UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME DE BRITO FREIRE

SUMARIZAÇÃO HÍBRIDA COM REDES POINTER-GENERATOR

RIO DE JANEIRO
2018

GUILHERME DE BRITO FREIRE

SUMARIZAÇÃO HÍBRIDA COM REDES POINTER-GENERATOR

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. João Antonio Recio da Paixão

RIO DE JANEIRO

2018

CIP - Catalogação na Publicação

R484t

Ribeiro, Tatiana de Sousa
Titulo / Tatiana de Sousa Ribeiro. -- Rio de Janeiro, 2018.
44 f.

Orientador: Maria da Silva.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2018.

1. Assunto 1. 2. Assunto 2. I. Silva, Maria da, orient. II. Titulo.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a).

GUILHERME DE BRITO FREIRE

SUMARIZAÇÃO HÍBRIDA COM REDES POINTER-GENERATOR

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 21 de Agosto de 2019

BANCA EXAMINADORA:

Prof. João Antonio Recio da Paixão
D.Sc. - PUC

Prof. João Carlos Pereira da Silva
D.Sc. - COPPE/UFRJ

Hugo Cesar de Castro Carneiro
D.Sc. - COPPE/UFRJ

Felipe Fink Grael
M.Sc. - COPPE/UFRJ

Fabrício Firmino de Faria
M.Sc. - UFRJ

"I refuse to say anything beyond five years
because I don't think we can see much beyond five years."

Geoffrey Hinton

RESUMO

Esse projeto procura introduzir o ramo de sumarização automática, bem como estudar um modelo de sumarização híbrida extrativa e abstrata proposto recentemente com redes neurais: Pointer-Generator Network. O trabalho explica profundamente o funcionamento desse modelo, assim como explicita os desafios de uma implementação prática. O projeto utiliza o modelo em um dataset de sumarização não convencional construído a partir de *posts* na internet. Por fim, considerações finais e críticas são apresentadas, bem como trabalhos futuros.

Palavras-chave: pointer generator. rede neural. sumarização abstrata.

ABSTRACT

This project introduces the field of automatic summarization as well as studies a recently proposed extractive and abstractive hybrid summarization neural network model: the Pointer-Generator Network. It deeply explains the inner workings of the model, as well as exposes the challenges of a practical implementation. The project also uses the model on a different kind of summarization dataset made out of internet posts. At the end, final thoughts are shown alongside criticism and future works.

Keywords: pointer generator. neural network. abstractive summarization.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

GRU	Gated Recurrent Unit
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network

SUMÁRIO

1 INTRODUÇÃO

Esse trabalho começa explorando o cenário atual de sumarização automática. Serão expostos diferentes métodos de se construir resumos a partir de textos sem auxílio humano. Uma técnica recente promissora será estudada de modo mais detalhado, bem como críticas e sugestões de melhorias serão feitas e analisadas.

É fácil ver como resumos são importantes e valiosos em diversos cenários da vida moderna. Existem utilizações no meio acadêmico, como a leitura do resumo antes de se comprometer a ler um artigo completo. Muitos leem o resumo de um capítulo de um livro didático para reforçar as ideias principais tratadas no mesmo. Para fins jornalísticos, um resumo pode disponibilizar um panorama geral de um assunto complexo ocorrendo ao redor do globo.

Também é possível ver a utilização de resumos no ramo jurídico. Diariamente, juízes e advogados se deparam com pilhas de processos, as quais devem analisar com cuidado. Entretanto, o volume de texto e a velocidade na qual precisam lidar com tais processos tornam essa tarefa um tanto árdua. Possuindo acesso a resumos dos documentos, profissionais do ramo legal podem agilizar os procedimentos necessários para que os processos sigam curso de forma mais rápida.

É possível enxergar uma demanda por sumarização cada vez maior, ao levar em conta o ritmo cada vez mais acelerado de produção de conteúdo, especialmente, textual. Principalmente com redes sociais e dispositivos de internet das coisas, a produção de conteúdo nunca esteve maior (HAJIRAHIMOVA; ALIYEVA, 2017).

Levando em conta essa demanda, algoritmos que permitem a sumarização automática de textos trazem alternativas para tornar esse nível de produção de resumos uma realidade.

A sumarização automática de textos está dentro do vasto ramo de Processamento de Linguagem Natural (NLP) (RADEV; HOVY; MCKEOWN, 2002). Nele são estudadas diversas formas de se criar resumos a partir de textos sem intervenção humana. Observando os métodos existentes de sumarização, é possível categorizá-los em duas grandes vertentes: a **extrativa** e a **abstrata** (EDMUNDSON, 1969).

Essas vertentes, extrativa e abstrata, de realizar sumarização podem ser equiparadas a um "marca-texto" e uma caneta, respectivamente. Um marca texto destaca passagens relevantes do texto original a fim de torná-lo mais conciso. Da mesma forma, a abordagem extrativa seleciona trechos do texto original e os ordena para formar um resumo.

Seguindo a analogia, o método abstrato se assemelha à caneta. Nessa abordagem, o leitor lê o texto, entende seu conteúdo e, com suas próprias palavras, escreve um resumo usando uma caneta. De forma semelhante, a abordagem abstrata cria uma representação do texto original e gera um texto com "suas próprias palavras" que seja um resumo do original.

1.1 SUMARIZAÇÃO EXTRATIVA

Os algoritmos do paradigma extrativo foram os primeiros a serem inventados e concretizaram a sumarização automática como uma realidade (EDMUNDSON, 1969). O que todos os algoritmos dessa categoria têm em comum é o fato de que produzem um resumo composto apenas de trechos do texto original.

Em sua maioria, algoritmos de sumarização extrativa selecionam frases que julgam relevantes no texto original e as utilizam para compor o resumo. Alguns selecionam partes e não frases completas, mas em todos os casos, os resumos são feitos a partir de trechos copiados do documento original.

De modo geral, os algoritmos extrativos podem ser divididos em duas subcategorias. A primeira são os algoritmos baseados em um sistema de pontuação (WONG; WU; LI, 2008). Esse tipo de algoritmo estabelece um critério para pontuar cada frase do texto original e constrói um novo texto a partir das frases com maior pontuação. A ideia é que frases com pontuação mais alta sejam mais relevantes ou importantes para o texto do que as outras.

Só com essas especificações há diversos parâmetros que se podem ajustar, talvez o mais importante sendo o método de pontuação. Abordagens iniciais desse método utilizavam propriedades baseadas em frequência para mensurar a relevância de frases. Conforme foram evoluindo, os algoritmos passaram a englobar o aspecto semântico das frases.

A segunda grande subcategoria dos algoritmos extrativos abrange os algoritmos de clusterização (FUNG; NGAI; CHEUNG, 2003). Algoritmos dessa categoria utilizam técnicas de agrupamento para juntar frases que têm significado parecido ou estão tratando do mesmo assunto.

Em geral, uma vez com os clusters formados, cada um representaria uma ideia do texto original. Sendo assim, para construir um resumo, basta selecionar uma frase representante de cada cluster e juntá-las em um texto. Há diversas formas de se selecionar um representante do cluster, a mais simples delas sendo escolher a frase mais próxima ao meio de determinado cluster.

1.2 PROBLEMAS DA ABORDAGEM EXTRATIVA

Os modelos extrativos, em geral, são mais simples de se implementar, devido à sua natureza de copiar frases do texto original ao invés de gerá-las por conta própria. Essa facilidade de implementação torna essa classe de algoritmos atrativa para aplicações de mundo real. Entretanto essa simplicidade vem com um custo.

Justamente pelo fato de só poderem copiar trechos do texto original, os algoritmos extrativos têm sua expressividade limitada. Seu léxico está restrito àquele usado pelos autores do texto original. As ideias centrais do texto podem não ser facilmente comprimidas em poucas frases, quando só utilizando o que está disponível no texto original.

As frases originais não só restringem a capacidade de síntese desses algoritmos, como também o foco e o público ao qual se dirigem. A tarefa se torna difícil se o objetivo de resumir um texto for para apresentar para um público de outra área. Pouco pode ser feito quanto a jargões, termos técnicos ou linguajar específico de determinado grupo ou área não necessariamente conhecidos pelo leitor do resumo.

Além dos problemas destacados acima, a abordagem extrativa também pode conter problemas estruturais. Por se tratar de recortes do texto original, o resumo pode acabar não coeso e com frases que não se encaixam muito bem. Se o algoritmo tomar como unidade básica trechos de frases, pode inclusive acabar com um resumo que não faz muito sentido.

Seguindo esse raciocínio, faz mais sentido pensar nesses resumos como listas de tópicos abordados no texto, do que um texto em prosa propriamente dito.

1.3 SUMARIZAÇÃO ABSTRATA

Desenvolvidos após os primeiros algoritmos extrativos, os algoritmos abstratos surgem contornando as limitações de sua classe irmã. A grande diferença dessa categoria é a capacidade de criação de texto novo. O fato de poderem parafrasear conteúdos do documento original elimina uma grande restrição presente nos algoritmos extrativos. Os modelos abstratos são, em sua maioria, divididos em duas fases: compreensão/síntese de conteúdo e geração de texto.

A primeira parte - compreensão de conteúdo - diz respeito à geração de uma representação abstrata do texto original. Muitas vezes essa representação abstrata se dá na forma de um vetor em um espaço de alta dimensão. A tarefa de vetorização de textos em si é uma categoria de estudo extensa - originando diversos métodos, desde o clássico TFIDF até mais modernos como word2vec (MIKOLOV et al., 2013a) e FastText (BOJANOWSKI et al., 2017a).

Uma vez com uma representação vetorial do texto, uma série de técnicas podem ser utilizadas de forma a gerar uma representação final do resumo. Essa representação será utilizada para gerar o texto na segunda fase da sumarização abstrata.

Gerar conteúdo textual com qualidade humana é uma tarefa notoriamente complexa para máquinas. Por caminhar por essa vertente, a abordagem abstrata precisa de algoritmos mais elaborados e robustos.

Uma abordagem cada vez mais popular entre pesquisadores da área é a de Redes Neurais. Modelos baseados, principalmente, em Redes Neurais Recorrentes (RNN) têm se mostrado extremamente versáteis e capazes em diversas tarefas de geração de texto. Essa abordagem trouxe ao ramo modelos estado da arte e tem trazido cada vez mais estudos ao ramo ((CHOPRA; AULI; RUSH, 2016); (NALLAPATI et al., 2016); (SEE; LIU; MANNING, 2017)).

Passando a utilizar redes neurais, é possível adaptar a parte geradora dos algoritmos anteriores para a incorporar esse novo modelo. Entretanto, mais do que modificar algoritmos existentes, torna-se possível uma abordagem nova.

Conhecidos como *end to end models* (modelos ponta a ponta), esses modelos delegam toda a tarefa de sumarização para uma rede neural. Desde a entrada do problema até sua saída, o processamento é feito completamente por uma única arquitetura de rede neural.

Apesar de esforços recentes para interpretar o funcionamento minuciosamente de redes neurais, problemas resolvidos sob tal paradigma foram até então tratados como caixa preta (OLAH et al., 2018). Em paralelo aos esforços de interpretabilidade, os modelos de ponta a ponta procuram contornar essa questão de outra forma. Ao invés de segmentar o problema original em pedaços menores e designar a cada parte uma rede neural separada, nas arquiteturas *end to end* o problema todo é delegado a apenas uma arquitetura.

Simplesmente entregando a uma arquitetura uma entrada sem muita engenharia de atributos e a saída esperada, o modelo consegue um fluxo de processamento eficaz. O modelo todo sendo diferenciável, torna seu desempenho mais alto, pois cada parte da arquitetura está sendo otimizada em conjunto por um otimizador. Contrastando com métodos onde etapas são segmentadas e treinadas separadamente, os modelos de ponta a ponta mostram resultados promissores ((CHIU et al., 2018), (DUŠEK; NOVIKOVA; RIESER, 2019)).

1.4 PROBLEMAS DA ABORDAGEM ABSTRATA

Por gerar seu próprio texto, um modelo abstrato não possui os mesmos problemas presentes em um modelo extrativo. Entretanto, possui suas próprias particularidades.

Um dos mais notórios problemas com qualquer modelo gerador é como lidar com palavras desconhecidas. Para gerar um texto, os algoritmos possuem um vocabulário do qual podem selecionar as palavras que comporão a peça final. Entretanto, existem casos nos quais um texto a ser resumido possua palavras fora do vocabulário do gerador. Dessa forma, será impossível, por mais que necessário, que tais palavras apareçam no resumo final.

Outro grande problema com modelos abstratos é a confusão de termos "semelhantes", como ilustrado no exemplo abaixo.

```
Texto Original:
Em uma partida acirrada, o time alemão derrota a seleção argentina por
2 a 0 neste último sábado. [...]

Resumo A:
Alemanha vence Argentina por 2 a 0.
```

```
Resumo B:
Alemanha vence Chile por 1 a 0.
```

As divergências entre os resumos A e B incluem os países que supostamente foram derrotados pela Alemanha e o placar do jogo. Apesar de parecerem estranhos para humanos, esses erros são decorrentes da proximidade nos casos de uso dos termos que foram substituídos. Essa confusão é um artefato do modo como as palavras são codificadas para serem usadas - *word embeddings*. Essa codificação, bem como sua utilização, será explorada em uma seção mais adiante do trabalho.

Mesmo não sendo imediatamente óbvio o motivo da troca de termos, é possível enxergar que há similaridades entre os termos confundidos. Tanto **Argentina** quanto **Chile** são países situados na América do Sul cuja língua principal é o espanhol. Em termos de aparições em textos, ambos os termos tem casos de uso parecidos - apesar de possuírem sentido semântico distintos. Analogamente, tanto **1** quanto **2** são números cardinais, apesar de expressarem quantidades diferentes.

1.5 PROBLEMAS DE IMPLEMENTAÇÃO

É importante ressaltar que além de problemas em níveis conceituais, também são presentes problemas de implementação. Muitos modelos propostos fazem completo sentido em um cenário hipotético ideal, entretanto trazê-los à realidade pode se provar uma tarefa não-trivial.

As dificuldades de implementação pode ter diferentes origens: mal condicionamento do problema, instabilidade numérica, dificuldade de tradução do modelo teórico para código com as abstrações de software do momento, limitações de hardware, entre outras.

Um exemplo desse fenômeno encontra-se em redes adversariais (SALIMANS et al., 2016). Essas redes se mostram altamente instáveis durante treino. Se não houver uma atenção especial na configuração dos híperparâmetros, o treino pode não convergir. Estabilização e convergência de treinos de redes geradoras adversariais são tópicos que têm atraído pesquisadores recentemente ((MIRZA; OSINDERO, 2014), (KARRAS et al., 2017), (MESCHEDER; GEIGER; NOWOZIN, 2018)).

Outro caso muito comum é o treino de RNN. Um problema grave que assolava esses modelos são os *exploding and vanishing gradients*. Propostas, como *gradient clipping*, se mostram efetivas para atacar esses problemas (PASCANU; MIKOLOV; BENGIO, 2013).

O modelo estudado a fundo nesse trabalho também sofre de problemas de implementação. Desde tradução do modelo matemático elaborado no artigo (SEE; LIU; MANNING, 2017), até instabilidade numérica, a arquitetura escolhida não vem sem desafios. Esse assunto será detalhado ao longo do trabalho e principalmente nas seções 3.2.7.1 e 5.2.

1.6 VISÃO EM ALTO NÍVEL

Para facilitar a compreensão do assunto tratado nesse trabalho, o código que será implementado e os tópicos que serão explicados nesse trabalho podem ser resumidos de acordo com o diagrama a seguir.

Figura 1 – Passo a passo em alto nível desse trabalho.

Cada uma dessas etapas será cuidadosamente estudada e explicada em sua devida seção.

1.7 OBJETIVOS DO TRABALHO

Tendo exposto uma visão em alto nível do projeto, é interessante destacar os objetivos do mesmo.

Em primeiro lugar, procura-se uma explicação detalhada e de fácil entendimento dos algoritmos, técnicas e estrategias utilizadas no trabalho. O objetivo principal é tornar o conhecimento desse ramo de pesquisa mais acessível a todos.

Com isso em mente, cada escolha será acompanhada de uma explicação e todos os algoritmos e arquiteturas de redes neurais serão explicadas até os componentes mais simples - sem assumir que o leitor tenha conhecimento prévio sobre o assunto.

Para ajudar a sedimentar o conhecimento teórico, segue em paralelo detalhes de implementação. Bem como serão destacados e discutidos problemas e dificuldades que uma implementação de uma rede neural complexa traz.

Será exposto ao longo desse trabalho que o modelo de estado da arte do artigo (SEE; LIU; MANNING, 2017) não é facilmente reprodutível. Há diversas pessoas que encontraram problemas com a implementação oficial e os próprios autores do artigo de referência não souberam responder a tais inquéritos. A implementação feita para esse trabalho não conseguiu reproduzir o exposto na referência principal.

