionam e interligam entre si. A arquitetura pode ser visualizada na forma de grafos, no qual as ligações entre os nodos são orientadas.

De uma forma geral existem três tipos distintos de topologia, sendo que quanto mais complexa a arquitetura de uma RNA, maior a sua capacidade para lidar com problemas mais complexos.

- ***Single Layer Feedforward***: As ligações da rede seguem apenas uma direção. Nesta arquitetura não existem camadas intermédias, existindo apenas as camadas de *input* e *output*.

  A partir de um nodo de entrada existem apenas ligações divergentes (1 para N) e os nodos de saída apresentam ligações convergentes (N para 1). Esta topologia representa as arquiteturas inicias das RNAs. (Figura 2.2-a)

- ***Multi-Layer Feedforward***: Seguindo os mesmos conceitos da arquitetura anterior, as RNAs *multi-layer* permitem a definição de uma ou mais camadas



(a) Rede *FeedForward de uma só camada*

(b) Rede *FeedForward multi camada, com 2 camadas intermédias*

(c) Rede *Recorrente*

Figura 2.2: Exemplos de algumas topologias de arquiteturas de RNAs

intermédias. Embora esta relação não seja linear, a adição de camadas intermédias a uma rede permite aumentar a sua capacidade de aprendizagem, podendo assim lidar com funções e relações entre os dados mais complexas.

Estas redes são assim úteis em contextos onde existe um grande número de dados de input, sobre os quais se espera observar alguma relação ou padrões gerais. Contudo, o aumento do número de camadas de uma rede neuronal, leva a um aumento exponencial do seu tempo de aprendizagem. (Figura 2.2-b)

Neste tipo de arquiteturas, o algoritmo de *Back-Propagation* é um dos principais métodos de aprendizagem. Este método calcula a contribuição do erro para cada um dos nodos da rede, ajustando o peso de cada nodo com base no erro calculado (Secção 2.4).

- **Redes recorrentes**: Numa sucessão à arquitetura anterior, as RNAs baseadas em redes recorrentes permitam a existência de conexões de um nodo para si próprio. A ligação de um neurónio para si próprio permite a introdução de conceitos de "memória" no próprio nodo.

  Além deste fator, um neurónio pode ainda ligar-se a outros nodos de camadas anteriores ou da própria camada, permitindo assim a existência de ciclos na estrutura. (Figura 2.2-c)

2.4 *Back-Propagation*

Associado ao método de aprendizagem do Gradiente Descendente, o algoritmo de *Back-Propagation* é um dos algoritmos mais utilizados no paradigma de aprendizagem supervisionada de redes neuronais artificiais.

Regras de aprendizagem do tipo Gradiente Descendente baseiam-se na diminuição de um erro, sendo este estimado entre um valor pretendido e um valor obtido pela previsão da rede. [9] Para cada estado do processo de evolução da rede, o

objetivo prende-se com a minimização de uma função de custo, definida em termos do valor de erro.

Este processo de avaliação é descrito pela Equação 2.1, onde η é a taxa de aprendizagem e $\Delta\epsilon$ a diferenças entre o valor esperado e o valor obtido para uma dada entrada (gradiente da função de custo).

$$\Delta w = \eta \Delta \epsilon \tag{2.1}$$

No caso especifico do algoritmo de *Back-Propagation*, a sua implementação segue os seguintes passos:

- Antes de se iniciar o treino, são atribuídos valores aleatórios aos axónios da arquitetura, marcando assim o peso inicial de cada ligação;
- Iniciado o treino, os valores (vetores entrada) fornecidos aos nodos de input vão-se propagando para a frente, até chegarem aos nodos de saída. Chegando à camada final, é calculado o erro entre o valor obtido pela rede e o valor esperado;
- O referido erro é propagado para as camadas anteriores, desde os nodos de saída até aos aos nodos da camada de *input*. Esta retro propagação leva ao ajustamento dos pesos de cada nodo segundo a regra de *Widrow-Hoff* Equação 2.1;
- Este processo é repetido para cada iteração. A fase de treino termina quando forem cumpridos os critérios de paragem definidos.

3. Análise Exploratória Dados

3.1 *Dataset* em estudo

Como referido, neste projeto procura-se utilizar as capacidades de generalização de uma *Rede Neuronal Artificial* para a identificação do nível emocional de uma determinada publicação, com base nas palavras utilizadas na sua composição.

Como suporte de dados para o processo de treino e teste da rede, foi utilizado um *dataset* com informações de publicações da rede social *Twitter*, recolhidas num período entre 1 de janeiro de 2016 e 31 de Julho de 2017. Será sobre este conjunto de dados que recai a fase de pré processamento de dados, abordada ao longo deste capítulo.

Como principais atributos, o referido *dataset* apresenta inicialmente os campos:

- ***UserLocation*** – (*String*) Representação textual da localização do utilizador que realizou a publicação. A localização toma a forma de uma string, geralmente com o nome da cidade e o país. Pode também apresentar apenas as coordenadas de longitude e latitude ou valores nulos;
- ***IdPost*** – (*Inteiro*) Identificador único de cada uma das publicações;
- ***TextPost*** – (*String*) Conteúdo textual da publicação. Apresenta a informação relevante que se procura analisar, após pré tratamento e adaptação, com base em RNAs;
- ***Hashtags*** – (*String*) Conjunto de *hashtags* utilizadas junto da publicação.

  Num sentido de uniformização, no contexto do *dataset* identifica-se por "*hashtags*"o conjunto de palavras iniciadas pelo carácter # e que procuram identificar as palavras chave ou tópicos da publicação.

  Este campo pode encontrar-se vazio (valores nulos) ou conter um conjunto de um ou mais *hashtags*;
- ***Date*** – (*Data*) Registo da data em que a publicação foi colocada online, segundo o formato "Dia/Mês/Ano Hora:Minutos".

  Cerca de 279 instâncias contêm este atributo como nulo;
- ***TweetLat*** – (*Inteiro*) Valor da latitude na qual a publicação foi colocada online;
- ***TweetLon*** – (*Inteiro*) Valor da longitude na qual a publicação foi colocada online;
- ***PhotoLat*** e ***PhotoLon*** – (*Inteiros*) Valores da latitude e longitude associados à fotografia colocada junto com a publicação.

No total, o *dataset* é composto por um conjunto de 990473 instâncias. Nas seguintes secções serão destacadas as alterações realizadas ao dataset a nível de limpeza e criação de novos atributos.

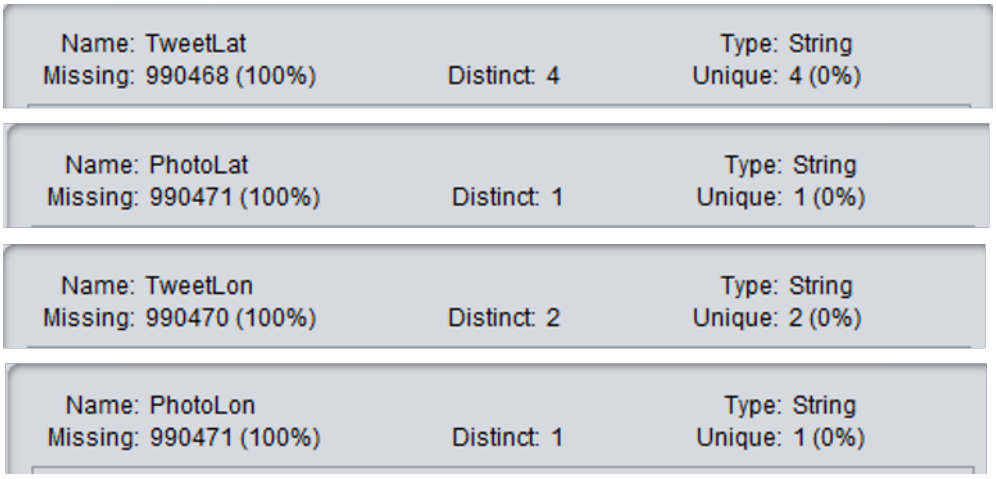


Figura 3.1: Análise do número de valores nulos e distintos dos atributos do dataset, recorrendo à ferramenta *Weka*

3.2 Análise dos atributos

No sentido de identificar o nível emocional de uma publicação, foi inicialmente realizado um estudo aos dados fornecidos, no sentido de interpretar os seus valores e determinar qual a relevância de cada um dos atributos neste contexto.

Nesta secção, procura-se assim justificar a escolha dos atributos do dataset a as alterações que se realizaram ao mesmo. Tal como já referido, o conjunto de dados recolhidos resulta da recolha de um conjunto de informações associadas a diversas publicações na rede social *Twitter*.

- *UserLocation*
- *IdPost*
- *TextPost*
- *Hashtags*
- *Date*
- *TweetLat / TweetLon*
- *PhotoLat / PhotoLon*

O primeiro processo de filtragem de qualidade destes parâmetros baseia-se na presença de valores nulos, ou seja, no número de registo em falta para um determinado atributo. Nesse sentido, os atributos *TweetLat, TweetLon, PhotoLat* e *PhotoLon* foram automaticamente removidos, por apresentarem uma taxa de valores nulos de 100%. Sendo valores inteiros, esta informação foi conseguida através de uma análise segundo a ferramenta *Weka* (Figura 3.1).

Sendo que por si só, a análise de linguagem natural já é um processo bastante dependente do contexto da frase e da forma como se organizam as palavras, uma das dificuldades associadas a este tipo de processamento recai na ambiguidade das palavras.

Neste sentido, o uso de palavras soltas pode estar associado a diversos contextos emocionais ou até com formas de estilo como ironia. Por este motivo, foi tomada a decisão de ignorar quais *hashtag* existente no dataset, dado que no contexto não apresentam um conhecimento sólido para o processo de aprendizagem da rede.

Assim, foi removida integralmente a coluna relativa ao atributo *Hashtags*, que continha a listagem de eventuais *hashtags* associados à publicação.

O atributo *IdPost* foi também removido nesta fase inicial de limpeza, dado que por ser único para cada publicação, não apresenta qualquer informação capaz de ser apreendida e generalizada pelo processo de aprendizagem da rede neuronal artificial.

Os parâmetros *UserLocation* e *Date* foram mantidos por apresentarem alguma integridade nos seus valores. Numa fase posterior ao treino e teste de desempenho da RNA, estes registos podem eventualmente ser utilizados para gerar dados e estatísticas com base na localização do utilizador e o nível emocional da publicação. Contudo, não serão utilizados nem adaptados para o processo de treino da rede.

Em suma, de todos os atributos do conjunto de dados, aquele que será exclusivamente adaptado para ser utilizado no processo de evolução da rede será o conteúdo textual da publicação, *TextPost*, dado que é sobre este texto que se procura classificar o nível emocional da frase.

3.3 Adaptação do *Dataset*

Como referido na secção anterior, a análise da semântica de um determinado texto está intrinsecamente ligado com o contexto em que a frase é composta e de que forma as palavras estão organizadas na mesma.

Para um processo de análise de texto com base em algoritmos de *Machine Learning*, registos com palavras soltas como os *hashtags* ou hiperligações para páginas online, são alguns dos exemplos de informações que não apresentam nenhum material útil para o processo de extração de conhecimento.

Relativamente a estes dois tipos de "palavras":

- Analisar um *hashtag* pelo valor da palavra em si, seria assumir de forma implícita um contexto para essa palavra e utilizar o peso da palavra nesse contexto em qualquer publicação onde esse *hashtah* fosse utilizado.

  A titulo de exemplo, uma publicação com um contexto negativo e associado a tristeza pode apresentar um hashtag com a palavra *"happy"*, meramente aplicada num contexto de ironia e como tal, oposta ao verdadeiro sentido emocional da publicação.

  Neste sentido, atribuir valor às palavras individuais dos *hashtags* seria afetar o desempenho de classificação e enviesar a capacidade de análise de qualquer processo de aprendizagem que fosse utilizado.

  Por este motivo, foram também removidos todos os *hashtags* que surgissem no conteúdo textual da publicação. Para isso foi simplesmente utilizada a ferramenta *NotePad++* e utilizada a expressão regular *#([a-zA-Z0-9] | [À-ÿ])** para apanhar todos os *hashtags*.

- Apesar de ser possível, a partir do endereço de uma hiperligação, utilizar uma API para extrair todas as palavras-chave associadas ao conteúdo dessa página, a obtenção desta lista de palavras-chave levaria ao mesmo problema que os *hashtags*, descrito no tópico anterior.

Como consequência, todas as referências a hiperligações foram também removidas do conteúdo do texto da publicação.

Contudo, a representação de *urls* estava bastante fragmentada e incoerente, tendo mostrado alguma dificuldade numa completa limpeza dos mesmos. A solução passou pelo uso de expressões regulares mais complexas, criadas para o contexto e, uso dos métodos do *package Tweet PreProcessor* existentes em *Python*.

Focando no conteúdo da publicação que é efetivamente útil para análise, é relevante frisar que no *dataset* fornecido não existe qualquer relação entre as frases das publicações e o valor emocional associado às mesmas.

Dado que este projeto recai sobre o uso de RNAs segundo um paradigma de aprendizagem supervisionado, é necessário fornecer à rede um conjunto de dados de treino junto com o respetivo conjunto de *labels*, que identifique qual o resultado de *output* esperado para um cada *input* de treino.

Para gerar estas informações é necessário recorrer àquilo que pode ser visto como um dicionário, que mapeia o valor de uma palavra com o nível de sentimento a si associado. Analogamente, a forma mais simples de medir o peso emocional de uma frase, recai também no uso destas estruturas, somando o peso das palavras que compõe uma frase.

Para gerar esta métrica para cada uma das instâncias do dataset, foram tidas em conta as seguintes considerações:

- O dataset é composto por publicações em diversas linguagens, nomeadamente inglês, francês e italiano;
- Utilizar as instâncias apenas com publicações em inglês seria reduzir significativamente o volume do *dataset* e, com isso, perder informação relevante para o processo de aprendizagem da rede;
- Não sendo viável remover todas as publicações que não se encontrem em inglês e dado que a maioria dos dicionários <Palavra, Valor-Emocional> existentes e disponíveis recaem na terminologia inglesa, uma das soluções passa pela tradução de todos as instâncias para a inglês.

  Contudo, recorrendo à API de tradução disponibilizada pela plataforma *Google Translate* [1], o processo de tradução é implementado sobre pedidos http. Cada pedido envolve o encapsulamento (*marshling e unmarsheling*) de pacotes *JSON* com o conteúdo e resposta da tradução, pelo que o tempo de cada tradução andava acima de 1 segundo por frase/publicação.

  Dado o tempo disponível para este projeto e a dimensão do dataset (+ de 900 mil instâncias), a opção de traduzir cada publicação não se apresenta viável;

A solução encontrada passou pelo uso de uma nova API de mapeamento <Palavra, Valor-Emocional> que fosse capaz de lidar com várias linguagens, evitando assim que fosse necessária uma tradução prévia do conjunto de dados.

É com base neste nível de sentimento, quantificado num valor numérico, que se mapeia o peso emocional das frases de uma publicação numa das seguintes classes:

- *Negativo*
- *Neutro*
- *Positivo*

Apesar do enunciado do projeto especificar que as classes de classificação da RNA deveriam ser apenas "*Negativo*" ou "*Positivo*", foi tomada a decisão que a classificação seria mais coerente e rigorosa de existisse também o nível *Neutro*.

Desta forma, a *tag* "$Neutro$" procura representar as publicações que não apresentam qualquer nível sentimental, seja porque o seu conteúdo é ambíguo de interpretar ou porque são frases simples, que procuram apenas descrever factos ou eventos.

Pelo facto de uma frase não ser simplesmente positiva ou negativa, justifica-se assim a decisão de existirem 3 classes de previsão.

3.4 Criação de Atributos *Target/Output*

Para a criação deste novo atributo, que representa o output sobre o qual a rede deve treinar e ser capaz de realizar previsão, foram exploradas algumas soluções de análise de texto já implementadas na linguagem *Python*.

Relativamente à escolha de uma API capaz de realizar uma análise em linguagem natural e deteção de sentimentos, foram exploradas as seguintes soluções:

- ***Natural Language Toolkit - NLTK*** [4]: A plataforma *NLTK* apresenta-se como uma das ferramentas *Open-source* mais utilizadas para desenvolver algoritmos ou métodos que necessitem de analisar linguagem humana num contexto semântico.

  No contexto de análise de sentimentos em frases, os métodos desta ferramenta baseiam-se em técnicas de Classificação hierárquica para determinar o nível de polaridade e subjetividade de uma frase. Por polaridade entende-se o nível de sentimento de uma frase (positivo ou negativo) e por subjetividade entende-se o nível de confiança da classificação, conforme o contexto e conteúdo da frase analisada.

  A *API* desenvolvida para *Python* [5] apresenta como pontos positivos e facto de devolver a Polaridade dividida em dois valores decimais, para identificar a percentagem de sentimento negativo ou positivo na frase.

  Contudo, o seu foco encontra-se ainda muito voltado para a língua inglesa, tornando-a assim inadequada para o dataset em análise, pelos motivos supracitados na listagem anterior.

- ***Microsoft Azure: Text Analytics AP*** [3]: Disponibilizados na plataforma *Azure*, a Microsoft disponibiliza um conjunto de soluções que designa como "Serviços cognitivos"pela sua envolvência com a temática da inteligência artificial.

  Estes serviços oferecem APIs para processamento de imagem, voz, conhecimento e linguagem. É nesta ultima área que surge a ferramenta *Text Analytics* [2].