Em segundo lugar, o trabalho tem o objetivo de testar uma arquitetura de estado da arte em um conjunto de dados diferente. A referência principal desse trabalho (SEE; LIU; MANNING, 2017) trabalha com o dataset *CNN/DailyMail* (HERMANN et al., 2015). O dataset é disponibilizado apenas mediante pagamento de um valor fora de alcance para esse projeto.

Foi então escolhido outro dataset para treinar o algoritmo - será detalhado na seção 4. Essa mudança traz consigo o fato de expor a arquitetura a um novo tipo de documentos a serem resumidos. Com a análise do comportamento do modelo em um cenário novo, seria possível avaliar melhor a adaptabilidade e capacidade de generalização que possui.

Além disso, o dataset escolhido é relativamente recente (2017) e não existem muitos trabalhos feitos em cima dele atualmente. É interessante descobrir que tipo de experimentos podem ser realizados e que características possui.

2 CONCEITOS BÁSICOS

Para a melhor compreensão dos assuntos tratados nesse trabalho, é necessário o entendimento de alguns conceitos básicos. Esses conceitos são fundamentais para que o resto desse documento faça sentido.

2.1 EMBEDDINGS

Esse trabalho trata de dados do tipo textual. Em sua representação no computador, esses dados são `strings`, ou seja, sequências de caracteres. Para muitos propósitos esse formato é suficiente, ou, mais do que isso, ideal. Entretanto, muitos modelos de aprendizado de máquina requerem entradas e saídas estritamente numéricas.

Tendo em vista que o modelo utilizado neste trabalho requer uma entrada numérica, devemos, de alguma forma, converter o tipo do dado original.

2.1.1 Embedding a nível de caractere

Talvez a forma mais imediata de se converter texto para números seja substituir caracteres por números. Afinal, na memória do computador, caracteres essencialmente já são números. Caracteres normalmente são apenas tratados como tal na visualização para o usuário ou em abstrações de linguagens de programação de mais alto nível.

O computador sabe como desenhar cada caractere de acordo com um *encoding*. Um encoding essencialmente diz que valor de memória (um número) corresponde a que glifo. Um exemplo seria o caractere `A`, cujo valor em ASCII é o número 65. A letra `B` é representada pelo número 66 e assim por diante.

Dessa forma, é natural de se pensar em utilizar o código ASCII para converter textos em sequências numéricas. Se esse for o padrão de conversão utilizado, o texto "Hello World" seria codificado da seguinte forma:

H	e	l	l	o		W	o	r	l	d
72	101	108	108	111	32	87	111	114	108	100

Analisando o modelo de codificação de texto em ASCII, é possível identificar que muitos números não são utilizados. Alguns desses números não são utilizados, pois não têm glifos correspondentes. Por exemplo, os primeiros 32 números da codificação ASCII não possuem representação visual e portanto não correspondem a nenhum caractere que possa ser utilizado em um texto. Além disso, há caracteres que não são utilizados em textos.

Outro problema em usar a codificação ASCII é no fato de ela ser muito simples. A codificação não suporta uma variedade de caracteres comumente utilizados em textos

(caracteres com acentuação, letras de outros alfabetos, alguns símbolos, entre outros). Poderia-se substituir a codificação ASCII por Unicode - que é uma codificação mais completa -, contudo existe uma saída mais fácil.

Para resolver o problema da codificação ideal, é necessário definir o que é esperado da solução. Nesse caso, deve ser possível mapear cada caractere no corpus utilizado neste trabalho a um número que o representa. Para resolver esse problema de forma elegante, basta criar uma lista de todos os caracteres utilizados nos textos e utilizar o índice como a representação numérica.

Dessa forma, haverá uma correspondência para todo o universo de caracteres utilizado e não haverá excesso.

Entretanto, existe um problema intrínseco nesse método. Ao associar cada caractere a um valor inteiro, implicitamente está sendo construída uma ordem. Mais do que uma ordem, cada caractere agora possui uma magnitude associada. Se `B` equivale a 2 e `J` equivale a 10, estabelece-se que `B` precede `J`, bem como que `J` é 5 vezes maior que `B`. Sob um ponto de vista semântico, essa relação não faz sentido. É um artefato indesejado desse modo de codificação utilizado para converter caracteres em números.

2.1.2 One-hot encoding

Uma forma de contornar esse problema é designando diferentes dimensões para representar cada caractere. Nessa codificação, conhecida como *one-hot encoding*, cada caractere é representado por um vetor multidimensional. Nesse vetor, cada dimensão corresponde a um caractere. Dessa forma, se apenas forem utilizadas as letras do alfabeto latino sem acentuação, a dimensão do vetor seria 26. Para codificar um caractere, basta colocar o valor correspondente à dimensão desse caractere como `1` e todas as outras como `0`, como na figura 2.

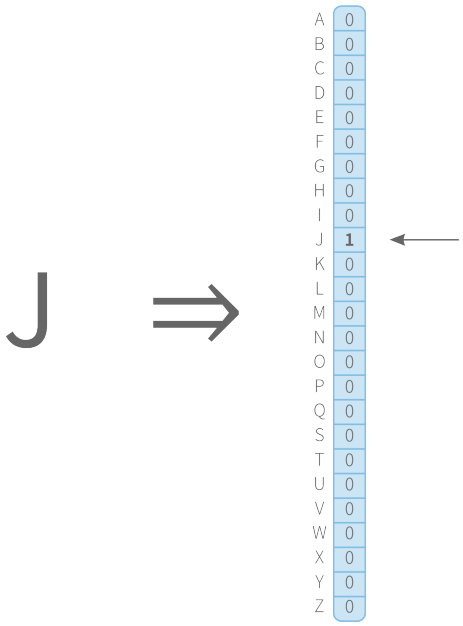
Tratando cada caractere como uma dimensão elimina-se o problema das relações indesejadas entre caracteres. De fato, elimina-se toda e qualquer relação entre os caracteres, afinal, como cada dimensão é ortogonal a todas as outras, existe completa independência linear entre elas.

2.1.3 Embedding a nível de palavras

Utilizando o one-hot encoding, é possível ter uma representação de textos sem introduzir relações indesejadas entre os caracteres. Nessa codificação, um modelo de sumarização receberia o input caractere por caractere para "ler" o texto integral. Analogamente, o texto gerado (resumo) também seria escrito caractere por caractere.

Isso introduz uma dificuldade adicional à tarefa de sumarização. Não só o modelo deverá ser capaz de interpretar um texto e condensá-lo, como também deverá associar conjuntos de caracteres com palavras. O modelo deverá ser capaz de compreender que a

Figura 2 – Representação da letra J em one-hot encoding



sequência de letras **L**, **Á**, **P**, **I** e **S** forma a palavra **LÁPIS** e que essa palavra, por sua vez, refere-se a um utensílio de escrita ubíquo.

Essa tarefa adicional de interpretar sequências de caracteres como palavras com significado torna-se uma dificuldade extra devido à codificação escolhida. Felizmente, esse passo adicional de interpretabilidade é facilmente eliminado trocando o significado de cada dimensão do vetor de embedding.

O grande fato a ser notado é que interpretar um texto como uma sequência de caracteres é granular demais. Quando humanos leem textos, não estão assimilando sequências de caracteres, mas sim de palavras. Palavras representam ideias. São essas ideias e suas relações que efetivamente compõem o conteúdo de um texto. Dessa forma, se um texto for passado ao modelo de sumarização palavra por palavra, o mesmo terá mais facilidade em cumprir sua tarefa principal de resumir.

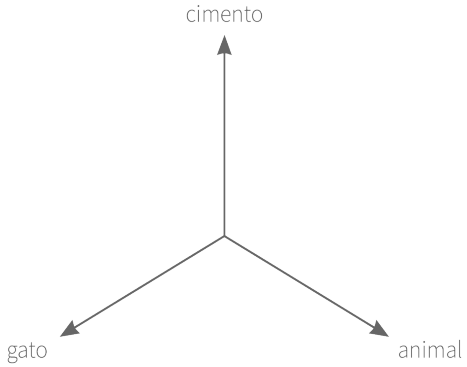
É possível utilizar a mesma estratégia de one-hot encoding para codificar palavras inteiras. Dessa vez, cada dimensão do vetor representa uma palavra. Uma consequência dessa decisão é o fato de que a dimensionalidade desse vetor one-hot cresce para o tamanho do vocabulário usado no corpus. Isso pode levar os vetores a serem de dimensões na ordem de dezenas de milhares - ou até mais. Dessa forma, é importante tomar cuidado quando trabalhando com vetores dessa magnitude. Afinal, será necessário mais memória para armazená-los, além de tornar os algoritmos mais lentos. Dependendo da complexidade do algoritmo escolhido, o tempo de execução pode se tornar um verdadeiro empecilho para a utilização dessa abordagem.

2.1.4 Word embeddings

A representação one-hot de palavras a princípio parece ser um modo bom de se representar um texto - resolvendo todos os problemas das abordagens anteriores. Contudo, essa codificação não vem sem suas próprias peculiaridades.

Como cada palavra é representada por uma dimensão ortogonal a todas as outras, não existe relação entre duas palavras do vocabulário. Isso reflete bem a relação de "**gato**" e "**cimento**", que semanticamente pouco têm a ver. Entretanto essa codificação implica em dizer que "**gato**" e "**animal**" nada têm a ver. Ou que "**gato**" está tão relacionado a "**animal**" quanto está a "**cimento**". É possível encontrar casos ainda mais críticos, como o fato de "**gato**" e "**gatos**" também não possuírem relações.

Figura 3 – Representação de "**gato**", "**cimento**" e "**animal**"



Um humano analisando a língua semanticamente percebe que há dependências e relações entre o significado de palavras. Suprimir um vocabulário de todas as suas relações semânticas é prejudicial para o modelo que o utilizará. Este, agora, será obrigado a internamente construir todas essas relações.

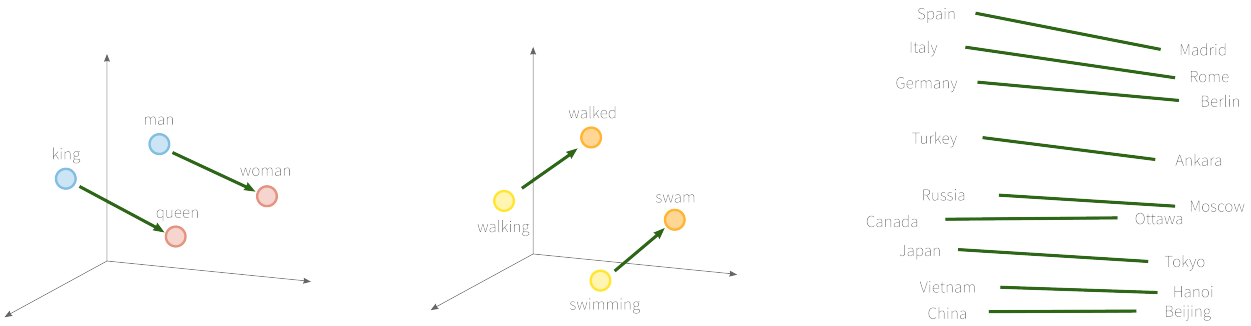
Tendo em vista essa questão, pesquisadores de NLP conceberam métodos de criação de representações de palavras em vetores densos (MIKOLOV et al., 2013b). As dimensões de um vetor, agora não representam mais uma palavra. Em vez disso, representam atributos que são compartilhados por palavras.

Isso se torna mais claro ao perceber que existem direções, nesse espaço vetorial, que representam desde conceitos gramaticais - como gênero, número e tempo verbal - até conceitos semânticos - como relações de capitais e países, relações de artistas e gêneros musicais, entre outras (MIKOLOV; YIH; ZWEIG, 2013).

Andar por esse espaço vetorial, agora, tem sentido semântico e gramatical. É uma representação muito mais rica para se usar em modelos de NLP. Inclusive, é possível realizar operações aritméticas com significados interpretativos entre vetores.

Um exemplo é o que é mostrado na figura 4. Ao subtrair o vetor da palavra "**man**" do vetor da palvra "**woman**", obtem-se a direção que representa o conceito de *gênero*. Fazendo o mesmo com os vetores "**king**" e "**queen**", obtem-se uma direção muito próxima

Figura 4 – A imagem mostra diferentes direções em espaço vetorial com correspondências semânticas.



à encontrada com o par "**woman-man**". Como mais um indício da existência de uma relação gramatical com o espaço vetorial, se o vetor representando o gênero obtido com o par "**woman-man**" for somado ao vetor da palavra "**king**", o resultado é o vetor da palavra "**queen**".

O mesmo procedimento pode ser realizado com tempos verbais, subtraindo palavras no gerúndio de palavras no pretérito perfeito para obter uma direção indicativa de tempos verbais. Mais do que isso, essas relações não estão restritas ao campo de regras gramaticais. Podem ser vistas no campo semântico.

Ao comparar o vetor diferença de diversos pares de país e capital, verifica-se que são todos muito próximos. Isso indica que existe uma direção responsável por representar o conceito de "ser capital de um país". O mesmo pode ser verificado com artistas e gêneros de música.

Em suma, a representação de textos como uma sequência de vetores densos (representando palavras) é uma escolha muito mais rica que as anteriores. Será esta a codificação usada para lidar com esse tipo de dado nesse trabalho.

2.2 MÉTRICAS DE AVALIAÇÃO

Para avaliar o desempenho de uma solução de sumarização é necessário definir um método de avaliação. Existem diversos aspectos de um resumo que podem ser avaliados: sua cobertura de conteúdos do texto original; o quão sintético é; entre outras medidas.

O conjunto de dados utilizado neste trabalho conta com resumos de referência - como será explorado na seção 4. Com essa disponibilidade, a alternativa de comparar o resumo gerado com sua referência se torna possível. Assumindo a referência como resumo ideal, ainda resta a dúvida de como efetivamente realizar a comparação entre os resumos.

Naturalmente, humanos poderiam ler ambos os resumos e decidir o quão próximos estão. Entretanto, com um volume muito grande de textos, essa solução começa a se tornar inviável. É nesse cenário que surgem os algoritmos de avaliação automáticos.

Desenvolvido com o propósito de avaliar a qualidade de sumarizações automáticas e traduções por máquinas, foi criado o **ROUGE** (LIN, 2004). Essa técnica é amplamente utilizada no ramo de NLP e é a que será usada nesse trabalho.

O algoritmo é baseado em palavras em comum em ambos os resumos. Quanto mais palavras em comum, mais próximos os textos. Para tornar o algoritmo mais robusto, esse procura não apenas a ocorrência de palavras soltas, mas também de conjuntos de palavras. Esses conjuntos de palavras são conhecidos como **n-grams** - onde **n** indica a quantidade de palavras em um conjunto. Dessa forma, se o objetivo for trabalhar com duplas de palavras em comum, seria relevante o **2-gram** ou **bigrama**.

O algoritmo é flexível para aceitar diferentes valores de **n**. Para cada escolha, um nome é dado. A métrica **ROUGE-1** (LIN, 2004) é referente ao algoritmo executado em cima de 1-grams (as palavras soltas).

O algoritmo, em si, calcula uma série de métricas. A primeira é a **revocação** (*recall* em inglês). Essa métrica calcula a porcentagem de palavras do resumo de referência que aparecem no resumo sendo avaliado. A revocação é interessante, pois captura se o conteúdo desejado aparece no resumo gerado. Entretanto, essa métrica sozinha não seria muito boa para avaliar um resumo.

Se algum modelo gerar um resumo contendo todas as palavras do vocabulário, esse resumo receberia nota máxima em revocação. Entretanto, intuitivamente sabe-se que esse resumo é ruim.

Para resolver esse problema utiliza-se em conjunto a métrica de **precisão**. A precisão mede a porcentagem de palavras do resumo gerado que foram utilizadas no resumo de referência. Nesse caso, se muitas palavras aparecerem apenas no resumo gerado, a precisão será baixa.

Com essas duas métricas, o resumo gerado é forçado a ser o mais próximo possível do de referência para que obtenha uma nota alta.

Para combinar os dois resultados, utiliza-se uma métrica chamada **F1 score**. A métrica é a média harmônica entre a precisão e a revocação.

$$F1 = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \tag{2.1}$$

Até então foi explicado apenas o ROUGE-1. Entretanto sua generalização para ROUGE-N é simples e diz respeito apenas aos n-grams utilizados. Para o ROUGE-2 a unidade mínima de comparação para calcular o F1 score é o bigrama (uma dupla de palavras); para o ROUGE-3, um trio de palavras; e dessa forma segue.

Existem casos especiais do ROUGE. O único relevante para esse trabalho é o **ROUGE-L**. O ROUGE-L (LIN, 2004) está interessado em capturar métricas relacionadas à maior subsequência comum entre o resumo gerado e o de referência.

Primeiro a maior subsequência comum (também conhecido como LCS - *Longest Common Sequence*) entre os dois resumos é calculada. O tamanho dessa sequência é utilizado

para calcular versões modificadas de revocação e precisão.