  Esta solução é capaz de, para uma determinada entrada de texto, automaticamente determinar a sua linguagem e devolver o seu nível de sentimento, junto

com uma lista de palavras chave/*target* que permitem atribuir um determinado nível de sentimento à frase.

Apesar da ferramenta funcionar com um grande número de linguagens e apresentar uma elevada capacidade de análise para frases com palavras relacionadas entre si, a chave de utilização deste serviço é limitada a 30 dias e a 5000 pedidos por dia.

Novamente, a exploração integral do dataset com esta API seria impossível, devido ao seu volume de dados e ao domínio temporal deste projeto.

- ***TextBlob*** [6]: Procurando contornar as limitações encontradas nas duas ferramentas anteriores, a biblioteca *TextBlob* apresenta-se como a solução encontrada para pré processar o dataset limpo, como forma de calcular o nível de sentimento de cada frase.

  Esta plataforma é construida sobre a ferramenta *NLTK* referida no primeiro ponto, embora acrescente sobre os seus métodos a capacidade de deteção de sentimentos em texto em várias linguagens nativas.

  Por este motivo e, sendo totalmente *open-source* e sem limite de pedidos, esta solução foi a utilizada para calcular os valores de sentimento de cada frase, sobre os quais a rede deve treinar e aprender a classificar/prever futuras frases de publicações.

O uso da ferramenta *TextBlob* envolve assim a tradução de publicações num idioma estrangeiro para inglês nativo, realizando uma tradução estocástica [13].

Considerando assim os resultados devolvidos pelos métodos da API *TextBlob* para *Python*, foram então acrescentadas duas colunas ao dataset:

- A coluna **Polaridade**, que indica o nível de sentimento de um texto.

  Toma valores decimais entre [-1, 1], sendo que frases com valores de polaridade próximos de -1 apresentam um conteúdo negativo e frases com valores de polaridade próximos de 1, associam-se a sentimentos positivos.

- A coluna **Subjetividade**, onde se quantifica a confiança do nível calculado para o texto, conforme o seu contexto.

  Toma valores decimais entre [0,1], sendo que valores próximos de 0 estão associados a textos bastante objetivos -nível de sentimento concreto para o contexto- e valores próximos de 1, recaem sobre textos subjetivos,onde a medição do nível de sentimento não é concreta para o contexto.

Relembrando que se procura classificar o nível emocional de uma frase apenas em três classes concretas (Secção 3.3), é necessário realizar uma descriterização dos valores tomados pelo atributo Polaridade. Uma vez que a API fornece duas métricas relevantes, foi tomada a decisão de conjugar estas duas informações num único atributo, que mapeia os valores dentes os parâmetros Polaridade e Subjetividade para uma das classes objetivo: Negativo, Neutro ou Positivo.

Como forma de não criar tendências no processo de aprendizagem, nem classificar como positiva ou negativa frases com conteúdo/contexto ambíguo, foi tomada

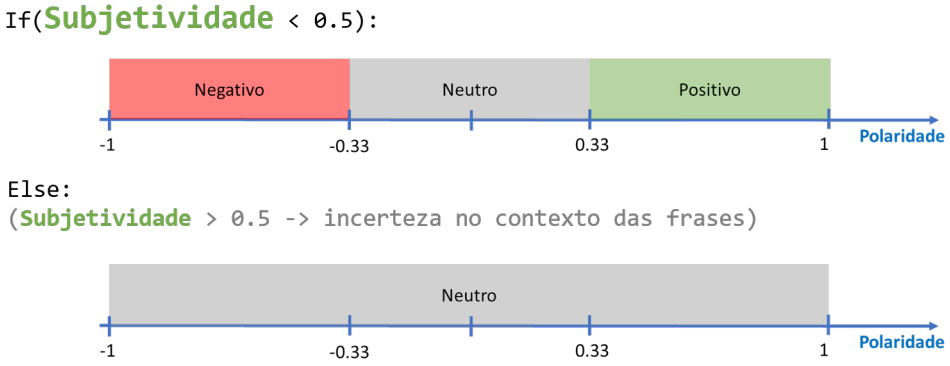


Figura 3.2: Mapeamento dos valores de polaridade e subjetividade para as classes "Negativo", "Neutro"e "Positivo".

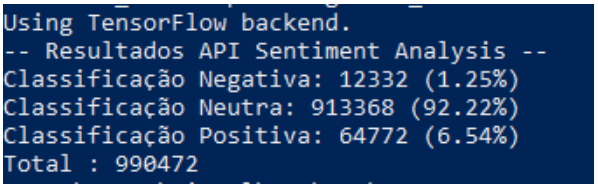


Figura 3.3: Percentagens de publicações classificadas num determinado nível de sentimento: Negativo, Neutro ou Positivo.

a decisão que todas as instâncias com subjetividade superior a 0.5 representam publicações com um conteúdo neutro.

Para os restantes casos, onde exista pelo menos 50% de certezas face à objetividade da publicação e ao seu contexto, o mapeamento do valor da polaridade para uma das três classes referidas foi realizado dividindo o intervalo de valores deste atributo de forma linear. A figura 3.2 procura descrever esquematicamente esta divisão.

Conjugando os resultados da API *TextBlob* com o mapeamento acima referido, foi possível observar que uma massiva maioria dos casos foi identificada apenas com um valor neutro de sentimentos. (Figura 3.3) Apenas cerca de 8% das instâncias do dataset foram identificadas como associadas a um sentimento negativo/positivo.

Sendo que as outras APIs de classificação textual, possíveis de utilizar para criar o atributo *target*, se baseiam nos mesmo processos de classificação de linguagem natural, conclui-se que o dataset pode não ser o melhor para treinar de forma viável uma RNA, devido à discrepância entre casos neutros com os casos associados a sentimentos negativos/positivos.

3.5 Redução Dataset

Tendo sido observado que o volume inicial do dataset se apresentaria como uma entrave para a construção de RNAs mais complexas e de maiores dimensões, foi tomada a decisão de reduzir significativamente o número de casos utilizados no processo de aprendizagem da rede.

Uma vez que a manipulação do conjunto de dados completo implicaria recursos de memória e tempo de processamento não disponíveis no decorrer deste projeto, foram assim criadas a partir dele cerca de 100 mil instâncias para Treino e Teste das RNAs. Este novo dataset, apesar de conter cerca de $\frac{1}{10}$ do volume inicial, apresenta ainda assim uma quantidade significativa de casos, filtradas com mais detalhe e rigor a nível de tradução para a língua inglesa.

Neste conjunto de dados foram selecionados 100 mil casos aleatórios do dataset inicial. Como pré processamento adicional, foi dado ênfase à limpeza de palavras com caracteres desconhecidos e referências a conteúdos sem informação útil para o processo de aprendizagem. Além disso, foi ainda utilizada a API do *Google Translator* para realizar a tarefa de tradução cuidada dos *tweets*. (No dataset completo não foi possível usar esta ferramenta devido ao tempo de tradução do dataset inteiro)

Esta redução permite aumentar a complexidade da rede, a nível de camadas intermédias e número de *Epochs*, levando à possibilidade de mais cenários de treino/teste de RNAs. Ao utilizar-se um dataset de menores dimensões, o tempo de aprendizagem da rede e recursos necessários para representar os *tweets* do dataset em formato de input da rede são significativamente menores.

4. Desenvolvimento da RNA

Uma vez que a fase de pré processamento do dataset foi implementada com funções auxiliares desenvolvidas em *Python*, para a frase de desenvolvimento da RNA manteve-se a mesma linguagem de programação. Assim, alem de se manter o mesmo ambiente de desenvolvimento, *Python* disponibiliza as principais APIs de criação de RNAs atualmente utilizadas.

Como plataforma escolhida para executar a rede, foi utilizada a ferramenta *Keras*. Esta ferramenta apresenta um conjunto de características que se adequam com o contexto deste projeto, nomeadamente:

- É bastante simples de usar, permitindo rapidamente obter uma estrutura de rede capaz de analisar os dados.

  Contudo, permite também evoluir a rede a nível de complexidade, interligando com outras plataformas de criação de RNAs (*TensorFlow, Theano, CNTK*) e permitindo ainda alterar o contexto de execução da RNA (CPU ou GPU);

- Devido ao volume do dataset apresentar mais de 900 mil instâncias, alimentar a rede com todos os dados numa fase de treino única exigia recursos e uma capacidade de processamento continua durante demasiado tempo.

  Contudo, os métodos da plataforma *Keras* permitem criar várias fases de treino, que evoluem a rede face ao estado de treino anterior. Desta forma é possível dividir o dataset em subconjuntos que são utilizados em fases de treino iterativas, exigindo assim menores recursos de memória e tempo de processamento na fase de aprendizagem;

- Pelo mesmo motivo do tópico anterior, treinar a rede sempre que for necessário realizar algum tipo de previsão não é viável em problemas com um elevado número de dados.

  Como solução, *Keras* disponibiliza funções para guardar o estado da RNA. É assim possível armazenar tanto a arquitetura, como os pesos das ligações dos axónios, após uma fase de treino ou teste da rede;

- *Keras* permite parametrizar praticamente todos os aspetos relacionados com a topologia da rede, como o número de camadas; ligações entre camadas; funções de ativação por camada; métricas de avaliação interna da rede e o *optimizer* (método de aprendizagem da rede).

Tendo por base a experiência anterior com outras plataformas de desenvolvimento de RNAs, como *R, Julia ou MXNet*, a plataforma *Keras* apresenta-se como a mais indicada e modular para este contexto. Focando o conjunto de motivos anterior e sendo atualmente uma plataforma bastante utilizada para a criação de RNAs, a plataforma *Keras* foi assim a escolhida para o desenvolvimento da rede deste projeto.

4.1 Representação Texto *Input* - Matrizes One-Hot

Uma das decisões essenciais do uso de RNAs num problema de classificação de texto, passa por codificar as palavras/frases que compõe o texto num formato que seja capaz de ser analisado e processado corretamente pelos neurónios da rede.

Devido à natureza da sua implementação, como o caso das funções de ativação que a rede utiliza na fase de aprendizagem, a sua estrutura está preparada para lidar apenas e preferencialmente com dados numéricos. Desta forma, uma palavra no seu formato literal e digital (*Strings*) não são assim formatos indicados para alimentar e treinar uma rede que se procure treinar de forma viável e adequadamente.

É assim necessário converter as palavras e a estrutura sequencial de uma frase num atributo (ou vários) caracterizado por valores numéricos. Encontrar formas de representação/tradução de linguagem natural para modelos de representação úteis em inteligência artificial não se apresenta como uma decisão simples, variando conforme o contexto sobre o qual se procura analisar a frase.

Os dois procedimentos mais utilizados para representar texto são associados com as estruturas *Vector Embeddings* ou Matrizes *One-Hot*. *Embeddings Vectors* são mapeamentos no espaço de palavras e/ou frases, em que a sua localização indica as semelhanças (Figura 4.1). Esta representação permite ainda associar relações semânticas, através da distancia dos vetores, sendo por isso muito usadas em contextos que envolvem a geração de analogias.

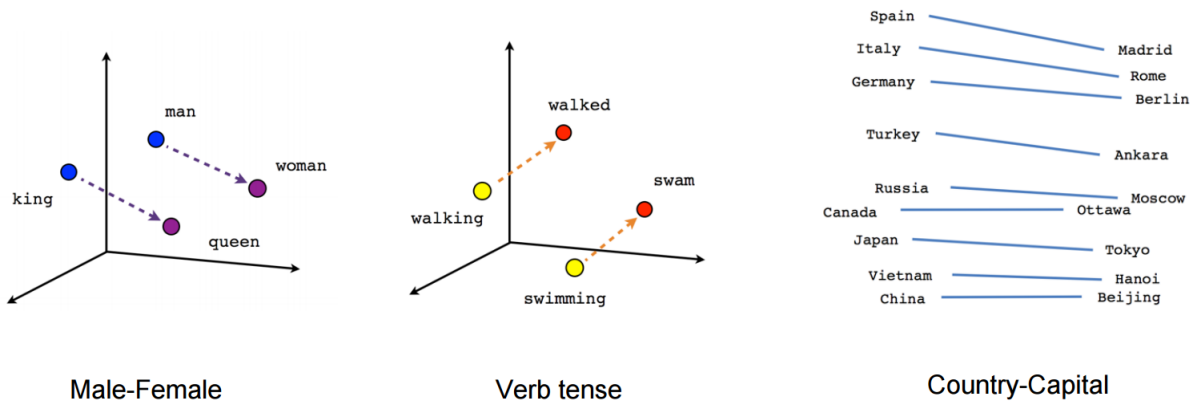


Figura 4.1: Representação simbólica de *Vector Embeddings*

Em oposição, as estruturas matriciais *One-Hot* não contêm informação de relações linguísticas entre as palavras. De uma forma geral, permitem apenas saber quais as palavras que surgem num determinado texto, não contendo informações sobre as relações entre as palavras, como a sua ordem ou semelhança na frase.

Para o contexto de classificação emotiva de texto, procura-se exatamente representar as palavras que se encontrem num determinado texto, num formato capaz de ser processado pelas camada de input de uma RNA. Nesse sentido, devido à sua simplicidade, as matrizes One-Hot foram o modelo de representação escolhido.

Este tipo de matrizes são chamadas de *One-Hot* já que cada matriz tem uma característica distinta ("hot") de todas as outras.

Considerando como exemplos as seguintes frases:

```
Errors should never pass silently.
Unless explicitly silenced.
```

A representação numa matriz One-Hot das seguintes frases começa por "partir"as frases numa lista (vetor) de palavras, sem ocorrência de repetições.

```
['errors', 'should', 'never', 'pass', 'silently', 'unless',
↪  'explicitly', 'silenced']
```

Depois de criada a lista de palavras únicas no texto a representar, é criado aquilo que pode ser visto como um dicionário com todas as palavras distintas, atribuindo um id único a cada palavra. Neste processo, não é relevante a ordem das palavras. Cada id será apenas um identificador único de cada palavras e irá representar apenas o índice *i,j* da matriz que representa aquela palavra.

```
{
      'errors': 0,
      'should': 1,
      'never': 2,
      'pass': 3,
      'silently': 4,
      'unless': 5,
      'explicitly': 6,
      'silenced': 7,
}
```

Distribuídos os ids pelas palavras encontradas, a matriz *One-Hot* terá tantas colunas quantas palavras encontradas no texto (dimensão dicionário) e, tantas linhas quantas as palavras encontradas na frase a codificar. No exemplo anterior, o dicionário de dados apresenta 8 palavras distintas e a primeira frase 5 palavras. Assim, uma matriz One-Hot que represente esta primeira frase terá dimensões 5x8.

Em cada linha, o valor 1 no indice i,j que representa a palavra indica a ocorrência da mesma na palavra. Em oposição, o valor 0 indica a não ocorrência da palavra identificada pela linha.

Para construir estas matrizes foram usados métodos disponibilizados na API *Pre Processing Text* da plataforma *Keras*. Nestas funções, as matrizes One-Hot devolvidas apresentam a vantagem de serem matrizes quadradas, podendo ser facilmente convertidas num vetor de palavras, considenrado apenas os valores da sua diagonal. Deste modo, o valor 1 num determinado indice indica a ocorrência, ou não, daquela palavra no texto representado pelo vetor de palavras.

```
[
     [1, 0, 0, 0, 0, 0, 0, 0], #errors
     [0, 1, 0, 0, 0, 0, 0, 0], #should
     [0, 0, 1, 0, 0, 0, 0, 0], #never
     [0, 0, 0, 1, 0, 0, 0, 0], #pass
     [0, 0, 0, 0, 1, 0, 0, 0], #silently
     [0, 0, 0, 0, 0, 0, 0, 0], #unless
     [0, 0, 0, 0, 0, 0, 0, 0], #explicitly
     [0, 0, 0, 0, 0, 0, 0, 0], #silenced
]
```

Na implementação utilizada, utilizaram-se vetores de palavras com 3500 posições, representando assim a 3500 palavras mais utilizadas. Infelizmente, apesar

desta representação ser bastante simples e eficaz no contexto, implica que as RNAs apresentem tantos nodos de *input* quantos índices do vetor.

Por este motivo e por questões de memória, para representar em vetores de palavras todos os tweets utilizados na fase de aprendizagem da rede, foram apenas utilizados vetores com 3500 posições.

4.2 Arquitetura da RNA

Para a definição da topologia da rede neuronal, foi utilizada uma arquitetura multi-layer FeedForward (Secção 2.3). Em *Keras* este tipo de rede designa-se como *sequencial* e representa assim uma rede *multi-layer perception*.

A criação de diferentes camadas camadas intermédias recorre ao método *Dense*, que na plataforma caracteriza uma camada de neurónios totalmente conectada com os neurónios da camada anterior.

```
# Exemplo da Definicao da topologia da rede
self.model = Sequential()
self.model.add(Dense(512, input_shape=(self.max_words,),
 ↪  activation='tanh', kernel_initializer="uniform"))
self.model.add(Dropout(0.5))
self.model.add(Dense(256, activation='tanh',
 ↪  kernel_initializer="uniform"))
self.model.add(Dense(128, activation='tanh',
 ↪  kernel_initializer="uniform"))
self.model.add(Dense(3, activation='tanh'))
```

Neste processo de criação de diferentes topologias foram tido alguns cuidados, destacando-se os seguintes aspetos:

- Na primeira camada intermédia, através do parâmetro ***input_shape*** a camada identifica automaticamente o número de neurónios da camada de input. Neste caso, existem tantos neurónios de input quantos elementos do vetor de palavras que representa a frase do *Tweet* (Secção 4.1);
- O argumento ***activation*** define a função de ativação a utilizar na camada.