Para o ROUGE-L a revocação passa a ser o comprimento da maior subsequência divido pelo tamanho do resumo de referência. Analogamente a precisão passa a ser definida como a razão entre o comprimento da maior subsequência e o tamanho do resumo gerado.

Já a versão modificada do F1 score pode ser descrita pela equação 2.2:

$$F_{lcs} = \frac{(1 + \beta^2) R_{lcs} P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \tag{2.2}$$

Na equação 2.2 R_{lcs}, P_{lcs}, F_{lcs} são referentes à revocação, precisão e F1 score adaptados para o ROUGE-L. Já β é definido como P_{lcs}/R_{lcs}.

Nesse trabalho, serão usados ROUGE-1, ROUGE-2 e ROUGE-3 para avaliar a qualidade dos resumos.

3 MODELAGEM

Já exposta uma visão em alto nível do problema, agora, é necessário formalizar um pouco a definição do problema, bem como estabelecer o modelo que será usado.

3.1 DESCRIÇÃO DO PROBLEMA

Em essência a tarefa de sumarização consiste em receber um texto e comprimí-lo em um menor mantendo sua ideia central. Apenas com essa definição do problema, já é possível determinar algumas partes da modelagem.

Em primeiro lugar, sabe-se que **textos** são esperados como entrada e saída. Como já discutido na seção 2.1, devido ao modelo escolhido para esse trabalho, as palavras do texto deverão ser convertidas em vetores numéricos. Dessa forma, a entrada e saída do problema passa a ser sequências de vetores multidimensionais.

Em segundo lugar, é necessário definir critérios para qualificar um bom resumo. Nesse trabalho foram definidos três métricas quantitativas para tal fim: ROUGE-1, ROUGE-2, ROUGE-L. Bem como foram escolhidos três critérios qualitativos: conteúdo, tamanho e fluidez.

Para um resumo ser considerado como tal, ele deve preservar as ideias centrais do texto original. Afinal, se não o fizesse, não poderia ser considerado um resumo - ou talvez fosse considerado um resumo ruim.

Além disso, é desejável que o texto gerado seja significativamente menor que o original. Dessa forma, há efetivamente o ganho de velocidade e foco esperado da leitura de um resumo. Se o resumo for de um tamanho próximo ao texto original, esse item não será considerado cumprido.

O último critério, fluidez, diz respeito à sintaxe e gramática do texto. O algoritmo deve produzir um texto que seja legível. Frases incompletas ou inconsistentes sintática ou gramaticalmente serão consideradas ruins.

Note que ao dizer que o texto deve ser legível, não está subentendido que este siga a norma culta e todas as regras formais da língua. O texto pode ter caráter informal, mas deve refletir o que um humano fluente na língua escreveria.

3.2 MODELO DE SUMARIZAÇÃO

Dentre as possibilidades de soluções para o problema de sumarização automática, esse trabalho se aventura pelo ramo de redes neurais. Algoritmos baseados nesse paradigma têm se mostrado bastante capazes em tarefas desse tipo ((CAO et al., 2015), (LI et al.,

2017), (SEE; LIU; MANNING, 2017)). Em especial, destacam-se arquiteturas baseadas em Redes Neurais Recorrentes .

3.2.1 Redes Neurais Recorrentes

Em uma rede neural tradicional, o tamanho da entrada deve ser fixo. Devido à sua natureza, uma vez escolhida, não é possível modificar a quantidade de inputs. Entretanto, textos são naturalmente sequências de tamanho variável.

É possível contornar essa restrição usando alguma forma de preprocessamento. Podem ser utilizadas diferentes técnicas de conversão de dados de tamanho variável para dados de tamanho fixo. Entretanto, não importando a técnica escolhida, sempre há alguma perda ou uma distorção no resultado ao tentar fixar o tamanho de um dado sequencial.

Felizmente, existem tipos alternativos de redes neurais que podem ser usados. **Redes Neurais Recorrentes** (RNN) são modelos de redes neurais modificados para trabalhar com sequências de dados. As RNN foram pensadas justamente com o intuito de expandir a abrangência do modelo original para aceitar dados sequenciais tanto como entrada quanto como saída. Essa modificação elimina a necessidade de fixar um tamanho de entrada de forma que os dados possam ser utilizados em redes neurais.

Munidas de um mecanismo de memória, as RNN recebem dados sequênciais em passos. A ideia é ir passando a sequência de entrada elemento a elemento, de forma que o processamento leve em conta tanto o novo input quanto o conteúdo de passos anteriores. Analogamente, o output também é gerado passo a passo. Isso tudo possível devido à memória introduzida no modelo.

O mecanismo de retenção de dados é, na prática, apenas um vetor de números reais. Esse vetor, chamado de *vetor de estado*, é responsável por guardar informações que a rede deseja passar do momento atual para o próximo.

A cada passo da sequência, a unidade recorrente recebe dois vetores como entrada. O primeiro é o vetor que codifica um dado novo oriundo do problema em questão. Já o segundo é o vetor de estado do passo anterior da sequência.

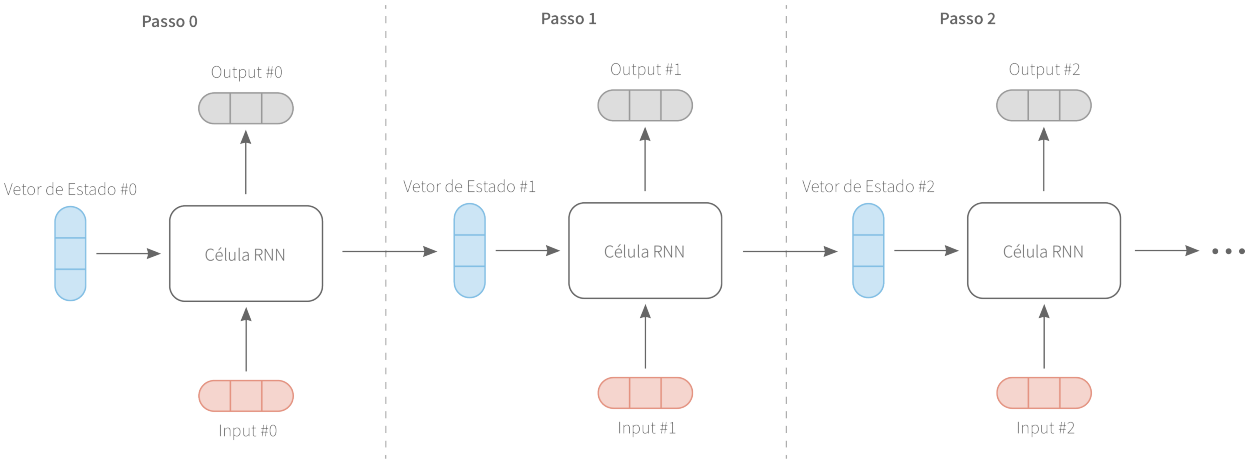
Similarmente, através de multiplicações de matrizes e funções de ativação, a unidade recorrente gera dois outputs: um novo vetor de estado e um vetor de saída. O vetor de saída diz respeito diretamente ao problema em questão e pode ser utilizado tanto para gerar a sequência de output ou ser ignorada dependendo da aplicação.

Já o vetor de estado contém informações sobre o processamento até então. Este é passado junto com a próxima entrada na sequência, de forma que informações relativas a entradas passadas possam ser armazenadas e recuperadas em passos futuros da sequência. Isso, em essência, caracteriza o funcionamento de uma RNN.

Mais formalmente o cálculo pode ser descrito de acordo com as equações 3.1 e 3.2.

$$\text{out}_t = \tanh(V_o[h_{t-1}; x_t] + b_o) \tag{3.1}$$

Figura 5 – Uma célula RNN passo a passo.



$$h_t = \tanh(V_s[h_{t-1}; x_t] + b_s) \tag{3.2}$$

O termo out_t representa a saída da rede no passo t. Já o termo h_t representa o vetor de estado do passo t. As matrizes V_o e V_s, bem como os vetores b_o e b_s são parâmetros a serem aprendidos durante o treino da rede. A operação $[\ \cdot\ ;\ \cdot\ ]$ indica a concatenação de dois vetores coluna. Por fim, a tangente hiperbólica é aplicada elemento a elemento no vetor em questão.

3.2.2 Sumarização Abstrata com RNN

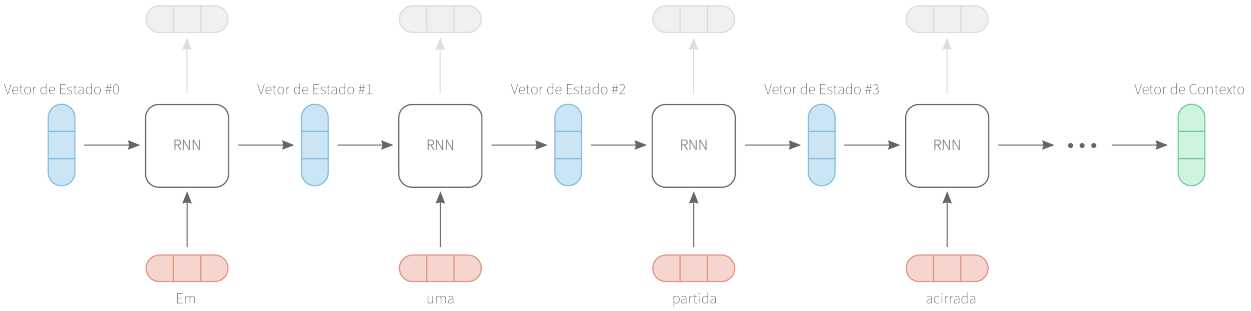
De acordo com o fluxo de uma solução de sumarização abstrata, primeiro deve-se obter uma representação do texto original. No paradigma de RNN, isso é feito passando o texto pela rede de modo a gerar um vetor de estado que armazene o conteúdo relevante sobre o material a ser resumido.

Nesse regime, as representações de cada palavra do texto original são passadas uma a uma como entradas da rede. Como o intuito dessa etapa é apenas de condensar informações no vetor de estado final, os outputs intermediários da rede são ignorados.

A ideia é que, com essas operações, o vetor de estado - inicialmente desprovido de conteúdo relevante - acumule as informações significantes a serem postas no resumo. Esse vetor de estado resultante da primeira etapa é conhecido como *vetor de contexto*. É o responsável por prover o conteúdo na etapa de geração de texto.

Para a etapa inicial, utiliza-se uma rede recorrente denominada *encoder*. Essa é responsável por pegar as palavras de entrada e converter em um vetor de contexto de acordo com a Figura 6. A rede codifica uma sequência de tamanho arbitrário em um único vetor de tamanho fixo. Note que aqui converte-se uma sequência de tamanho variável em um vetor de tamanho fixo. Como dito na seção 3.2.1, isso acarreta em limitações na representação. Essa situação será melhor explorada mais a frente, na seção 3.2.6.

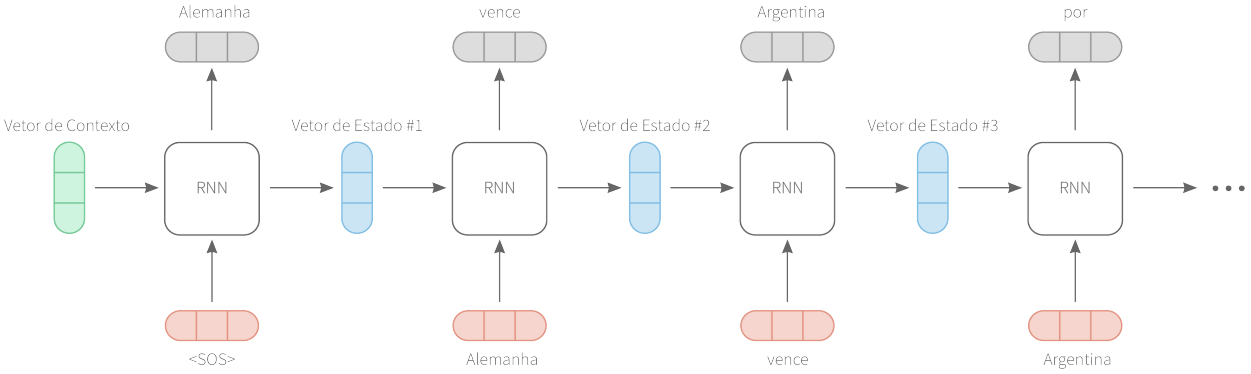
Figura 6 – Gerando o vetor de contexto do texto original



Nessa figura, passam-se as palavras do texto original na rede recorrente, uma de cada vez. Como os outputs intermediários não são interessantes nesse momento, estão apagados para indicar que são ignorados. Ao fim dessa etapa, está representado em verde o vetor de contexto.

Uma vez com o conteúdo do texto salvo nesse vetor de memória, pode-se passar para a etapa de geração de texto. Dessa vez, inicia-se o processo com o vetor de contexto. Este será introduzido como o estado inicial de memória da célula recorrente.

Figura 7 – Gerando o resumo a partir do vetor de contexto



A figura mostra a geração do resumo palavra por palavra, utilizando o vetor de contexto como memória inicial.

Para gerar o resumo, utiliza-se uma outra rede neural, também recorrente. Essa, por sua vez, tem o papel de converter um vetor de contexto - com informações condensadas - em uma sequência de palavras. É, portanto, conhecida como *decoder*, ou decodificador. A mesma estrategia, de trabalhar com o texto palavra por palavra, é empregada aqui (Figura 7).

A questão fundamental nesta etapa é condicionar a rede a escrever o texto do resumo baseado no vetor de contexto gerado anteriormente. Isso é feito introduzindo o vetor de contexto como o vetor de estado inicial da rede. Lembrando que o vetor de estado representa a memória da rede recorrente. Pode-se interpretar essa estratégia como a inserção de uma memória específica sobre a qual a rede deve atuar.

Com o vetor de contexto inserido como estado inicial, passa-se à geração do resumo. As palavras são escolhidas a partir da sequência de saída da unidade recorrente. Deve-se ter em mente que para gerar uma saída em uma unidade recorrente, é necessária uma entrada. Portanto, cabe a dúvida: o que seriam as entradas dessa etapa?

Devido à tarefa a qual esta rede deve desempenhar, foi sugerido que seria um bom método utilizar a palavra anterior como entrada para a geração da palavra atual. Dessa forma, como no exemplo da Figura 7, se a frase até o momento for "**Alemanha vence**", a entrada do próximo passo da rede é "**vence**". Seguindo o exemplo, a saída da rede nesse passo é a palavra "**Argentina**", que por sua vez servirá de entrada ao próximo passo da sequência.

De fato, empiricamente essa estratégia se mostrou eficaz (SUTSKEVER; VINYALS; LE, 2014). Ao receber a palavra anterior como entrada, a rede tem acesso a um demarcador que mostra em que parte da frase está. Isso facilita a confecção de textos inteligíveis. Mais ainda, devido ao modo como uma palavra é escolhida a partir do output numérico da rede - o que será explorado mais à frente -, a rede também recebe uma confirmação de qual foi a palavra inserida no texto. Essa confirmação é crucial para o bom funcionamento do modelo.

Entretanto, essa abordagem introduz um leve problema. Qual seria a primeira entrada, uma vez que nenhuma palavra foi gerada ainda?

Para resolver essa situação, utiliza-se uma palavra especial de início. Palavra, essa, normalmente representada por `<SOS>` (sigla de *Start of Sentence*). Sob a ótica da rede, é vista apenas como mais um elemento do vocabulário. O que há especial nessa palavra é o modo como é empregada nos textos. Artificialmente, adiciona-se essa palavra ao início de todo exemplo de texto. Durante a fase de treino, qualquer texto recebido pela rede será iniciado por esta palavra.

Dessa forma, será aprendido que todo texto começa com `<SOS>`. Assim, quando passa-se à fase de geração de texto, a palavra de entrada inicial será `<SOS>` e a rede estará preparada para continuar o resumo a partir de então.

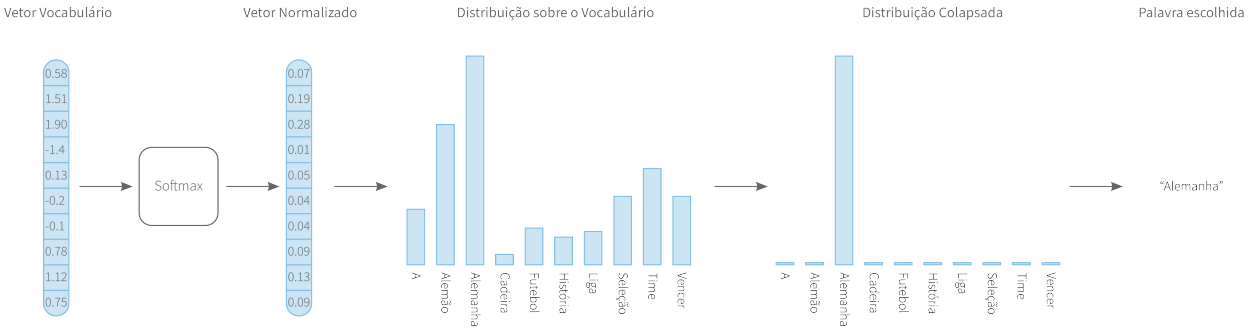
Similarmente, o termo `<EOS>` (*End of Sentence*) é utilizado para indicar a finalização de textos. Durante a geração do resumo, entende-se que quando aparecer `<EOS>`, o texto acaba. É o artifício utilizado pela rede para indicar que não se deve mais continuar gerando palavras.