  Muitas das principais funções utilizadas em contextos de RNAs apresentam apenas um contradomínio entre [0,1] ou [0, inf]. Uma vez que existem três classes target (Negativo, Neutro, Positivo), identificadas respetivamente pelos valores -1, 0, 1, foi tido o cuidado de utilizar funções de ativação que mantivessem o seu contradomínio entre [-1, 1].

  Utilizar funções sem contradomínio negativo, reduziria a capacidade de aprendizagem dos casos de publicações com nível emocional negativo. Contudo, como em teoria o processo de aprendizagem é capaz de lidar com nodos que sejam reduzidos ao valor 0, foram também utilizadas algumas funções com contradomínio diferente de [-1, 1].

  Nomeadamente, foram experimentadas algumas adaptações da função ReLU;
- O parâmetro ***Kernel initializer*** define qual o método utilizado para gerar o valor inicial dos pesos das ligações.

Neste projeto foram exploradas funções como a distribuição normal, uniforme e, maioritariamente, a distribuição de *Xavier* [8].

- Na tentativa de evitar cenários de overfitting, podem ser adicionadas camadas *Dropout* através de um método com o mesmo nome.

```
# Exemplo da adição de uma dropout layer com
↪ probabilidade = 0.2
self.model.add(Dropout(0.2))
```

O processo de *dropout* acontece apenas na fase de treino e ocorre depois da fase de ativação do neurónio. Esta camada opera de forma aleatória, colocando algumas ativações com o valor zero. Desta forma é simulado o conceito de "esquecimento", descartando alguma informação e "apagando"partes da rede neuronal em algumas iterações.

Contudo, estas camadas não devem ser introduzidas logo na fase inicial e foram exploradas apenas em cenários de overfitting ou na tentativa de obter melhores resultados de classificação.

4.3 Padrões de Treino

Depois de especificada a arquitetura que dará suporte à representação da rede neuronal, o processo de aprendizagem da rede pode ser dividido em duas etapas: a definição de uma método de aprendizagem, segundo um conjunto de métricas de avaliação e, a fase de análise e generalização efetiva dos dados, realizando o ajustamento dos pesos da rede aos mesmos.

4.3.1 Método de Aprendizagem

Para definir o comportamento de aprendizagem da rede é utilizado o método *Compile*.

Na escolha de um optimizador e métricas de avaliação, necessários de fornecer a este método, foram tomadas as seguintes considerações:

- O parâmetro *optimizer* define a técnica de aprendizagem utilizada. Neste projeto foi escolhido o método Adam [12].

  Este algoritmo, além de atingir bons resultados de forma rápida, apresenta ainda benefícios dos métodos AdaGrad (Gradiente descendente adaptativo) e RMSProp (*Root Mean Square Propagation*) [10].

  Uma vez que o método já está incluído dentro do *Keras*, basta fazer referência ao seu nome. Por omissão são utilizados os parâmetros aconselhados no artigo em que a técnica foi apresentada.

```
Keras.optimizers.Adam(lr=0.001, beta_1=0.9,
↪ beta_2=0.999, epsilon=None, decay=0.0,
↪ amsgrad=False)
```

- O parâmetros *loss* define a técnica de avaliação de desempenho da rede, através de uma função objetivo que o modelo da RNA irá minimizar.

  Apesar de várias implementações se basearem na métrica *Binary Cross Entropy*, esta adequa-se apenas a cenários com apenas 2 classes de previsão. Como foi assumido que neste projeto a RNA será capaz de classificar três níveis de emoção, terá que ser utilizado o método *Categórico*;

  Por se tratar de um problema de classificação, foram exploradas as opções *Mean Squared Error (MSE)* e *Categorical Cross Entropy* para este parâmetro.

- O último parâmetro indica uma lista de métricas a utilizar para avaliar a capacidade de previsão/classificação da rede.

  Neste processo de treino, por se tratar de um problema de classificação, foi mantida a métrica *Accuracy*.

O seguinte excerto de pseudo código procura descrever a definição do método de aprendizagem dentro da plataforma *Keras*.

```
# Definição arquitetura e topologia
model = Sequential()
# model.add( ... )
# (...)

# Definição processo aprendizagem RNA
model.compile(optimizer='adam',
 ↪  loss='categorical_crossentropy', metrics=['accuracy',
 ↪  'mse'])
```

4.3.2 Parâmetros Treino Rede

Para o treino da rede, é utilizada a função *fit*, que permite realizar o treino da rede num determinado contexto de execução.

Relativamente ao processo de treino, foram tomadas as seguintes considerações ao longo das sucessivas experiência de treino:

- O parâmetro *Epochs* define o número de passagens completas pelos dados, na fase de treino da RNA. O processo em que todas as instâncias de um dataset avança para a frente (*forward* propagation) na rede, até aos nodos de output e novamente para trás (*backward* propagation), até aos nodos de input, define assim um *Epoch*.

  Nos contextos em que uma rede treine sobre um dataset de pequenas dimensões, justifica-se que se explorarem valores de *Epochs* elevados, dado que atualizar os pesos das ligações apenas com poucas passagens pelos dados pode não ser suficiente.

  Se por um lado um baixo número de *Epochs* pode levar a *underfitting* da capacidade de aprendizagem da rede, valores elevados pode levar a tendências de *overfitting* aos dados.

  Como neste cenário o dataset apresenta uma grande volume de dados (perto de 1 milhão de linhas/ cerca de 600 mil para treino da RNA), o dataset contem por si só imensos exemplos diversificados de publicações. Por este motivo, o

número de passagens pelos dados foi explorado apenas com valores baixos, como 10, 12, 15, 20 ou 25.

Valores de Epochs superiores a 25 aliados a *Batchs* de pequenas dimensões, fazem crescer exponencialmente o tempo de treino da rede.

- o parâmetro *Batch_size* controla o número de exemplos presente num *Batch*.

  Apesar do dataset ser externamente dividido e processado em fases de treino iterativas, um subconjunto do dataset não pode ser passado integralmente à rede para treino. Assim, este parâmetro controla o número de instâncias que são alimentadas à rede em cada iteração de uma fase de treino.

  Batch sizes de maiores dimensões permitem que mais dados sejam processados em simultâneo, reduzindo o tempo de execução da fase de treino. Contudo, aumentar este parâmetro existe também uma maior quantidade de memória a cada iteração de treino da RNA.

  Dado que neste projeto os dados de imput são vetores inteiros com pelo menos 3500 índices (Secção 4.1), este parâmetro foi explorado apenas com os valores 50, 100, 150, 250, 500.

  Colocar valores mais pequenos não são viáveis, devido ao impacto que causa a nível do tempo de execução. Mesmo o uso de 50 ou 100 exemplos por iteração, para RNAs com arquiteturas mais complexas, não se tornou viável de executar devido a limitações de memória, recursos de processamento e tempo necessários. (Treinar uma RNA durante dias não se adequa ao espectro de resultados esperados neste projeto simples).

  Em contrapartida, o uso de valores demasiado elevados, como 1000 ou 10000, leva a fases de treino que terminem de forma muito rápida mas precoce, sem permitir que a rede generalize adequadamente os dados que analisa.

- O parâmetro *validation_split* permite indicar qual a percentagem de dados do conjunto de treino que devem ser reservados para teste do processo de aprendizagem. Como neste projeto o processo de treino é dividido em várias fases iterativas, devido às dimensões do dataset, não faz sentido criar avaliações de previsão em cada fase de treino.

  Assim, só no final de todos os subconjunto de treino serem analisados pela rede é que é realizada uma fase de avaliação de classificação. Para isso será utilizada a função *predict*, junto com um dataset inicialmente separado e reservado para treino.

- Apesar do dataset completo já ser previamente baralhado aleatoriamente, antes da fase de divisão entre casos de treino e teste da RNA, a opção *shufle* foi mantida com o valor *True*.

  Assim, cada subconjunto é novamente baralhado aleatoriamente, evitando quaisquer possíveis tendências de generalização, que pudessem levar a overfitting.

O seguinte excerto de pseudo código procura descrever a chamada do método *fit*.

```
# Treino parcial da RNA, com 1 subconjunto do Dataset total
model.fit(train_x, train_y, batch_size=150, epochs=2,
↪  verbose=1, validation_split=0, shuffle=True)
```

5. Análise dos resultados

Neste capitulo serão apresentadas algumas das observações e resultados obtidos ao longo dos diferentes cenários de construção e treino de RNAs para resolver o problema de classificação de testo.

	Parâmetros			Função Ativação	Método Fit						Fase Avaliação Desempenho				
					Métrica Fase treino			Métricas Fase Teste			Método Evaluate			Método Predict	
Teste	Epochs	Batch size	Camadas Intermédias		MSE	Cross Entropy	Accuracy	MSE	Cross Entropy	Accuracy	Cross Entropy	MSE	Accuracy	MSE	Accuracy

Figura 5.1: Informações registadas em cada cenários de teste e topologia de Rede exploradas

O sumário dos resultados estará descrito através de uma tabela, caracterizada pelo formato visível na figura 5.1 e que apresenta a seguinte estrutura:

- Cada coluna representa um parâmetro relevante no processo de treino ou teste da rede, como o número de *Epochs*, *Batch Size*, topologia da rede ou métricas de avaliação conseguidas;
- Na coluna *Camadas Intermédias*:
  - O número de camadas intermédias definidas é dado pelo comprimento do vetor;
  - O número de nodos de cada camada, é dado pelo valor do índice *i* que representa essa camada;
  - Camadas de Dropout são identificadas pela letra *D*.
- As *Métricas da Fase Teste* dentro da opção *Método Fit* (a amarelo) representam as avaliações nos cenários em que no método fit se utilizou o parâmetro *Percentage split* com valor 0.33;
- A secção laranja, representa a avaliação do desempenho da rede, através dos métodos predict e evaluate disponibilizados na plataforma keras.
  Em ambos as funções referidas é utilizado o mesmo dataset de teste, previamente criado e não utilizado em fase de treino.

Tabela 1

Dataset Inglês 100k instâncias
Dimensão vetor palavras: 3500

Teste	Parâmetros			Função Ativação	Método Fit						Fase Avaliação Desempenho					
					Métrica Fase treino			Métricas Fase Teste			Método Evaluate			Método Predict		
	Epochs	Batch size	Camadas Intermédias		MSE	Cross Entropy	Accuracy	MSE	Cross Entropy	Accuracy	Cross Entropy	MSE	Accuracy	MSE	Accuracy
1	15	500	[512, 256, 128]	tanh	0,1496	0,0891	0,9762	0,1586	0,1855	0,9519	0,5755	0,067	0,8897	0,116	0,8897
2	15	100	[512, D, 256, D, 128]		0,1274	0,0712	0,9808	0,1006	0,2031	0,9492	0,6686	0,069	0,8888	0,1193	0,8888
3	5	100	[512, D, 256, D, 128]		0,1631	0,1165	0,9658	0,1714	0,1601	0,9536	0,5786	0,0693	0,8867	0,1225	0,8867
4	3	10	[512, D, 256, D, 128]		0,0275	0,134	0,9607	0,0278	0,1529	0,9553	0,5682	0,0706	0,8841	0,12453	0,8841
5	15	150	[512, D, 128]	ReLU	0,0326	0,1533	0,9591	0,0263	0,171	0,9664	0,489	0,0689	0,8882	0,1192	0,8882
6	25	150	[512, D, 128]		0,0201	0,055	0,998	0,0464	0,3951	0,9509	-	-	-	-	-
7	60	350	[300, D, 200, D, 100]	LeakyReLU	1,299	0,0695	0,9872	1,299	0,3251	0,9479	0,2023	1,2953	0,8842	3,867	0,8843

Tabela 2

Dataset Completo 1M instâncias
Dimensão vetor palavras: 3500

Teste	Parâmetros			Função ativação	Método Fit						Fase Avaliação Desempenho					
					Métrica Fase treino			Métricas Fase Teste			Método Evaluate			Método Predict		
	Epochs	Batch size	Camadas Intermédias		MSE	Cross Entropy	Accuracy	MSE	Cross Entropy	Accuracy	Cross Entropy	MSE	Accuracy	MSE	Accuracy
1	20	500	[256,128,60]	LeakyReLU	0,0021	0,008	0,9978	0,02	0,2706	0,9682	1,4395	0,0799	0,8776	0,13609	0,87752
2	12	200	[128,D, 80,D]		0,0174	0,0777	0,9781	0,0199	0,1266	0,9713	0,7248	0,0774	0,8811	0,1274	0,8811
3	10	50	[512, 256]	tanh	0,0224	0,1384	0,9623	0,193	0,1206	0,9692	0,4512	0,0752	0,8749	0,1362	0,8748
4	10	50	[512,D, 256]		0,0296	0,1767	0,9517	0,0217	0,1332	0,9632	0,4945	0,0715	0,884	0,1177	0,884
5	12	250	[512, D, 256]		0,0161	0,1048	0,9714	-	-	-	0,5755	0,067	0,8897	0,116	0,8897
6	10	150	[1500, 512, 256]		0,0173	0,1124	0,96	-	-	-	0,5786	0,0693	0,8867	0,1225	0,8867
7	15	150	[512, D, 128]	ReLU	0,0326	0,1533	0,9591	-	-	-	0,489	0,0689	0,8882	0,1192	0,8882
8	10	50	[512,D, 256]	Sigmoid	0,0107	0,0661	0,98	-	-	-	0,6686	0,069	0,8888	0,1193	0,8888
9	12	250	[128, 60]	PReLU	0,0149	0,0986	0,9737	-	-	-	0,5682	0,0706	0,8841	0,12453	0,8841

Figura 5.2: Tabelas com os parâmetros e resultados dos cenários de avaliação de desempenho das RNAs criadas.

5.1 Dataset completo *Tweets*

A Tabela 1 da Figura 5.2 descreve os resultados os cenários de treino de RNAs realizados sobre a versão reduzida do dataset.

Focando alguns dos resultados obtidos:

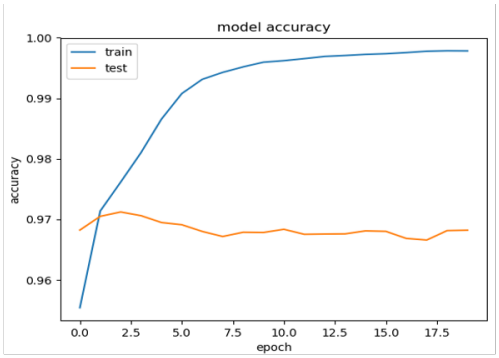
- Numa primeira abordagem ao dataset completo, foram desde logo utilizadas algumas considerações destacadas nas secções 4.3.2.

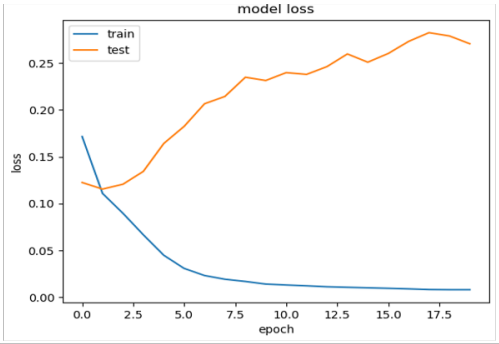
  Para o Teste 1, foi inicialmente criada uma topologia de rede com 3 camadas intermédias, tendo cada uma delas, respetivamente, 256 e 128 e 60 neurónios.

  Na tentativa de obter rapidamente bons resultados, utilizou-se como forma de inicialização dos pesos o método de *Xavier* e como funções de ativação dos neurónios de cada camada selecionou-se a função LeakyReLU.

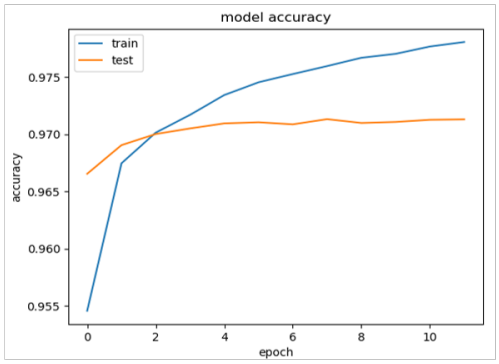
  Foram realizadas 20 passagens pelos dados, alimentando a rede com 500 instâncias a cada iteração do treino.

Teste 1
- Batch size = 500
- Epochs = 20
- LeakyReLU camadas intermédias





Teste 2
- Batch size = 200
- Epochs = 12
- LeakyReLU camadas intermédias
- Criação *Dropout layers*



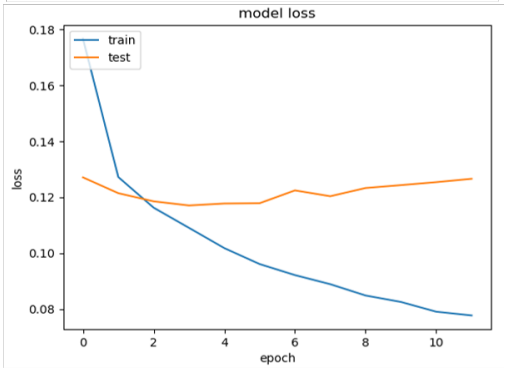


Figura 5.3: Resultados Teste 1 e 2, com uso do dataset completo para criação de conjuntos de treino + teste.

Estes resultados (Figura 5.3) tornam-se relevantes no sentido em que, apresentam desde logo um cenário de overfitting. A accuracy da rede no final da fase de treino apresenta-se na casa dos 99% e, na fase de teste, atinge apenas os 88%.