Com isso, resta ainda um detalhe sobre a modelagem dessa solução. Não está claro como um output numérico da unidade recorrente é convertido em uma palavra.

O modo mais tradicional de se realizar essa tarefa é interpretando o output da unidade recorrente como uma distribuição de probabilidade sobre um vocabulário.

Define-se que cada posição do vetor de saída corresponda a uma palavra do vocabulário. Para simplificar, supõe-se que as palavras estão ordenadas alfabeticamente - de modo que a primeira palavra seja, por exemplo, "**a**" e a última "**zoológico**".

Figura 8 – Escolhendo uma palavra a partir do output da rede



O desejado é que cada palavra do vocabulário tenha uma probabilidade de ser escolhida como a próxima palavra da sequência de saída. Essa probabilidade seria computada de acordo com o vetor de saída a cada passo da rede recorrente. Contudo, é necessário algum mecanismo para transformar os números reais irrestritos oriundos da saída bruta em probabilidades.

Com este intuito, é introduzido o *softmax*. Essa função garante tanto que cada valor esteja entre 0 e 1, bem como que o somatório de todos esses valores resulte em 1. Pode ser pensada como a função de ativação da última camada da rede, cujo objetivo é normalizar os resultados em probabilidades. Sua fórmula é dada por:

$$\text{softmax}(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_i e^{x_i}} \tag{3.3}$$

Entende-se que na equação 3.3, $\mathbf{x}$ é um vetor e que a exponenciação aplicada é feita elemento a elemento. O símbolo x_i refere-se ao *i-ésimo* elemento do vetor $\mathbf{x}$.

Analisando em detalhes cada parte do *softmax* é possível tirar sentido das operações. Todas as entradas do vetor são exponenciadas como uma forma de eliminar valores negativos. Esse modo de lidar com números negativos é interessante se considerado o intuito final da operação: gerar probabilidades. Se entradas com valores positivos expressivos correspondem a probabilidades altas, então, as com valores negativos devem possuir uma probabilidade mínima como resultado. A exponenciação de um valor negativo cumpre esse papel, deixando o valor muito próximo de zero.

A segunda parte é responsável por normalizar os valores exponenciados. Tendo em mente que as probabilidades devem ser proporcionais aos valores brutos, basta verificar o valor de cada entrada sobre a soma total dos valores. Pelo passo anterior é garantido que todos os valores são positivos, portanto, essa operação reflete a proporcionalidade esperada do *softmax*.

O processo todo da geração de distribuição de probabilidades sobre o vocabulário pode ser condensado na equação 3.5.

$$p_{vocab} = \text{softmax}(V\text{out}_t + b) \tag{3.4}$$

$$p(w) = p_{vocab}(w) \tag{3.5}$$

Onde V e b são parâmetros de uma camada de rede neural. O vetor p_{vocab} representa a distribuição sobre o vocabulário - na qual a i-ésima entrada do vetor corresponde à i-ésima palavra no vocabulário. A probabilidade final relativa à palavra w é descrita por $p(w)$.

Munido dessa distribuição sobre o vocabulário de saída, resta decidir como usá-la para efetivamente escolher uma palavra a ser anexada ao fim do resumo atual.

O modo mais simples de fazer essa escolha é tomando como próxima palavra aquela de maior probabilidade de acordo com a saída da rede. Intuitivamente, essa escolha faz sentido. Afinal trata-se da palavra que a rede mais acredita que deva vir em seguida. É possível gerar resumos decentes usando essa abordagem, entretanto ela é um tanto problemática.

Uma consequência indesejada de sempre escolher a palavra mais provável é o fato do texto se tornar raso e repetitivo. As probabilidades geradas pela rede refletem a ocorrência de palavras vistas durante a fase de treino. Dessa forma, termos mais frequentes na língua de escolha acabam tomando conta das frases geradas. Inadvertidamente, o texto ficará repleto de preposições, artigos e substantivos mais populares da língua. Além disso, palavras que talvez fossem mais adequadas, mas não utilizadas muito frequentemente, acabam não sendo escolhidas nesse regime. É comum, também, que haja repetições na própria frase, gerando ciclos - isso será explorado na seção 3.2.7.2 desse trabalho.

Uma solução melhor seria se aproveitar da distribuição para sortear a palavra baseado nas probabilidades. Dessa forma ainda respeitam-se as prioridades entre as escolhas e ao mesmo tempo há espaço para diversidade. Empiricamente as frases geradas usando esse método são mais realistas e naturais.

Utilizando o método do sorteio baseado nas probabilidades ainda é possível controlar o grau de "criatividade" da rede por meio de um artifício chamado *temperatura*.

$$softmax(\mathbf{x}, T) = \frac{e^{\frac{\mathbf{x}}{T}}}{\sum_i e^{\frac{x_i}{T}}} \tag{3.6}$$

A *temperatura*, representada por T na equação 3.6, é um escalar usado para controlar a magnitude dos valores reais oriundos da rede antes de serem passados pelo softmax.

Escolhendo um valor entre 0 e 1 para a temperatura, a magnitude dos valores aumenta, causando a disparidade entre probabilidades a ser maior quando computado o softmax. Em contrapartida, se o valor da temperatura for muito grande, as probabilidades se tornam mais uniformes.

Em termos semânticos, uma temperatura alta indica um aumento na diversidade dos textos. Ao passo que uma temperatura baixa implica em um conservadorismo maior nas escolhas. É importante lembrar que uma temperatura elevada não só traz mais variedades,

mas também potenciais erros. Afinal palavras que não cabem em certas posições de frases têm chances maiores de acabar sendo selecionadas.

Mesmo com o controle da temperatura das probabilidades, ainda é possível criar frases ruins usando a abordagem de sorteio. A escolha de uma palavra errada pode tornar uma frase terrível. Em uma tentativa de mitigar esse problema, aparece mais um método de formação de textos: Beam Search.

A ideia é não escolher apenas uma palavra por vez, mas s palavras ao mesmo tempo - onde s é o tamanho da busca.

No primeiro passo da geração do texto, escolhem-se as s palavras mais prováveis. Para o segundo passo, passam-se as s palavras selecionadas em paralelo de forma a gerar s novos cenários concorrentes.

Para cada cenário, uma distribuição para a próxima palavra é gerada. De cada distribuição, são sorteados s termos para serem candidatos de próxima palavra de sua respectiva frase. Tem-se então s^2 frases formadas. O próximo passo é ordená-las com base em suas probabilidades e armazenar apenas as s mais prováveis. Esse processo é repetido até o fim dos textos. A frase com maior probabilidade de ser gerada é tida como a escolhida.

Essa abordagem é mais indulgente, pois erros (escolhas de palavras ruins) são recuperáveis. Afinal, caso alguma palavra fora de lugar seja sorteada e não haja correção para esta frase, sua pontuação geral (probabilidade da frase ser formada) será baixa e a frase será eliminada. Ao mesmo tempo, mantém-se o espaço para a variedade desejada de uma abordagem não determinística.

3.2.3 Função de Custo

Existem diferentes modos de se treinar uma rede neural. Métodos baseados em otimização por gradiente são muito populares nesse meio e é o modo como a rede será treinada nesse trabalho.

De modo que seja possível utilizar métodos de gradiente, é necessário primeiro definir uma *função de custo*. A função de custo é o que indica numericamente o quão longe do objetivo a rede está. Através de métodos de gradiente a rede ajusta seus parâmetros de forma a minimizar a função de custo - e consequentemente melhorar seu desempenho na tarefa em questão.

Outro fator importante a ser considerado é o fato de que a computação toda - desde a entrada à saída da rede e o cálculo da função de custo - deve ser diferenciável. Isso se dá devido à necessidade de se calcular o gradiente da função de custo em relação a cada parâmetro ajustável da rede.

Para cada passo da RNN a função de custo será descrita de acordo com a seguinte equação:

$$\text{loss}_t = -\log P(w_t^*) \tag{3.7}$$

Para obter o custo total, basta fazer uma soma normalizada dos custos parciais - como na equação 3.8 (SEE; LIU; MANNING, 2017).

$$\text{loss} = \frac{1}{T_y} \sum_{t=0}^{T_y} \text{loss}_t \tag{3.8}$$

Nas equações acima, w_t^* refere-se à palavra correta no passo t de acordo com o resumo de referência e T_y é o comprimento do resumo de referência.

Nesse caso, a função de custo é simples de interpretar. Analisando a equação 3.7, tem-se que caso a probabilidade da palavra desejada seja 1 (o melhor cenário), o custo será 0. No extremo oposto, com a probabilidade tendendo a zero, o custo tenderá ao infinito.

Para os valores intermediários, quanto mais próximo de 1, menor será o custo. Por se tratar de uma função monotônica para valores reais positivos, quanto mais próximo de zero, maior o custo.

Como o custo total é proporcional à soma dos custos, quanto mais palavras corretas com probabilidades baixas houver, maior será o custo. Inversamente, se as palavras corretas tiverem probabilidades altas de serem escolhidas, o custo será reduzido.

3.2.4 Modificações em Redes Recorrentes

A modelagem até então parte do princípio que as unidades recorrentes da rede sejam suficientemente poderosas para tanto armazenar o conteúdo original do texto em um vetor de contexto, quanto gerar resumos inteligíveis e coerentes com base nessa representação condensada.

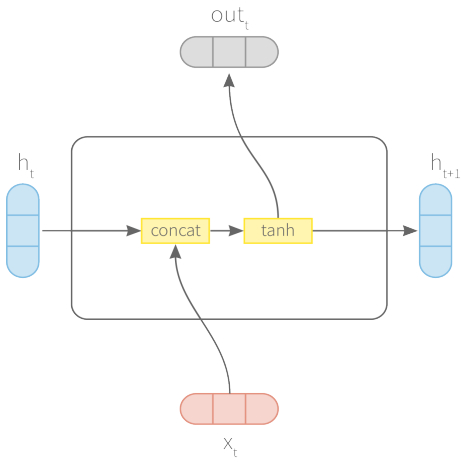
Entretanto, a unidade padrão de uma rede recorrente é um tanto fraca. Exigir tarefas com o nível de complexidade necessário para gerar resumos de qualidade, muitas vezes se prova demais para sua capacidade (TALATHI; VARTAK, 2015). Com o intuito de tornar redes recorrentes mais poderosas, novos modelos de unidades recorrentes foram sugeridos ((HOCHREITER; SCHMIDHUBER, 1997), (CHO et al., 2014), (GRAVES; SCHMIDHUBER, 2005)).

Como é possível verificar na figura 9, em uma célula convencional de RNN, o vetor de estado anterior é concatenado ao input da vez e o resultado passa por uma camada densa de rede neural. Isso permite a correlação temporal entre passos distintos da sequência de entrada. Entretanto não é fácil de manter dependências temporais de longo prazo com esse esquema.

Com o intuito de melhorar a perseverança de informações de longo prazo, inventaram a *Long Short Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997). A LSTM é uma alternativa à célula original de uma RNN com foco em memórias distantes.

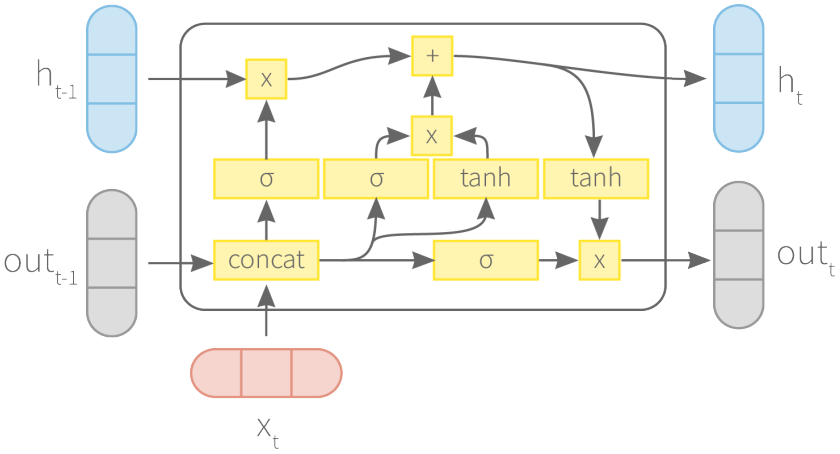
Esse tipo de célula recorrente é capaz de realizar tal tarefa, pois separa em diferentes camadas tarefas específicas no tratamento do vetor de estado. A LSTM possui "*portões*",

Figura 9 – Esquema de uma RNN tradicional



cada um com um propósito específico para lidar com o fluxo de dados passando pela célula.

Figura 10 – Esquema de uma célula LSTM



Nessa imagem, o símbolo σ representa a função *sigmoide*; o termo *concat* representa a operação de concatenação; o termo *tanh* representa a tangente hiperbólica; finalmente os símbolos x e + representam multiplicação e soma elemento a elemento respectivamente.

O primeiro portão é o *portão do esquecimento*. Esse portão é responsável por eliminar do vetor de estado informações que já não são mais necessárias. O portão recebe como entrada o vetor de saída do último passo concatenado ao vetor de entrada do passo atual. A esse novo vetor é aplicada por uma camada linear com sigmoide. A operação completa pode ser vista na equação 3.9, na qual f_t representa o portão do esquecimento.

$$f_t = \sigma(W_f[\text{out}_{t-1}; x_t] + b_f) \tag{3.9}$$

$$i_t = \sigma(W_i[\text{out}_{t-1}; x_t] + b_i) \tag{3.10}$$

$$\widetilde{C}_t = \tanh(W_C[\text{out}_{t-1}; x_t] + b_C) \tag{3.11}$$

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{3.12}$$

$$o_t = \sigma(W_o[\text{out}_{t-1}; x_t] + b_o) \tag{3.13}$$

$$\text{out}_t = o_t * \tanh(\widetilde{C}_t) \tag{3.14}$$

Nas equações 3.9 a 3.14, a função sigmoide é representada pela letra σ e quando aplicada a um vetor é feita de forma elemento a elemento. As matrizes W_f, W_i, W_C e W_o, bem como os vetores b_f, b_i, b_C e b_o são parâmetros aprendidos durante a fase de treino.

Em seguida vem o *portão de entrada*. Esse portão é responsável por introduzir ao vetor de estado novas informações a serem guardadas.

Da mesma forma como no portão do esquecimento, a entrada desse portão é a concatenação da saída do último passo com a entrada do passo atual. Esse vetor é então passado por uma camada com sigmoide. Seu resultado é representado por i_t na equação 3.10.

As novas informações candidatas a serem guardadas no vetor de estado estão representadas por $\widetilde{C}_t$ na equação 3.11. São calculadas de forma parecida com o portão de entrada, com a distinção de trocar a função de ativação de sigmoide por tangente hiperbólica.

Tendo calculado o portão do esquecimento, o portão de entrada e o candidato a novo vetor de estado, passa-se agora ao cálculo efetivo no novo vetor de estado - representado na equação 3.12 como C_t.

O papel do portão de esquecimento é descrito na primeira parte da equação 3.12 com a multiplicação elemento a elemento entre o portão do esquecimento e o vetor de estado da etapa anterior.

A ideia é que ao multiplicar o vetor oriundo desse portão (cujos elementos estão entre 0 e 1 devido ao sigmoide) com o vetor de estado, os valores multiplicados por 0 serão apagados e os multiplicados por 1 mantidos. Dessa forma, é possível esquecer seletivamente partes do vetor de estado e manter outras.

Analogamente, o portão de entrada é utilizado para multiplicar elemento a elemento o vetor de informações candidatas a serem salvas (segunda parte da soma da equação 3.12). Dessa forma, também é possível aplicar um filtro seletivo às informações prestes a entrar.

Finalmente, é possível calcular o novo vetor de estado utilizando as versões filtradas do vetor de estado anterior e do candidato atual (equação 3.12).

Tendo atualizado o vetor de estado, resta apenas gerar a saída da unidade recorrente.

Novamente uma camada sigmoide é aplicada à saída do passo anterior concatenada à entrada atual para calcular o portão de saída. O portão está representado por o_t na

equação 3.13. Dessa vez, será multiplicado ao novo vetor de estado calculado após passar por uma camada de tangente hiperbólica (equação 3.14). O resultado dessa operação, representado por out_t, é a saída do passo atual da LSTM.