Na tentativa de reduzir este sobre ajustamento aos dados, no Teste 2 procurou-se introduzir camadas de *Dropout* na rede e simplificar a sua complexidade,

reduzindo a topologia para duas camadas com, respetivamente, 128 e 80 nodos.

Apesar destas alterações reduzirem o valor de *accuracy* em fase de treino, o valor de 97% continua relativamente superior face aos 88% novamente obtidos na fase de teste.

- Independente da função de ativação utilizada nas camadas intermédias da rede, mesmo que o seu contradomínio não se apresente entre [-1, 1], como no caso da ReLU ou da Sigmoide, foi mesmo assim possível obter uma taxa de acertos de cerca de 88% em fase de Teste da RNA. (Relembrando que o output deve ser -1, 0 ou 1, conforme a classificação de um texto num sentimento negativo, neutro ou positivo)

  Estes resultados permitem observar que as camadas intermédias são capazes de generalizar a informação que analisam, mesmo que não propaguem entre si sinais com valores negativos.

  Ao longo das iterações de treino, as sinapses das camadas intermédias com estas funções ReLU ou Sigmoide não transmitem valores negativos, porque o seu contradomínio é positivo. Contudo, o peso das ligações (*W*) pode eventualmente ajustar-se para valores negativos, sendo assim capazes de influenciar a aprendizagem da rede a ajustar-se inevitavelmente aos dados.

- Relativamente ao Teste 7, foi utilizada uma topologia de rede com 3 camadas intermédias, possuindo 300, 200 e 100 neurónios, respetivamente. Foi também a inicialização dos pesos com o método de *Xavier* e utilizada a função LeakyReLU.

Teste 7
- Batch size = 350
- Epochs = 60
- LeakyReLU camadas intermédias

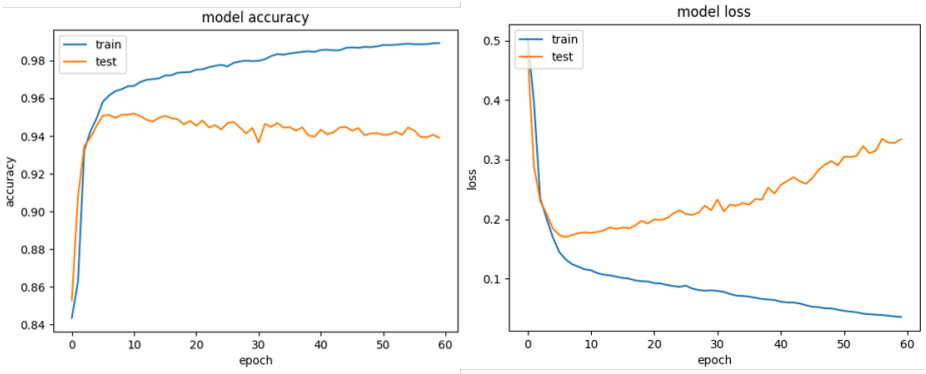


Figura 5.4: Resultados do Teste 7, com uso do dataset completo para criação de conjuntos de treino + teste.

A análise dos resultados conseguidos com esta arquitetura e apresentados na Figura 5.4 é interessante devido à evolução das curvas a cada *Epochs*.

O surgimento de picos (altos e baixos em *accuracy*) ao longo das diferentes *Epochs* permite concluir que com estas configurações, a função de ativação ou as métricas de avaliação não são as mais indicadas. Isto porque, num determinado *Epoch*, a atualização dos gradientes através da métrica *Loss*

definida leva a bons resultados mas, num *Epochs* seguinte, os resultados já não são favoráveis.

- No geral, independente da topologia ou parâmetros utilizados, os valores de accuracy em fase de teste, com dados exclusivamente reservados para treino, rondam sempre os 88%.

5.2 Dataset *Tweets* em Inglês

A Tabela 2 da Figura 5.2 descreve os resultados os cenários de treino de RNAs realizados sobre a versão reduzida do dataset e segue a mesma estrutura descrita na secção anterior.

Relembrando, este conjunto de dados contem apenas publicações em inglês e, por apresentar menores dimensões, permite explorar vetores de palavras com maiores dimensões e redes com maior complexidade, pois dá mais folga a nível dos recursos de memória necessários para o analisar.

Apenas descrevendo alguns dos principais resultados obtidos:

- Numa primeira abordagem (Teste 1 e 2), foi utilizada uma topologia de rede com 3 camadas intermédias, tendo cada uma delas, respetivamente, 512, 256 e 128 neurónios.

  Como funções de ativação utilizou-se a função tangente hiperbólica *tanh* e os pesos das ligações foram inicializados com uma distribuição uniforme.

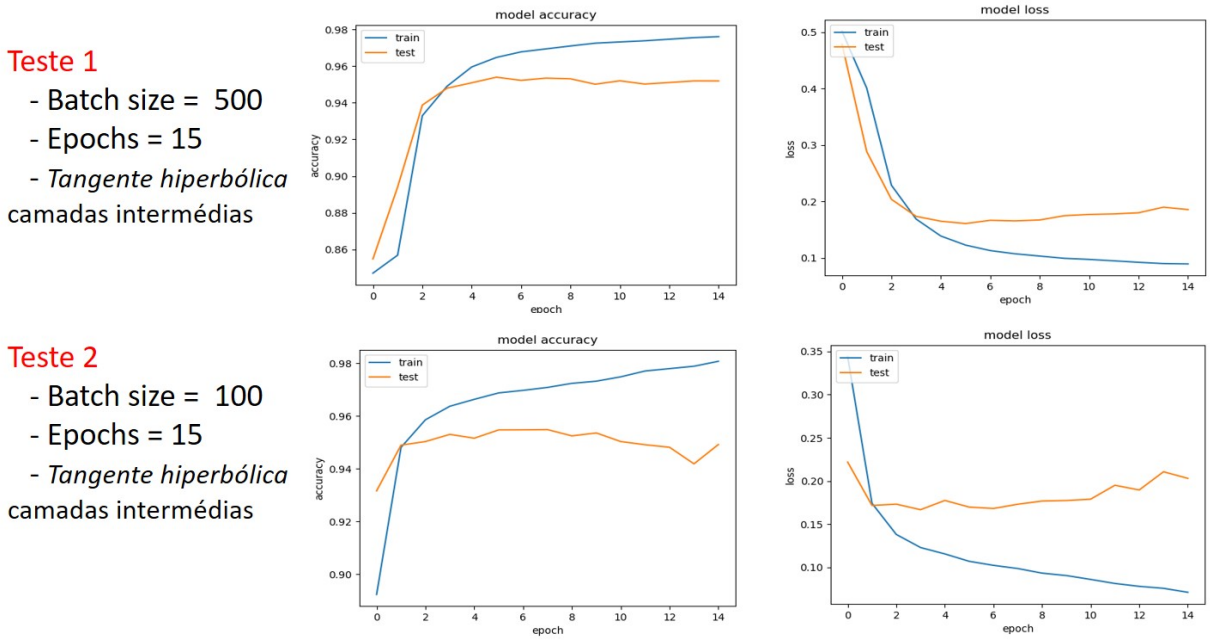


Figura 5.5: Resultados Teste 1 e Teste 2 com o dataset reduzido apenas com publicações em inglês

Observando os gráficos da Figura 5.5, as retas descreverem inicialmente um cenário de overfitting, onde os valores de *accuracy e cross entrophy (loss)* na

fase de treino se apresentam ligeiramente superiores face à fase de avaliação de desempenho.

No Teste 2, realizar um teste com o mesmo número de *epochs* (15) mas reduzindo o parâmetro *batch size* de 500 para 100 exemplos contribuiu apenas para acentuar ligeiramente os resultados de *overfitting* já assinalados.

O crescimento suave da curva de *accuracy* na fase de treino, permite assumir que a função de ativação e avaliação internas da rede levam a um correto ajustamento dos pesos a cada iteração, permitindo assim uma aprendizagem gradual, sem grandes variações entre cada *Epoch*.

- Reduzir o número de *Epochs* para um valor abaixo das dezenas (Teste 3 e Teste 4) não permite nenhuma avaliação concreta ao desempenho de aprendizagem da rede. As informações dos gráficos nestes cenários não são suficientes para compreender a evolução da rede na fase de treino.

  Contudo, as métricas devolvidas foram registadas e enquadram-se dentro dos resultados dos restantes testes.

- Relativamente ao Teste 6, foi aumentado o número de *Epochs*, procurando tirar o melhor partido possível da função de ativação *ReLU*.

  Contudo, o maior número de *Epochs* permitiu apenas aumentar o efeito de *overfitting* já visualizado noutros cenários de treino. A accuracy a nivel de treino manteve-se próxima dos 88%.

Teste 6
- Batch size = 150
- Epochs = 25
- ReLU camadas intermédias

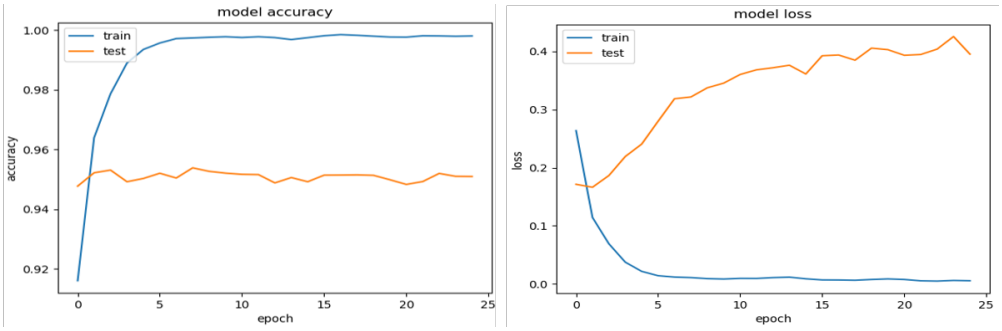


Figura 5.6: Resultados Teste 6, com o dataset reduzido apenas com publicações em inglês

6. Trabalho Futuro

Em contextos reais de exploração de datasets, através de algoritmos de *Machine Learning*, muitos dos conjuntos de dados encontram-se em bruto e com bastante ruído pelo meio. Neste projeto o foco principal não se encontra na extração de conhecimento deste dataset em especifico, mas sim na criação de uma RNA capaz de realizar a classificação de publicações.

Nesse sentido, poderia ser relevante reunir um conjunto de publicações da rede social *twitter* através de diversas APIs disponibilizadas na linguagem *Python*, sendo assim possível filtrar desde inicio o contexto da publicação e o idioma em que a mesma é recolhida.

Este aspeto é frisado por se ter observado apesar do dataset ter um grande volume de dados, a maioria dos exemplos são vagos e associados apenas a eventos ou referências a *websites*, não apresentando assim qualquer tipo de informação sentimental. Das 990472 instâncias utilizadas do dataset, cerca de 92% foram classificadas como neutras pela API utilizada para criar os atributos *target*, sobre os quais a rede desenvolve o seu processo de aprendizagem supervisionado.

Por este motivo, uma melhor seleção dos *Tweets* recolhidos ou uma redução do dataset para equilibrar o número de casos neutros com os casos associados a sentimentos negativos/positivos será uma possível mais valia para o projeto e capacidade de previsão das RNAs criadas.

Ainda assim, no sentido de tirar o melhor partido do dataset fornecido, foi realizada uma redução das dimensões do mesmo. Desta modo foi possível focar em especifico apenas publicações que se encontrassem em inglês. Uma vez que a API de análise sentimental de texto e representação de frases em vetores de palavras está essencialmente focada na língua inglesa, procurou-se assim melhorar a qualidade dos dados fornecidos à rede.

Ao utilizar um dataset de menores dimensões mas ainda assim significativo (100 mil instâncias), foi possível reduzir o tempo de aprendizagem e recursos necessários para executar o treino da RNA. Com estas vantagens, foi possível explorar mais configurações e topologias de RNAs, no sentido de obter melhores resultados.

Uma vez que uma RNA não está desenvolvida para ser "alimentada"com conteúdo textual na forma de *Strings*, outro ponto chave deste projeto está ligado com a representação de texto num formato capaz de ser interpretado e fornecido de forma coerente à camada de *input* da rede.

Neste projeto a representação de uma frase recai num vetor de palavras, sendo que cada índice do vetor representa uma determinada palavra. Esta representação está limitada a nível das dimensões do vetor, principalmente se a cada *Epoch* da rede for necessário ter preparados um número elevado (>500) de instâncias para processar numa iteração de treino.

Nos testes realizados foram utilizados vetores com 3500 posições, referentes às 3500 palavras mais utilizadas na língua inglesa. Contudo, outras formas de representação textual poderiam ser exploradas, como *N-grams*, no sentido de comparar os resultados obtidos a nivel de capacidade de previsão da RNA.

Contudo, devido ao espetro temporal do projeto, a equipa teve que se manter apenas com a representação inicialmente construida para representar texto.

Outra técnica bastante utilizada em contextos de *Machine Learning* passa por utilizar arquiteturas previamente criadas e com bons resultados estudados num contexto semelhante. Nesse sentido, a abordagem realizada por Pedro M. Sosa em [14] poderia ser um bom ponto de partida, se o seu trabalho fosse disponibilizado com mais detalhe. Na sua abordagem, o uso de redes convulocionais, com ligações recorrentes, permite ter em conta o contexto da frase num todo, dado que palavras já "processadas"influenciam as iterações posteriores, com as palavras seguintes do mesmo input.

7. Conclusão

No sentido de desenvolver uma Rede Neuronal Artificial capaz de realizar a classificação de texto em níveis emocionais distintos, os capítulos anteriores apresentam todas as etapas desenvolvidas ao longo deste projeto.

O capitulo inicial, onde se introduzem os principais conceitos associados com RNAs, apresenta-se essencial, no sentido em que foca todos os aspetos necessários para compreender muitas das decisões tomadas ao longo do projeto, relacionadas com o modo de funcionamento e aprendizagem destas estruturas.

Relativamente ao dataset utilizado, destaca-se a dificuldade encontrada na fase de pré processamento, até ser possível obter informação útil e totalmente "limpa"do mesmo. Só depois de converter os textos das publicações num formato uniforme, quer a nível de codificação de caracteres como de linguagem, é que foi possível aplicar métodos para determinar o valor sentimental de cada publicação e ser-se capaz de adaptar os *tweets* para um formato de *input* capaz de alimentar as RNAs construidas.

A nível dos padrões de treino executados, foram explorados praticamente todos os aspetos possíveis de configurar numa RNA. Estas configurações foram realizadas de acordo com o contexto do problema e objetivos a alcançar, focando as principais técnicas utilizadas atualmente.

Neste processo destaca-se a exploração de diferentes topologias, métodos de inicialização de pesos e funções de ativação, como as atuais variações da função ReLU. No geral, quer utilizando o dataset completo quer utilizando uma versão reduzida e exclusivamente com *Tweets* em inglês, sem traduções posteriores, verifica-se que a accuracy média a nível da fase de teste atinge os 88%. Qualquer um dos modelos propostos, baseados neste dataset com as limitações referidas ao longo do relatório, fica assim "fixo"a esta capacidade de previsão.

A conjugação dos diferentes parâmetros anteriormente referidos, permite atenuar ligeiramente efeitos de overfitting aos dados de treino. Contudo, a capacidade de previsão da rede mantém-se bastante semelhante em qualquer um dos diversos cenários de Treino/Teste realizados e interpretados ao longo do Capitulo 5.

Em suma, apesar da qualidade de um processo de treino de uma RNA estar intrinsecamente ligado com os parâmetros e técnicas utilizadas na fase de aprendizagem, destaca-se a importância que a qualidade dos dados utilizados também desempenham na fase de treino. Como consequência deste aspeto, apesar do tratamento do dataset ter sido uma parte constante ao longo do projeto, outras formas de representação de texto e cálculo do valor sentimental de cada *Tweet* seriam relevantes de explorar para obter resultados mais sólidos de classificação por parte das RNAs criadas.

Bibliografia

[1] Google translate api for python. `https://pypi.python.org/pypi/googletrans`. [Online; accessed 27-Março-2018].

[2] Microsoft azure text analytics. `https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/`. [Online; accessed 28-Março-2018].

[3] Microsoft cognitive services. `https://azure.microsoft.com/en-us/services/cognitive-services/`. [Online; accessed 28-Março-2018].

[4] Natural language toolkit. `https://www.nltk.org/#natural-language-toolkit`. [Online; accessed 25-Março-2018].

[5] Nltk api for python. `http://text-processing.com/demo/sentiment/`. [Online; accessed 25-Março-2018].

[6] Textblob library for python. `http://textblob.readthedocs.io/en/dev/`. [Online; accessed 26-Março-2018].

[7] Paulo Cortez e José Neves. Redes neuronais artificiais. *IEEE Computer Graphics and Applications*, 2000.

[8] Xavier Glorot and Yoshua Bengio. *Understanding the difficulty of training deep feedforward neural networks*. 2010.

[9] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014.

[11] Vinicius Goncalves Maltarollo, Kathia Maria Honorio, and Alberico Borges Ferreira da Silva. Applications of artificial neural networks in chemical problems. 2013.

[12] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2016.

[13] Ranjan Satapathy, Claudia Guerreiro, Iti Chaturvedi, and Erik Cambria. Phonetic-based microtext normalization for twitter sentiment analysis. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 407–413, 2017.

[14] Pedro M. Sosa. Twitter sentiment analysis using combined lstm-cnn models. June 7, 2017.

[15] Selva Staub, Emin Karaman, Seyit Kaya, Hatem Karapınar, and Elçin Güven. Artificial neural network and agility. *Procedia - Social and Behavioral Sciences*, pages 1477 – 1485, 2015. World Conference on Technology, Innovation and Entrepreneurship.