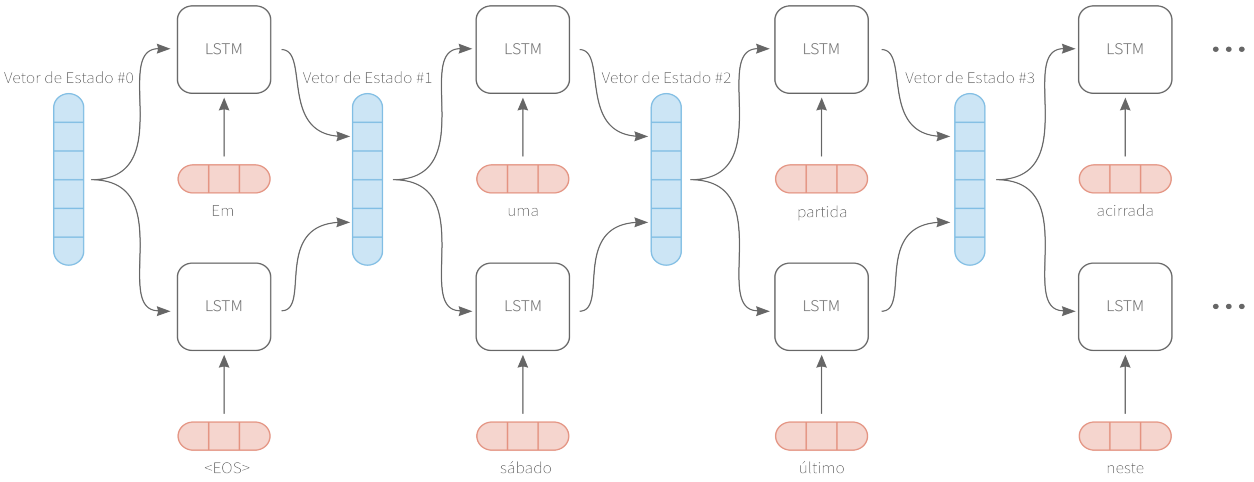
É possível verificar que uma unidade de LSTM tem um controle bem mais granular sobre o fluxo de informações dentro dela do que uma unidade recorrente convencional. Na prática isso também se concretiza, sendo a LSTM um avanço significativo quando comparada à RNN tradicional (HOCHREITER; SCHMIDHUBER, 1997).

Existem variantes dessa célula, algumas criando novas conexões entre os portões e algumas os simplificando. Uma das variantes mais populares é a *Gated Recurrent Unit* (GRU) (CHO et al., 2014), que visa simplificar o fluxo interno da unidade fundindo o portão de esquecimento com o de entrada em um único *portão de atualização*.

3.2.5 Redes Recorrentes Bidirecionais

Outra modificação muito popular no paradigma de RNN não se dá na unidade recorrente, mas na arquitetura da rede. Para alguns problemas de natureza sequencial é necessário um olhar atemporal para a sequência inteira de entrada. Informações relevantes podem estar mais a frente na sequência, prejudicando o desempenho da rede devido à ordem na qual as palavras são apresentadas.

Figura 11 – Esquema de uma rede LSTM bidirecional



A rede é composta por duas unidades de LSTM com vetores de estado compartilhados. As saídas a cada passo foram omitidas para fins de clareza.

Com o intuito de mitigar as dependências temporais da rede, foram criadas redes recorrentes *bidirecionais* (SCHUSTER; PALIWAL, 1997). A arquitetura bidirecional, simplesmente acrescenta uma segunda rede recorrente que recebe a entrada de forma inversa à primeira. Ambas as redes têm seus vetores de estado conectados, de forma que informações possam fluir em ambos os sentidos. Dessa forma, a rede tem uma visão mais

completa da sequência como um todo e obtém resultados melhores do que uma RNN tradicional.

Essa arquitetura de rede foi originalmente concebida para atacar o problema de reconhecimento de fala usando redes recorrentes. Nesse cenário, têm-se o áudio completo a ser transcrito a priori, e portanto uma abordagem atemporal pode ser empregada. Entretanto, nem todos os problemas sequenciais podem tirar proveito dessa arquitetura.

Felizmente, para o problema de sumarização é possível obter o texto completo a priori, e portanto esse tipo de arquitetura será utilizado neste trabalho.

3.2.6 Mecanismos de Atenção

Mesmo contando com a melhoria provida pela bidirecionalidade da nova arquitetura, ainda há melhorias a serem feitas no modelo. Uma restrição significativa já destacada no modelo atual é o fato de armazenar todo o conteúdo a ser resumido em um único vetor de contexto. Utilizando *mecanismos de atenção* é possível contornar essa situação.

Mecanismos de atenção foram criados para melhorar a performance de redes recorrentes em tarefas de tradução de máquina (BAHDANAU; CHO; BENGIO, 2014). Tomando como inspiração o modo como seres humanos traduzem textos, o mecanismo de atenção permite à rede olhar diversas vezes ao material de entrada com enfoques diferentes dependendo de certos parâmetros.

A ideia é que trechos diferentes da sequência de entrada são relevantes em momentos diferentes da geração da saída.

O mecanismo traz melhorias não só para problemas de tradução, como também o problema tratado nesse trabalho: sumarização. Tentar comprimir todo o conteúdo a ser resumido em um único vetor de contexto é muito restritivo. É uma exigência muito grande para uma representação. A utilização dessa abordagem de diferentes enfoques permite muito mais fidelidade na sequência de saída, afinal o material original está disponível na íntegra para ser consultado com focos em diferentes partes.

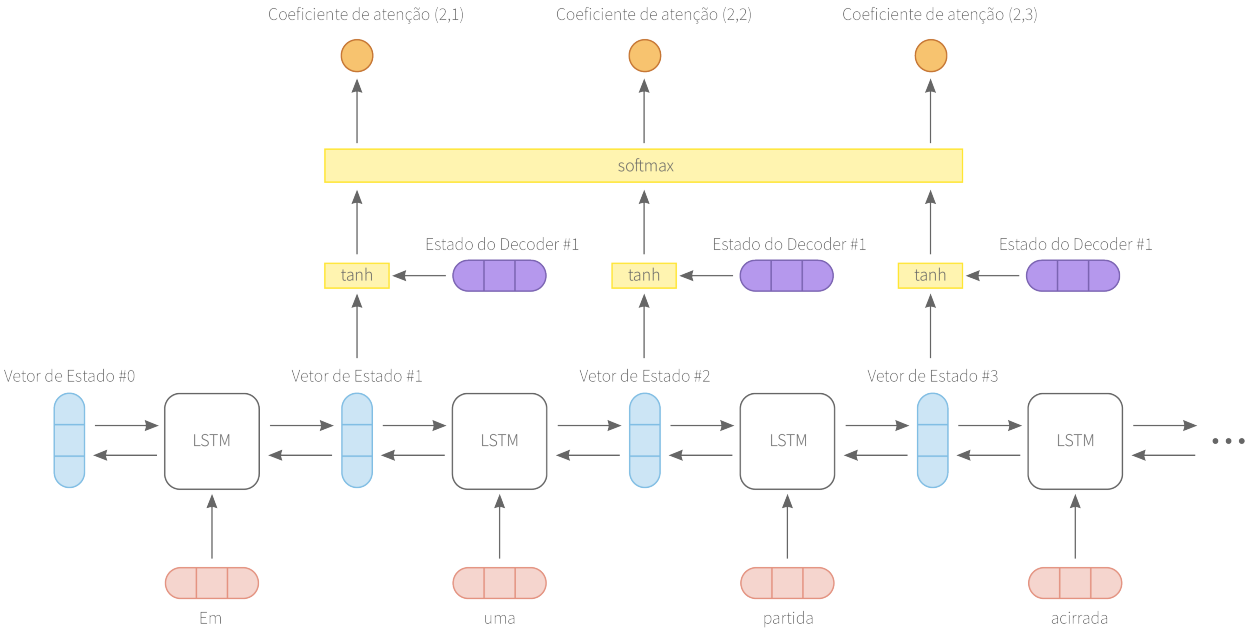
Para conseguir essa funcionalidade de direcionamento de foco, as redes com atenção utilizam vários vetores de contexto - um para cada passo da geração da saída.

Seguindo o paradigma, para cada passo da geração de saída, será alimentado um vetor de contexto com o foco especificamente criado para esse momento.

Primeiro são calculados os chamados *coeficientes de atenção* - os quais serão usados para focar em diferentes partes da sequência de entrada (figura 12). Para essa tarefa, utilizam-se o último vetor de estado do *decoder* (representado em roxo na figura 12) disponível e o vetor de estado do *encoder* ao qual se quer calcular o coeficiente. Ambos são concatenados e passados por uma camada de rede neural (eq. 3.15).

O resultado é um peso não normalizado de atenção. Para calcular o coeficiente final,

Figura 12 – Cálculo de coeficientes de atenção



A imagem mostra como são calculados os coeficientes de atenção para cada vetor de estado do *encoder*. A rede mostrada aqui é uma representação condensada de uma Bi-LSTM (GRAVES; SCHMIDHUBER, 2005). A notação (t, i) dos coeficientes de atenção indicam o coeficiente para o vetor de estado do *encoder* no passo i dado o contexto do vetor de estado do *decoder* no passo t.

passam-se todos os pesos intermediários por um *softmax* (eq. 3.16).

$$e_{t,i} = {v_a}^T \tanh(W_a[s_{t-1}; h_i] + b_a) \tag{3.15}$$

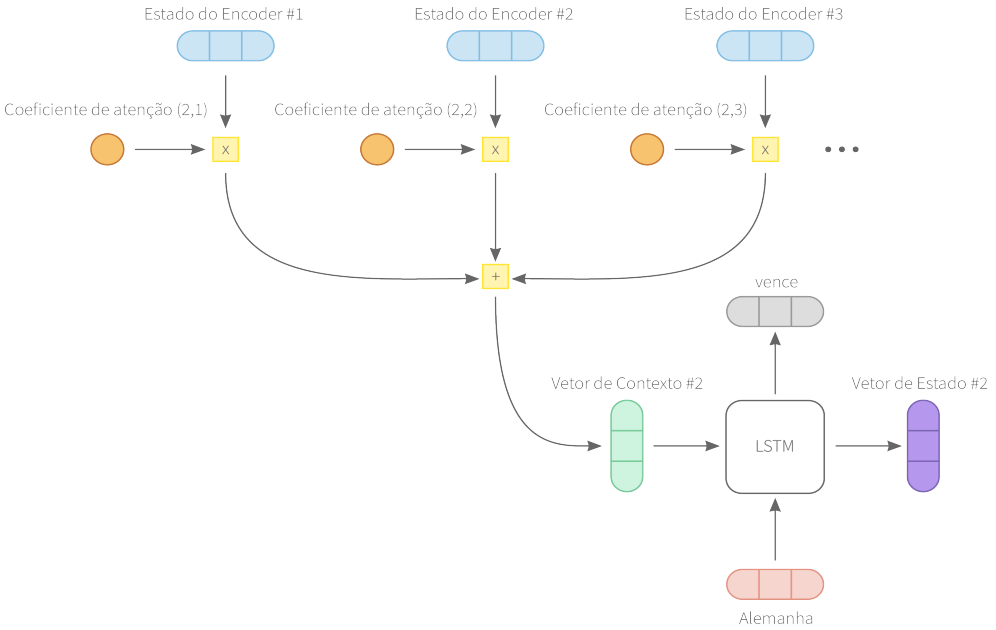
$$\alpha_{t,i} = \frac{e_{t,i}}{\sum_{i=0}^{T_x} e_{t,i}} \tag{3.16}$$

$$c_t = \sum_{i=0}^{T_x} \alpha_{t,i} h_i \tag{3.17}$$

Nessas equações c_t refere-se ao vetor de contexto com atenção no passo t do *decoder*; T_x refere-se ao tamanho da sequência de entrada; h_i refere-se ao vetor de estado do *encoder* no passo i; s_{t-1} refere-se ao vetor de estado do *decoder* no passo $t-1$; $\alpha_{t,i}$ refere-se ao coeficiente de atenção dado ao vetor h_i no momento t da sequência de saída; v_a e W_A e b_a são parâmetros a serem aprendidos durante a fase de treino.

Em sequência é feita uma soma ponderada dos vetores de estado intermediários do *encoder* como explicitado na equação 3.17 e ilustrado na figura 13. O peso que multiplica cada vetor de estado do *encoder* pode ser pensado como o nível de atenção dado ao mesmo.

Figura 13 – Cálculo de coeficientes de atenção



A imagem mostra um passo do *decoder*, utilizando os coeficientes de atenção e vetores de estado do *encoder* para gerar o vetor de contexto apropriado. Assim como na imagem anterior, todos os vetores de estado do *encoder* são utilizados e foram suprimidos por brevidade.

Isso possibilita que vetores de estado guardem apenas informações relevantes localmente. O que, combinado com o foco dos coeficientes de atenção, permite uma granularidade maior de controle sobre que informações são alimentadas em que passo temporal.

3.2.7 Pointer-Generator

Restam apenas mais duas modificações no modelo para chegar à arquitetura final utilizada nesse projeto. Essas modificações visam mitigar os efeitos comuns em algoritmos de sumarização abstrata: lidar com palavras de baixa frequência ou fora do vocabulário e ciclos ou repetições nas frases geradas.

A arquitetura utilizada neste trabalho é conhecida como Pointer-Generator (SEE; LIU; MANNING, 2017). Ela toma uma abordagem híbrida entre a sumarização extrativa e a abstrata. Munindo uma rede geradora de um mecanismo de cópia, é possível obter uma flexibilidade maior.

3.2.7.1 Mecanismo de ponteiro

A primeira modificação acrescenta à rede a habilidade de copiar palavras do texto original para o resumo (VINYALS; FORTUNATO; JAITLY, 2015).

A cada passo, a rede pode tanto gerar uma palavra a partir do vocabulário - como nos modelos até então - quanto indicar um termo no documento original a ser posto no

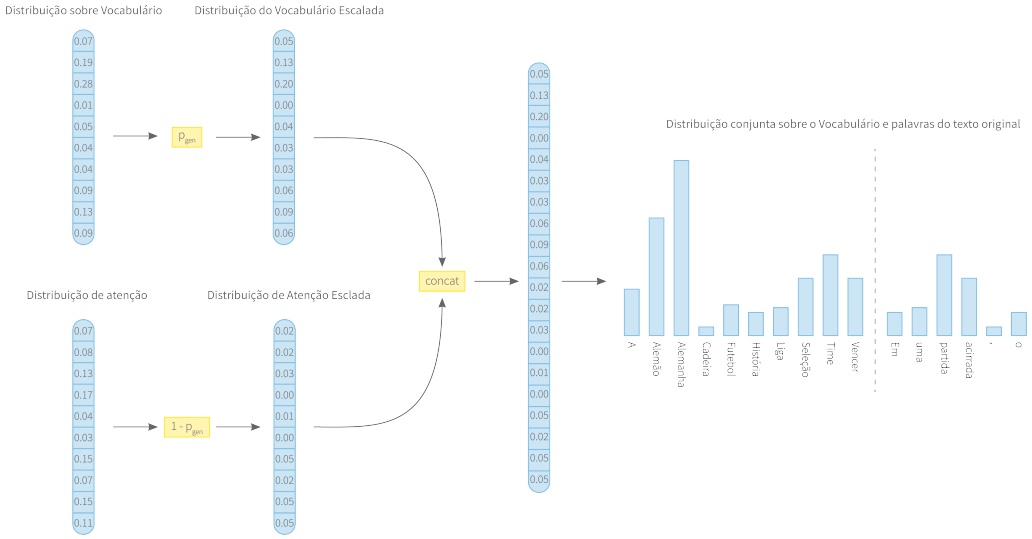
resumo. Essa possibilidade de apontar para palavras e trazê-las ao resumo é o que dá a parte *pointer* do nome Pointer-Generator.

A rede não está mais restrita a um vocabulário fixo para gerar textos. Então, mesmo quando lidando com tópicos específicos e jargões, ela será capaz de criar um resumo fiel. Mais do que isso, a rede também não está restrita ao léxico do documento original, podendo usar palavras novas para parafrasear e sintetizar com mais clareza o conteúdo.

De modo a permitir a cópia de palavras do material original, é necessária uma modificação no modo como as palavras são escolhidas para compor a sequência de saída.

Originalmente, a palavra era escolhida a partir de uma distribuição de probabilidades em um vocabulário predefinido. Com o mecanismo de ponteiro, esse vocabulário passa a ser estendido com os termos da sequência de entrada.

Figura 14 – Cálculo da nova distribuição levando em conta palavras do texto original



A imagem ilustra o processo de inclusão de palavras a serem copiadas do material fonte na distribuição original sobre o vocabulário.

O cálculo das probabilidades do vocabulário original continua o mesmo. Já para obter as probabilidades para as palavras do texto original, utilizam-se os coeficientes de atenção.

É importante lembrar que, separadamente, tanto a distribuição sobre o vocabulário quanto os coeficientes de atenção somam 1. Dessa forma, é necessário escalar esses valores de modo que a junção das distribuições some 1.

O escalonamento é controlado por uma variável de *probabilidade de geração* (representada por p_{gen}). Pode-se pensar em um sorteio com probabilidade p_{gen} de a rede gerar uma palavra nova e probabilidade $1 - p_{gen}$ de a rede copiar uma palavra do texto.

Dessa forma, é necessário atualizar as distribuições para refletir as novas probabilidades. O vetor com a distribuição sobre o vocabulário é multiplicado por p_{gen}. Analogamente, o vetor com a distribuição de atenção sobre as palavras do texto base é multiplicado por $1 - p_{gen}$.

Com essa modificação, a distribuição de probabilidade final pode ser descrita de acordo com a equação abaixo.

$$P(w) = p_{gen}P_{vocab}(w) + (1 - p_{gen}) \sum_{\{i|\forall w_i = w\}} \alpha_{t,i} \tag{3.18}$$

Onde $P_{vocab}(w)$ está definido de acordo com a equação 3.5 e $\alpha_{t,i}$ com a equação 3.16.

Note que para todas as palavras, são consideradas ambas as probabilidades. Dessa forma, se uma palavra está no vocabulário e na frase de entrada, ela terá ambas as probabilidades contribuindo a seu favor.

Para palavras que não compõem o vocabulário, $P_{vocab}(w) = 0$. Similarmente, se a palavra não ocorre na sequência de entrada, o termo $\sum_{i|w_i=w}$ resulta em 0.

3.2.7.2 Mecanismo de cobertura

A segunda modificação na arquitetura da rede é um mecanismo de cobertura de conteúdo. Seu intuito é tornar mais homogêneo o foco da rede durante a geração de resumos - para tentar garantir que todo o conteúdo do texto original recebeu um mínimo de atenção.

A cobertura também evita que a sequência de saída se repita muitas vezes, ou entre em recorrência (TU et al., 2016).

O mecanismo funciona mantendo uma contabilização da quantidade de vezes que a rede focou em cada trecho da sequência de entrada. A cada passo, acrescenta-se a um vetor inicialmente zerado a distribuição de atenção atual (eq. 3.19).

$$c^t = \sum_{t'=0}^{t-1} a^{t'} \tag{3.19}$$

O vetor c^t passa a servir como uma memória para a rede de todas as partes do material original que já foram atendidas. Dessa forma, ao ser incluído no cálculo da atenção do próximo passo, torna mais fácil a decisão de de quais partes merecem atenção no momento. A equação 3.15 é modificada para se tornar a equação 3.20.

$$e_{t,i} = {v_a}^T \tanh(W_a[s_{t-1}; h_i; c^t] + b_a) \tag{3.20}$$

Além de servir como um indicador de cobertura de conteúdo, esse acúmulo de distribuições de atenção também é utilizado para penalizar a rede na função de custo final (eq. 3.21). Essa penalização tem o intuito de desencorajar a rede a repetir muitas vezes o mesmo trecho - o que se traduziria em focar muitas vezes na mesma parte do documento original.

$$loss_t = -\log P(w_t^*) + \lambda \sum_{i}^{T_x} \min(a_i^t, c_i^t) \tag{3.21}$$

4 DATASET

Para botar em prática as ideias expostas até então, são necessários dados. Mais especificamentes dados estruturados e anotados para a tarefa de aprendizado supervisionado. O dataset escolhido foi o Webis-TLDR-17 (VÖLSKE et al., 2017). Esse dataset é fruto de uma nova técnica de extração de dados de sumarização por uma equipe da Bauhaus-Universität Weimar na Alemanha. A contribuição principal desse artigo foi notar que existem muitos resumos providenciados pelos próprios autores dos textos íntegros sob a forma de *TL;DR*.

Seja para poupar tempo, ter uma prévia do conteúdo ou outros motivos, tornou-se comum a prática de resumir textos encontrados na internet no formato TL;DR. A sigla representa o termo em inglês “Too Long; Didn’t Read", que traduzido para o português seria: “Muito Longo; Não Li". Essa forma de resumo assume a falta de paciência ou indisponibilidade de leitura do texto íntegro e tende a focar somente nos pontos essenciais do texto. Normalmente são utilizadas frases curtas e diretas, e o resumo não passa de três ou quatro frases - independentemente do tamanho do texto integral.

Isso pode ser comparado com o parágrafo *lead* de textos jornalísticos - que expõe mais sobre o conteúdo em comparação à manchete, mantendo o tamanho do texto curto e as ideias concisas.

O dataset Webis-TLDR-17 oferece cerca de 4 milhões de pares de texto-resumo para serem usados em um ambiente supervisionado. Diferentemente dos outros grandes datasets com propósitos similares - como o GigaWord, CNN / Daily Mail e DUC, focados em notícias jornalísticas - esse novo dataset oferece um recorte de mídias sociais.

4.0.1 Fonte

Os dados do Webis-TLDR-17 foram coletados de um site chamado Reddit. Reddit é um enorme fórum sobre os mais diversos assuntos e é o quinto site mais visitado dos Estados Unidos da América com mais de 540 milhões de usuários ativos mensais (ALEXA, 2018). Com esse nível de engajamento, o site se torna atraente sob o ponto de vista de produção de conteúdo. Somado a isso, a cultura de resumir textos no formato TL;DR é muito presente na comunidade frequentadora desse fórum.

Diferentemente de notícias, que tendem a ter um certo padrão de escrita e conteúdo - normalmente reportando eventos - o conteúdo hospedado no Reddit apresenta as mais diversas características. Os textos são escritos pelas mais variadas pessoas, desde novatos em determinado assunto sem prática de exposição argumentativa até especialistas responsáveis por desenvolver o *estado da arte* em seus respectivos ramos. Desde estrangeiros que não dominam a língua em questão completamente até escritores de livros.

Além de uma diversidade de autores, o Reddit também possui uma enorme variedade de assuntos. Os chamados "subreddits" são agregados de *posts* unidos pelo conteúdo sobre o qual tratam. Com mais de 1.1 milhões de subreddits (VÖLSKE et al., 2017), o site agrega discussões de matemática, física, política, jogos, filmes, histórias, humor, arte e diversos outros assuntos.

Levando toda essa variedade em consideração, dados coletados desse site trazem consigo uma riqueza tanto em estilo de escrita quanto em assuntos abordados. Entretanto é importante lembrar que, devido ao fato de serem posts de um fórum na internet, é comum encontrar erros gramaticais e sintáticos, bem como gírias e espressões específicas de determinados grupos. Ao mesmo tempo que enriquece o corpus, essa variedade também apresenta dificuldades a quem quiser trabalhar com o dataset.

4.0.2 Detalhes técnicos

O dataset foi disponibilizado pelos autores do artigo em novembro de 2017 em um arquivo de texto sob a formatação UTF-8. Com aproximadamente 18.3GB, o dataset possui 3,848,330 entradas. Cada entrada tem suas informações guardadas em um JSON que ocupa uma linha do arquivo. Os campos disponíveis são iguais para todas as instâncias do dataset, garantindo consistência.

A lista abaixo discrimina os campos de dados disponíveis para cada post coletado.

- **author**: `string`; Autor do post.
- **body**: `string`; Texto completo do post sem tratamento (incluindo TL;DR).
- **normalizedBody**: `string`; Texto completo do post com tratamento (incluindo TL;DR).
- **content**: `string`; Texto tratado sem o resumo (sem TL;DR).
- **content_len**: `long`; Comprimento do texto em `content`.
- **summary**: `string`; Resumo de `content` extraído a partir do TL;DR.
- **summary_len**: `long`; Comprimento do texto em `summary`.
- **id**: `string`; Identificador único do post no dataset.
- **subreddit**: `string`; Tópico do Reddit do qual o post foi retirado.
- **subreddit_id**: `string`; Identificador do `subreddit`.
- **title**: `string`; Título da discussão da qual o post foi retirado.

Por diversas razões, nem sempre é possível popular todos esses campos. O que implica em campos vazios em algumas instâncias do dataset. Ao todo, foram encontrados 2,085,437 itens com dados faltando (na maioria dos casos, o título da discussão). Entretanto, como para a tarefa em questão apenas serão utilizados os dados de `content` e `summary`, a quantidade de itens incompletos é reduzida a 0.

4.0.3 Tratamento de dados

Apesar de ter um tratamento de normalização, os autores mantiveram o texto quase que íntegro em seu dataset. Para o objetivo de treinar uma rede neural para sumarização de texto, alguns tratamentos adicionais devem ser feitos.

O primeiro leva em consideração a forma como uma rede lê dados. Como explicitado anteriormente, uma rede trabalha apenas com valores numéricos, implicando em alguma conversão de `string` para números. A opção escolhida para esse trabalho foi a utilização de *word embeddings*.

Quando se toma a decisão de usar word embeddings para a representação de palavras em um texto, deve-se tomar cuidado com algumas particularidades dessa opção. Em destaque, deve-se dar especial atenção em como lidar com termos com baixa frequência ou até que não façam parte do vocabulário.

Comumente referidas pelo seu acrônimo em inglês, OOV (Out of Vocabulary), essas palavras precisam ser representadas de alguma forma para que possam ser usadas pela rede. Um dos jeitos mais populares de contornar essa questão é atribuir a toda palavra fora do vocabulário um vetor especial - comumente representado por `<UNK>` ou `<OOV>`. Esse vetor é diferente dos demais, pois não representa uma palavra apenas, mas todas as que não pertencem ao vocabulário. Pode-se pensar que esse vetor representa a ideia de uma palavra fora do vocabulário - tanto semanticamente, quanto em contexto sintático quando inserida em uma frase.

Justamente por englobar vários termos em um único vetor, não é desejável ter muitas palavras mapeadas para `<UNK>`. É preferível que cada termo possua seu próprio vetor. Entretanto, isso não é possível quando há uma disjunção entre o conjunto de palavras usados para treinar os vetores do embedding e o conjunto de palavras dos textos que serão tratados na tarefa real.

Uma fonte comum dessa disjunção, ocorre quando baixando vetores pré-calculados de repositórios na internet. Os criadores desses dicionários procuram englobar o máximo de termos que conseguem, mas dependendo de onde serão aplicados podem estar faltando termos de interesse.

Outra razão pela qual não é tão simples livrar-se de termos mapeados para `<UNK>` se dá devido ao modo como os vetores são calculados. Existem diversas formas de se calcular vetores para representarem palavras, como, por exemplo, o Skip-Gram e suas variações (SG, SGNS) e o Continuous Bag of Words (CBOW) (MIKOLOV et al., 2013a). Con-

tudo, independentemente do algoritmo escolhido, os vetores das palavras são calculados baseados nos contextos nas quais elas aparecem.

Palavras podem ser usadas em diversos contextos, às vezes até mudando seu significado dependendo da frase em que está inserida. Dessa forma, é importante que se considere um número grande de cenários quando gerando um vetor, para que este seja condizente com o termo a que se propõe representar.

O segundo motivo para o uso de vetores `<UNK>` é para a representação de palavras com baixa frequência. Palavras que aparecem poucas vezes no corpus de treinamento pecam tanto no quesito de variabilidade de contexto quanto no de aparições frequentes para terem representações fiéis oriundas de um modelo de aprendizado baseado em dados. Por essa razão, muitas vezes é mais sensato aglomerar tais termos em um único vetor, o `<UNK>`.

Tendo ciência das causas que tornam o vetor `<UNK>` necessário, é possível tratar o texto de forma a reduzir o número de suas ocorrências. Pode-se pensar em uma técnica de acordo com os tipos de termos mais frequentes a serem mapeados a `<UNK>`: números e códigos.

Números de telefone, CPF, datas, URL de sites são tipos de termos que tendem a ter frequência baixa em na maioria dos corpus. A ocorrência de números de telefone em textos pode ser alta, entretanto a ocorrência de um número específico de telefone é pouco provável de ser alta. Isso implica que o modelo tem mais a ganhar aglomerando todos os números de telefone como um vetor especial `<TEL>` do que terem seus vetores individuais ou serem categorizados como `<UNK>`. O mesmo vale para os outros tipos de números e códigos.

A ideia é unir termos de baixa frequência, pelos mesmos motivos explicitados para `<UNK>`, de modo que o sentido semântico ainda seja preservado nessa união. Seguindo o exemplo dos números de telefone, todos têm o mesmo sentido semântico (com raras exceções, como: piadas e figuras de linguagem) e aparecem em situações sintáticas parecidas.

Os termos que serão aglomerados normalmente são decididos com base no problema em questão, o dataset disponível e como serão obtidos os word embeddings. No caso desse trabalho, foram substituídos os ítens explicitados na lista abaixo.

- <**DATE**>: O token engloba datas nos mais diversos formatos;
- <**DATE_RANGE**>: O token engloba períodos delimitados por datas - normalmente separados por hífen, como em "`03/04/1995 - 10/04/1995`";
- <**EMAIL**>: O token engloba emails;
- <**FRACTION**>: O token engloba frações escritas manualmente - como "`3/4`" ou "`2/7`";
- <**HANDLE**>: O token engloba nomes de usuários (nomes que começam com @);

- <**HOUR**>: O token engloba horas em diversos formatos;
- <**HOUR_RANGE**>: O token engloba períodos de horas - normalmente separados por hífen;
- <**MONEY**>: O token engloba valores monetários;
- <**ORDINAL**>: O token engloba números ordinais;
- <**NUMBER**>: O token engloba quaisquer outros números;
- <**NUMBER_RANGE**>: O token engloba intervalos de números;
- <**PERCENTAGE**>: O token engloba porcentagem;
- <**PERCENTAGE_RANGE**>: O token engloba interavlos de porcentagem;
- <**POSSIBLE_VALUE**>: O token engloba termos que potencialmente são valores monetários (mas podem estar faltando o símbolo da moeda);
- <**SUB_REDDIT**>: O token engloba subreddits;
- <**SYMBOL**>: O token engloba os símbolos {, }, #, $, %, &, \, §, _, =, +, º, ª, |;
- <**SYMBOL_SEQ**>: O token engloba sequência de caracteres inclusos em <SYMBOL>;
- <**TOPIC**>: O token engloba índices de uma lista ordenada;
- <**URL**>: O token engloba URL de sites;
- <**YEAR**>: O token engloba anos;
- <**YEAR_RANGE**>: O token engloba períodos de anos;
- <**HAS_NUMBER**>: O token engloba qualquer sequência delimitada por espaços em branco que possua um dígito, desde que não se encaixe em nenhuma outra categoria anterior;

Usando esse esquema de substituição, não resta nenhum número no texto. Isso elimina a fonte principal de termos de baixa frequência. Além disso, outras fontes de termos de baixa frequência são eliminadas, como URL e nomes de usuário.

A ideia de diversificar os tipos de token de aglomeração veio como uma originalidade desse trabalho em relação à referência principal. Da mesma forma que palavras substituídas por <UNK> ainda podem ser resgatadas através do mecanismo de cópia, as palavras substituídas pelos tokens listados acima, também podem.

Na arquitetura de referência, as palavras que não pertencem ao vocabulário podem ser postas no resumo através do mecanismo de cópia. Entretanto, se uma palavra fora

do vocabulário for escolhida para compor o resumo, na próxima iteração de geração de palavra, o vetor de <UNK> seria introduzido como entrada da rede.

Em contraste com a arquitetura de referência, se uma palavra fora do vocabulário for escolhida a ser posta no resumo, na próxima iteração um vetor da lista 4.0.3 poderá ser usado em vez de <UNK> de acordo com a palavra.

Isso disponibiliza à rede mais informações sobre o que está copiando do texto original. Afinal, os tokens estão discriminados semanticamente, ao invés de aglomerados em apenas um <UNK>.

4.0.4 Pré-processamento

Sabendo que tipos de tratamentos serão feitos no texto, passa-se à fase de efetivamente aplicá-los no dataset.

Neste ponto, é possível tomar dois caminhos: aplicar os processamentos de maneira *ad hoc* durante a execução do programa (treinamento do modelo, avaliação modelo e inferência com o modelo), ou aplicá-los preemptivamente e utilizar um dataset pré-processado para trabalhar.

No caso de inferência com o modelo, não é possível utilizar um pré-processamento. Afinal, não se sabe a entrada a priori. Dessa forma, seu tratamento só possível uma vez que o programa receba a entrada.

Já nos outros dois cenários - treinamento e avaliação do modelo-, é possível fazer esse tratamento antes. Resta decidir se vale a pena.

A maior vantagem de pré-processar o dataset é o fato de que o tratamento só rodará uma vez - indepentemente de quantas vezes o dataset for usado. Se a cada execução o dataset tivesse que ser tratado, muito reprocessamento iria ocorrer - o que não é desejável. Tendo em vista que o dataset tem cerca de 4 milhões de entradas, quanto menos processamento redundante houver, melhor.

Para esse trabalho, foi então criada uma versão tratada do dataset original. Em primeiro lugar, todos os textos tiveram sua capitalização reduzida a minúscula. Essa tarefa é relativamente rápida de se fazer e não muito custosa.

Para separar os textos corridos em tokens, foi utilizado o StanfordNLP. Essa operação já passa a ser mais custosa e demorada. É importante dizer que esse parser trabalha de forma paralela para tokenizar um texto em si. Entretanto, mesmo com essa arqutietura, os processadores da máquina estavam sendo subutilizados. Para garantir máxima utilização, foi feita uma paralelização adicional a nível de processos.

Com as palavras já tokenizadas, pode-se escolher descartar as chamadas *stop words*. As stop words são palavras com altíssima freqência em uma língua, mas que não carregam muito significado em si - palavras como artigos, preposições, pronomes, entre outras. Entretanto, faz parte do problema que esse trabalho se propõe a resolver escrever textos

gramaticalmente corretos. Dessa forma, não é adequado remover as stop words nesse cenário.

O próximo tratamento a ser feito é uma otimização para o treinamento da rede neural. Como explicitado anteriormente, é preciso transformar as palavras que são `strings` em vetores numéricos. Uma forma de fazer isso é utilizando um hashmap que leva de uma palavra à seu vetor.

Entretanto, existem maneiras de otimizar ainda mais esse processo. Em vez de usar a palavra diretamente como chave, pode-se criar um dicionário relacionando a cada palavra um índice. Posteriormente, esse índice pode ser usado para associar a palavra original a um vetor de embedding.

A vantgagem de usar essa representação intermediária de índices é que torna-se possível alocar todos os vetores de embedding em uma matriz contígua em memória. Isso torna suas operações mais fáceis de serem otimizadas.

Cada palavra têm um índice associado, o que gera outra vantagem. No arquivo com o dataset, podem ser salvos apenas os índices das palavras correspondentes. Isso torna o dataset mais leve, ocupando menos espaço em disco.

Se o cuidado de ordenar as palavras no vocabulário de acordo com frequência for tomado, a redução de espaço é maior ainda. Afinal, palavras mais frequentes terão índices menores - com menos dígitos -, e portanto ocuparão menos espaços.

Por fim, o dataset tratado pode ser salvo como um arquivo CSV contendo em cada linha um texto e seu resumo já normalizados. As linhas contém uma sequência de inteiros separados por vírgulas para representar o texto e o resumo. Para separar o texto do resumo, os caracteres "`||`" foram utilizados.

Salvar o dataset nesse formato, trouxe mais dois benefícios. O primeiro está no fato de reduzir a quantidade de informação salva. Guardando apenas os campos `content` e `summary` e utilizando a formatação de índices, o tamanho total do dataset reduziu para 3.9GB - uma redução de mais de 80% do tamanho original.

A segunda vantagem de se usar um CSV está no fato de que é um formato simples e fácil de fazer parse. Em contraste com o CSV, o JSON - formato original de cada linha do dataset - é muito mais demorado para se converter de bytes em um arquivo para objetos em memória utilizáveis.

Treinamento de redes neurais normalmente são demorados. Dessa forma, qualquer ajuda em redução de tempo de processamento é essencial. A versão tratada do dataset teve impacto significativo na agilização do treino.

5 RESULTADOS E DISCUSSÃO

Uma vez com a arquitetura implementada e o dataset tratado, resta efetivamente treinar a rede e medir seu desempenho na tarefa de sumarização automática.

5.1 HIPER PARÂMETROS

Quando treinando uma rede neural, há uma série de parâmetros que não são calibrados automaticamente pelo algoritmo de otimização - consequentemente devem ser configurados manualmente. A esse parâmetros, dá-se o nome de *hiper parâmetros*. São hiper parâmetros configurações como: taxa de aprendizado, tamanho do batch, tamanho das camadas da rede, algoritmo de otimização, entre outras.

Justamente por terem que ser calibrados manualmente, os hiper parâmetros são uma parte delicada do processo de treino. Algumas redes só são capazes de aprender com a combinação certa dos mesmos. Dependendo da arquitetura a rede neural pode ser mais ou menos sensível à mudanças feitas nos hiper parâmetros.

Na referência principal, a rede é treinada em batches de 16 exemplos e taxa de aprendizado fixa de 0,15 utilizando o algoritmo ADAGRAD (DUCHI; HAZAN; SINGER, 2011). O valor do acumulador inicial do ADAGRAD é 0,1. Além disso, os textos completos são truncados em 400 tokens. Já os resumos são truncados em 100 tokens durante o treino, mas podem ser gerados resumos de até 120 tokens durante a fase de teste.

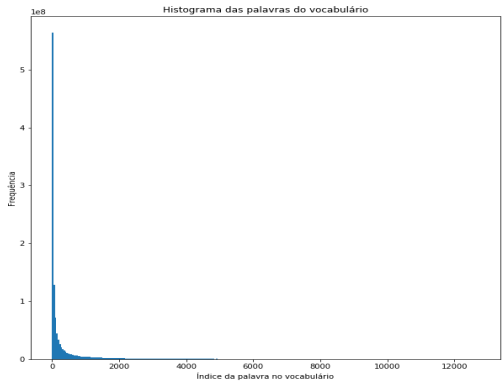
A decisão de truncar os resumos foi tomada para facilitar o aprendizado da rede. Isso, de fato, melhorou o desempenho do modelo, bem como torna o modelo mais rápido de treinar (SEE; LIU; MANNING, 2017).

O artigo também realiza testes com um vocabulário limitado em 50.000 e 150.000 palavras. Os resultados principais da comparação entre os dois tamanhos mostram que aumentar o vocabulário dobra o tempo de treino e não há melhora significativa em termos de performance do modelo.

Levando esses fatores em consideração, esse trabalho procura fazer otimizações para permitir o treino, alto desempenho e agilidade do modelo, ao mesmo que não comprometendo o problema original.

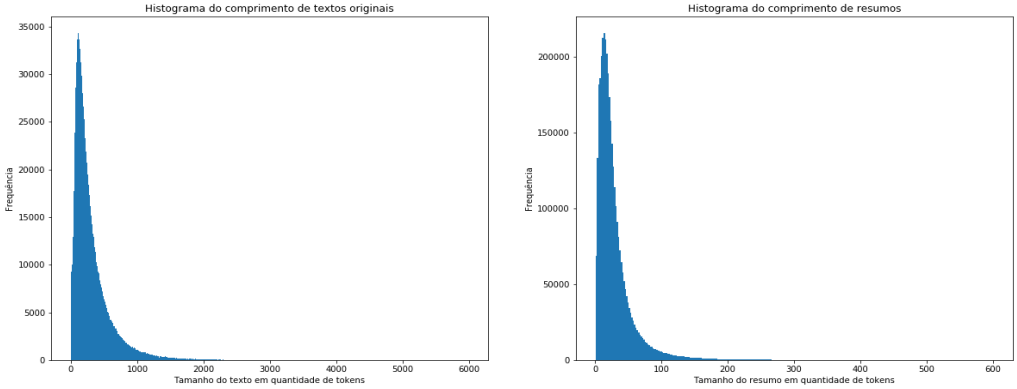
O primeiro passo foi analisar o histograma do vocabulário (figura 15). O vocabulário possui ao todo 1.690.739 palavras - extraído do dataset em questão. Inicialmente foi feito um corte em 50.000 palavras, como proposto na referência principal. Dada a frequência de ocorrência das palavras, mesmo retirando mais de 97% das palavras únicas, os textos ainda mantém 99,3% de suas palavras originais (palavras que não foram substituídas por `<UNK>`).

Figura 15 – Histograma de ocorrências das palavras do vocabulário



Seguindo os passos da referência principal, também é necessário escolher um valor para truncar as sequências de entrada e saída. Analisando o histograma da figura 16 e com o intuito de não cortar excessivamente o tamanho dos textos a serem resumidos, foi determinado um corte em 2500 tokens. Nesse mesmo contexto, os resumos foram truncados em 300 tokens.

Figura 16 – Histograma do comprimento dos textos completos e seus resumos



Além de decidir os hiper parâmetros baseados no dataset, precisa-se decidir hiper parâmetros relativos à arquitetura da rede. Foram utilizadas células de LSTM de tamanho 256 para construir a rede recorrente bidirecional. A dimensão dos vetores de embedding para as palavras do vocabulário é 128.

Além disso, é aplicado um corte em magnitude máxima do gradiente que será usado para atualizar os pesos. Isso foi feito para evitar o problema de *exploding gradient* (BENGIO et al., 1994). A norma máxima permitida a um gradiente é 2 - valor obtido a partir de (SEE, 2017).

Para inicialização dos pesos foram usadas distribuições uniformes, normais truncadas,

e a inicialização de Xavier (GLOROT; BENGIO, 2010) em diferentes partes da rede.

Seguindo a referência principal, o tamanho do batch foi 16 e a taxa de aprendizado 0,15.

5.2 DESAFIOS DE EXECUÇÃO

Entretanto, a primeira barreira ao tentar treinar o modelo com os hiper parâmetros destacados é memória em placa de vídeo. Muito menor que a capacidade de RAM principal, as memorias de placas de vídeo costumam ser bem mais escassas.

Utilizando a configuração mencionada, não é possível colocar o modelo em uma placa de vídeo convencional. Seriam necessários arranjos mais elaborados com múltiplas placas de vídeo, ou placas de vídeo especializadas com mais memória.

Para contornar esse problema, a primeira escolha foi reduzir o tamanho do batch. Ao reduzir o tamanho do batch, a arquitetura da rede em si é conservada e consequentemente sua capacidade de aprendizado também é mantida. Através de busca binária, foi encontrado o tamanho 8 como o máximo que cabe em memória para o modelo sem alterações.

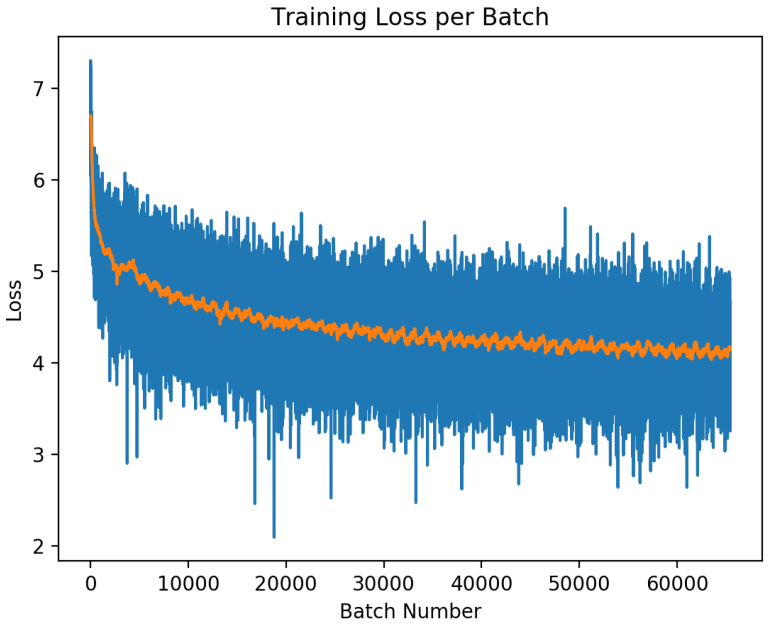
Entretanto, reduzir o tamanho do batch traz consequências indesejáveis. Com um batch menor, a rotina de treino se torna mais errática e consequentemente o modelo tende a variar mais. Afinal, os pesos do modelo são atualizados para melhorar o desempenho nos exemplos vistos em um batch. Se esse batch não for representativo do dataset como um todo, o modelo pode acabar melhorando para aqueles exemplos específicos e piorando para outros.

Como ilustrado no gráfico da figura 17, o desvio padrão do valor do custo é alto - quando comparado a outros experimentos feitos para esse trabalho. Com exceção das épocas iniciais, é possível identificar um processo lento, de acordo com o número de iterações, de aprendizado - no qual a média ainda está diminuindo, porém de forma muito devagar. Devido à natureza variada mais proeminente em batches de tamanho pequeno, a convergência dos pesos do modelo tende a demorar mais a acontecer - caso aconteça.

Uma possível tentativa de mitigação dos problemas oriundos de um batch pequeno é a redução da taxa de aprendizado. Reduzindo a taxa de aprendizado, os pesos tendem a se modificar menos a cada atualização. Dessa forma mesmo os batches potencialmente sendo discrepantes uns dos outros, cada um não modifica tanto os pesos da rede. A esperança é que, devido às perturbações pequenas, apenas a tendência principal compartilhada por todos os batches tem uma contribuição significativa às atualizações dos parâmetros da rede.

Foram feitos treinos com a taxa de aprendizado reduzida para 0,015 e 0,0015. A figura 18 mostra a evolução do custo para a rede treinada com 0,0015. Quando comparado ao treinamento com taxa de 0.15, não é possível notar uma redução considerável em desvio

Figura 17 – Custo por batch processado com taxa de aprendizado de 0,15



O gráfico mostra o custo por batch em azul e a média exponencial em laranja. Para calcular a média exponencial foi usado um fator de 0,99.

padrão do custo. Enquanto no treino com taxa de 0,15 o desvio padrão foi 0,45, no treino com taxa de 0,0015 o desvio padrão foi 0,41 - uma redução apenas de aproximadamente 9%. Isso é um indicador que o problema de discrepância alta dos batches pequenos não foi efetivamente mitigado.

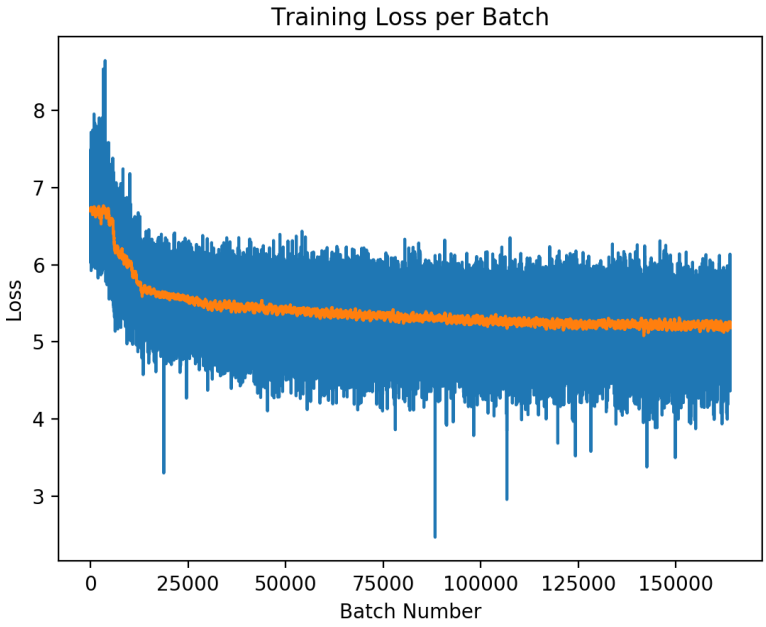
Além disso, reduzir a taxa de aprendizado traz consigo a desvantagem de tornar o processo de treinamento mais longo. Na prática, é possível ver que mesmo com uma quantidade significativamente maior de batches treinados, o modelo treinado com a taxa de 0,0015 não conseguiu performance superior ao treinado com taxa de 0,15.

A princípio esse problema não deveria ser tão grave, pois bastaria esperar mais iterações de treino para se obter um resultado melhor. Entretanto, devido aos problemas numéricos encontrados - que serão explorados na seção 5.2.1 -, manter uma rede treinando por tempo suficiente até atingir convergência não foi possível.

Buscando outra alternativa à redução do batch, é possível reduzir o tamanho do modelo em si para ocupar menos espaço. Uma opção interessante é reduzir o tamanho do vocabulário. Reduzindo o vocabulário, reduz-se a quantidade de vetores de embedding a serem aprendidos durante o treinamento.

Investigando o histograma da figura 15, é possível notar um corte no valor 5000 abrange uma quantidade farta de ocorrências das palavras do vocabulário. De fato, se o vocabulário for encurtado para as 5000 palavras mais frequentes - menos de 0,003% do total de palavras únicas -, o total de ocorrências de palavras mantidas cai apenas para 93,4%.

Figura 18 – Custo por batch processado com taxa de aprendizado de 0,0015



O gráfico mostra o custo por batch em azul e a média exponencial em laranja. Para calcular a média exponencial foi usado um fator de 0,99.

Contudo, ao testar essa redução, não foi possível aumentar o batch expressivamente. Mesmo com as reduções no tamanho do vocabulário, que afeta principalmente o tamanho da matriz de *word embeddings*, foi apenas possível aumentar o tamanho do batch em uma unidade. Essa situação mostrou que eram necessárias reduções mais severas nas condições de treino da rede.

Buscando inspiração nos hiper parâmetros da referência principal, o truncamento dos textos originais foi reduzido para permitir um máximo de 400 tokens. Para os resumos, a regra de 100 tokens para treino e 120 para teste foi adotada.

Essa modificação efetivamente trouxe uma redução significativa em espaço de memória, o que possibilitou o treinamento com um batch de tamanho 16.

Com a barreira de memória superada, uma nova se torna proeminente: instabilidade numérica.

5.2.1 Instabilidade de treino

Com o modelo treinando com parâmetros adequados, o problema de implementação introduzido na seção 1.5 se tornou mais evidente. Durante o treino, os cálculos feitos no processamento se provaram numericamente instáveis. Com isso variáveis passaram a assumir o valor de `NaN` (*Not a Number* em inglês, ou "Não um Número" em português).

Variáveis numéricas podem assumir o valor de NaN por diversas razões. As mais co-

muns sendo divisão por zero ou operações com infinito. Números que são muito próximos de zero ou muito grandes - próximos do limite superior de representação para números de ponto flutuante em um computador digital - também tendem a gerar esse comportamento dependendo das operações realizadas.

Encontrar a origem dessa instabilidade numérica requer um trabalho cuidadoso de apuração.

Um bom ponto para começar é com a parte do código que efetivamente faz os cálculos do modelo. A biblioteca numérica por trás da implementação da rede neural desse trabalho é o Tensorflow (ABADI et al., 2016).

O Tensorflow é um projeto de código aberto para trabalhar com redes neurais e outros modelos de aprendizado de máquina. A biblioteca tem uma camada de abstração exposta em Python, mas, por motivos de performance, tem o seu *back end* escrito em C++.

Para identificar as condições em que o erro de NaN ocorria, foram feitos diversos treinos testando diferentes combinações de hiper parâmetros. O resultado dos experimentos mostrou que não importando a configuração de hiper parâmetros, o modelo continuava encontrando NaN durante o treino. A rede treinava tão pouco que não existia clara distinção entre os diferentes hiper parâmetros, tampouco aprendizado significativo para ser reportado aqui.

Esse fato apontou que talvez a origem do problema não estivesse nos valores utilizados para calibrar a rotina de treino da rede, mas na própria arquitetura do modelo. De fato, foram encontrados relatos de outras pessoas com o mesmo problema no repositório com a implementação oficial do artigo - corroborando com a hipótese de que há alguma inconsistência no modelo.

Em discussões com a autora no repositório do código (DU, 2017), não chegaram a um consenso quanto à origem do problema. Tampouco foi descoberta uma solução definitiva.

Uma hipóteses levantada na discussão atribui a causa dos valores NaN a inconsistências entre palavras fora do vocabulário que podem ser copiadas dentro de um mesmo batch. A parte do código que permite a junção das distribuições do vocabulário com as distribuições de palavras no texto original (para realizar o mecanismo de cópia) é um tanto complicada e a própria autora levantou questionamentos sobre tal.

Se em algum momento durante o treino ocorresse de uma palavra correta ter probabilidade igual a zero, a função de custo teria valor infinito. A partir desse ponto, as operações com infinito gerariam valores NaN.

Como uma medida de teste, o código deste trabalho foi modificado para acrescentar um valor minúsculo (épsilon da máquina) às probabilidades para garantir que nenhuma seja 0. Contudo, mesmo com essa modificação, os erros de NaN continuavam a surgir.

Outras possibilidades de mitigação contra os erros de NaN foram levantadas na discussão (DU, 2017) e implementadas, porém nenhuma resolveu o problema.

Há também a preocupação com a biblioteca Tensorflow. Apesar de ser uma biblioteca

mantida pela Google e adotada em massa pela comunidade de Inteligência Artificial, é conhecida por ter problemas com NaN.

A biblioteca foi desenvolvida com os primeiros modelos de redes neurais em mente: redes *fully connected*, redes convolucionais, redes recorrentes. Entretanto, as arquiteturas de redes evoluiram com o tempo e se tornaram mais complexas e dinâmicas.

Arquiteturas dinâmicas são especialmente difícieis de se implementar no paradigma declarativo do Tensorflow. É o caso da arquitetura da rede pointer-generator. A própria autora do artigo de referência escreve em comentários no código que certas partes são "sensíveis" (*fiddly* em inglês).

Levanto o questionamento de se as bibliotecas atuais de redes neurais são suficientes e acompanham o ritmo de desenvolvimento da área. Potenciais respostas para essa pergunta serão comentadas na seção 6.

5.2.2 Resultados

Devido aos problemas numéricos, sob nenhuma condição a rede foi capaz de treinar por muito tempo. A referência principal mostra resultados de estado da arte após treinarem por aproximadamente 600.000 iterações com batches de 16 exemplos. Isso resulta em aproximadamente 9.600.000 de exemplos vistos pela rede durante o treino - gerando um valor de custo final próximo a 0,2 no melhor cenário de acordo com o artigo (SEE; LIU; MANNING, 2017).

A quantidade máxima de iterações que uma rede treinou nesse trabalho está próximo de 170.000 - considerando um batch de tamanho 16, foram 2.720.000 de exemplos vistos durante treino. O valor médio de custo após esse período de treino flutuava um pouco acima de 5. Mesmo considerando os *outliers*, o valor de custo nunca ficou abaixo de 2.

Comparar o valor de custo por si só não é um indicador definitivo de aprendizado de um modelo de rede neural. Entretanto a discrepância entre os custos desse trabalho e da referência principal é grande o suficiente para exigir atenção.

Para conseguir uma prova definitiva do desempenho do modelo, os melhores pesos foram usados em um conjunto de dados separados para teste. Observando esses resultados, foi constatado que o modelo sequer aprendeu a escrever frases coerentes.

Os textos gerados pelo modelo são repetições de palavras de altíssima frequência na língua inglesa, como `the` e `a`, bem como pontuações também muito presentes, como o ponto final e a vírgula.

Olhando para esses resultados, são condizentes com o valor de custo visto nos gráficos de treino. Afinal quando a rede aprende a usar palavras de alta frequência na língua, o custo tende a cair mais rápido. Entretanto, para que o custo continue caindo, é necessário que pelo menos um modelo da língua seja aprendido - de forma a escrever frases gramaticalmente plausíveis. Em sequência esperaria-se que o conteúdo do resumo fosse pertinente - e no melhor caso, representasse bem - o conteúdo do texto original.

Outro fator a ser considerado é que não será possível aplicar as métricas qualitativas de avaliação de resumos com os textos resultantes dos experimentos desse trabalho. Seria muito interessante ver como a rede se comportaria nos quesitos de conteúdo, tamanho e fluidez, conforme dito na seção 3.1.

Na mesma linha, faz pouco sentido também aplicar as métricas do ROUGE. Afinal em nenhuma delas, o modelo é capaz de atingir uma pontuação significativa. Consegue apenas interseções ocasionais com pontuação e palavras de alta frquência em relação ao resumo de referência.

6 CONCLUSÃO E TRABALHOS FUTUROS

Levando em conta os resultados e problemas vistos durante esse trabalho, pode-se constatar alguns pontos.

Em primeiro lugar, mesmo com um conjunto de dados anotados enorme, ainda é complicado treinar uma rede neural de estado da arte. Existem considerações de implementação e hardware que precisam ser levadas em conta. Existem considerações numéricas que também demandam atenção para que o treino seja possível.

Em segundo lugar, arquiteturas de redes neurais mais complexas podem demonstrar incrível potencial teórico, mas não se comportar bem quando implementada.

Apesar de não estar clara a origem do problema encontrado neste trabalho, uma potencial forma de se contornar os valores NaN é formular o problema de outra maneira. Talvez implementando o modelo de um modo alternativo - porém matematicamente equivalente - torne o programa mais estável numericamente. A implementação deveria levar em conta os fortes e fracos de computação numérica.

Outra alternativa seria trocar a biblioteca utilizada para implementar a rede. Esse ponto toca no questionamento feito sobre a capacidade de implementação das bibliotecas atuais. Existem alternativas promissoras, talvez a mais preponderante sendo o PyTorch (PASZKE et al., 2017). Até mesmo a segunda versão do Tensorflow, promete mais estabilidade numérica e facilidade de construção de arquiteturas complexas. O caminho que tais bibliotecas parecem seguir é o do dinamismo e maior complexidade de modelos de redes neurais.

Abordagens alternativas interessantes para reduzir o uso de memória do modelo contemplam a utilização de vetores de embedding pré-calculados. Existem diversos conjuntos de vetores para palavras em diferentes línguas disponíveis na internet. Em sua maioria, esses modelos são treinados em corpus enormes de documentos e prometem capturar as mais diversas características de cada palavra.

Existem trabalhos que procuram aumentar a quantidade de informações por palavra, gerando por exemplo um vetor distinto para cada classe sintática de uma mesma palavra (TRASK; MICHALAK; LIU, 2015). Existem pesquisas trabalhando com representações que levam em conta morfologia de palavras - de forma a deduzir novos vetores para palavras raras ou nunca antes vistas durante o treino (BOJANOWSKI et al., 2017b). Essas vertentes também parecem promissoras a se combinar com a sumarização estudada neste trabalho.

Além do aspecto de modelagem, seria interessante o estudo do comportamento de uma arquitetura com a capacidade de cópia em um dataset com resumos altamente sintéticos. De acordo com os autores do Webis- TLDR-17, apenas 966.430 dos quase 4 milhões de pares de texto-resumo possuem interseção em sintagmas nominais (*phrasal nouns* em

inglês). Isso pode ser devido à natureza altamente abstrata de resumos feitos por humanos.

Outro aspecto a ser levado em conta quando explorando os resultados com esse dataset é o intuito do Tl;Dr. Há diversos casos de uso para o termo Tl;Dr. Os autores do dataset constataram que os usuários do Reddit utilizam a expressão com três intuitos.

O primeiro é um resumo no sentido tradicional - um texto menor que contempla as ideias principais expostas no original.

O segundo caso de uso é para realizar perguntas. Enquanto o texto principal pode elucidar um contexto, dar detalhes, mostrar fatos relevantes, o Tl;Dr apenas consiste da pergunta final de interesse.

Finalmente, o terceiro caso é uma conclusão. O Tl;Dr pode ser utilizado para fechar um argumento dando uma conclusão ou opinião sobre o tema. Não necessariamente será um resumo do texto, e existe o caso de o conteúdo da conclusão não ser consequência lógica do conteúdo original.

Levando em conta que o dataset de treino da rede possui todos esses cenários, seria interessante investigar se todos esses casos de uso se transmitiram aos textos da rede. Quais são os casos mais proeminentes de Tl;Dr que a rede utiliza? Bem como, se a rede é capaz de aprender cada caso separadamente, ou se alguma mistura é feita são cenários interessantes a serem investigados.

Por fim, seria interessante calcular a taxa de compressão dos resumos da rede em relação aos textos originais. No dataset original a média de compressão flutua próximo de 88% (os resumos são 88% menores que o texto original). Comparar essas taxas levando em conta a relevância do conteúdo poderia mostrar o quão bem o modelo sintetiza um conteúdo - sendo uma forma importante de diagnóstico de desempenho da rede.

Outro lado a ser destacado desse trabalho é o detalhe exposto sobre arquiteturas e modelagens para resolver o problema de otimização. Um dos objetivos desse texto é justamente tornar mais claro o panorama nesse setor de pesquisa.

Com a explicação das minúcias de diversas arquiteturas compiladas em um só lugar, espera-se que sirva como material educativo para leitores que não necessariamente são peritos na área - especialmente aqueles que tenham apenas um entendimento básico do paradigma de redes neurais.

Aqui está a explicação de um tópico que desperta interesse em diversos pesquisadores em todo o mundo e potencialmente tornará o ramo mais acessível.

REFERÊNCIAS

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. In: **12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)**. [s.n.], 2016. p. 265–283. Disponível em: <https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf>. Acesso em: 06 jun.2019.

ALEXA. Reddit competitive analysis, marketing mix and traffic - alexa. 2018. Disponível em: <https://www.alexa.com/siteinfo/reddit.com>. Acesso em: 06 jun.2019.

BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. **arXiv preprint arXiv:1409.0473**, 2014. Disponível em: <https://arxiv.org/pdf/1409.0473.pdf>. Acesso em: 06 jun.2019.

BENGIO, Y. et al. Learning long-term dependencies with gradient descent is difficult. **IEEE transactions on neural networks**, v. 5, n. 2, p. 157–166, 1994. Disponível em: <http://www.comp.hkbu.edu.hk/~markus/teaching/comp7650/tnn-94-gradient.pdf>. Acesso em: 06 jun.2019.

BOJANOWSKI, P. et al. Enriching word vectors with subword information. **Transactions of the Association for Computational Linguistics**, v. 5, 2017. Disponível em: <https://www.mitpressjournals.org/doi/pdf/10.1162/tacl_a_00051>. Acesso em: 06 jun.2019.

BOJANOWSKI, P. et al. Enriching word vectors with subword information. **Transactions of the Association for Computational Linguistics**, MIT Press, v. 5, p. 135–146, 2017. Disponível em: <https://aclweb.org/anthology/Q17-1010>. Acesso em: 06 jun.2019.

CAO, Z. et al. Ranking with recursive neural networks and its application to multi-document summarization. In: **Twenty-ninth AAAI conference on artificial intelligence**. [s.n.], 2015. Disponível em: <https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9414/9520>. Acesso em: 06 jun.2019.

CHIU, C.-C. et al. State-of-the-art speech recognition with sequence-to-sequence models. In: IEEE. **2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. 2018. p. 4774–4778. Disponível em: <https://arxiv.org/pdf/1712.01769.pdf>. Acesso em: 06 jun.2019.

CHO, K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. **CoRR**, abs/1406.1078, 2014. Disponível em: <http://arxiv.org/abs/1406.1078>. Acesso em: 06 jun.2019.

CHOPRA, S.; AULI, M.; RUSH, A. M. Abstractive sentence summarization with attentive recurrent neural networks. In: **Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**. [s.n.], 2016. p. 93–98. Disponível em: <https://www.aclweb.org/anthology/N16-1012>. Acesso em: 06 jun.2019.

DU, S. Get nan loss after 35k steps 4. In: . [s.n.], 2017. Disponível em: <https://github.com/abisee/pointer-generator/issues/4>. Acesso em: 06 jun.2019.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, n. Jul, p. 2121–2159, 2011. Disponível em: <http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>. Acesso em: 06 jun.2019.

DUŠEK, O.; NOVIKOVA, J.; RIESER, V. Evaluating the state-of-the-art of end-to-end natural language generation: The e2e nlg challenge. **arXiv preprint arXiv:1901.07931**, 2019. Disponível em: <https://arxiv.org/pdf/1901.07931.pdf>. Acesso em: 06 jun.2019.

EDMUNDSON, H. P. New methods in automatic extracting. **Journal of the ACM (JACM)**, v. 16, n. 2, 1969. Disponível em: <http://courses.ischool.berkeley.edu/i256/f06/papers/edmonson69.pdf>. Acesso em: 06 jun.2019.

FUNG, P.; NGAI, G.; CHEUNG, C.-S. Combining optimal clustering and hidden markov models for extractive summarization. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the ACL 2003 workshop on Multilingual summarization and question answering-Volume 12**. 2003. p. 21–28. Disponível em: <https://www.aclweb.org/anthology/W03-1203>. Acesso em: 06 jun.2019.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: **Proceedings of the thirteenth international conference on artificial intelligence and statistics**. [s.n.], 2010. p. 249–256. Disponível em: <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>. Acesso em: 06 jun.2019.

GRAVES, A.; SCHMIDHUBER, J. Framewise phoneme classification with bidirectional lstm and other neural network architectures. **Neural Networks**, Elsevier, v. 18, n. 5-6, p. 602–610, 2005. Disponível em: <https://mediatum.ub.tum.de/doc/1290194/file.pdf>. Acesso em: 06 jun.2019.

HAJIRAHIMOVA, M. S.; ALIYEVA, A. S. About big data measurement methodologies and indicators. **International Journal of Modern Education and Computer Science (IJMECS)**, v. 9, n. 10, 2017. Disponível em: <http://www.mecs-press.org/ijmecs/ijmecs-v9-n10/v9n10-1.html>. Acesso em: 06 jun.2019.

HERMANN, K. M. et al. Teaching machines to read and comprehend. In: **Advances in neural information processing systems**. [s.n.], 2015. p. 1693–1701. Disponível em: <https://openreview.net/forum?id=BJJsrmfCZ>. Acesso em: 06 jun.2019.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997. Disponível em: <https://www.bioinf.jku.at/publications/older/2604.pdf>. Acesso em: 06 jun.2019.

KARRAS, T. et al. Progressive growing of gans for improved quality, stability, and variation. **arXiv preprint arXiv:1710.10196**, 2017. Disponível em: <https://arxiv.org/abs/1710.10196>. Acesso em: 06 jun.2019.

LI, P. et al. Deep recurrent generative decoder for abstractive text summarization. **arXiv preprint arXiv:1708.00625**, 2017. Disponível em: <https://arxiv.org/pdf/1708.00625.pdf>. Acesso em: 06 jun.2019.

LIN, C.-Y. Rouge: A package for automatic evaluation of summaries. **Text Summarization Branches Out**, 2004. Disponível em: <https://www.aclweb.org/anthology/W04-1013>. Acesso em: 06 jun.2019.

MESCHEDER, L.; GEIGER, A.; NOWOZIN, S. Which training methods for gans do actually converge? **arXiv preprint arXiv:1801.04406**, 2018. Disponível em: <https://arxiv.org/abs/1801.04406>. Acesso em: 06 jun.2019.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013. Disponível em: <https://arxiv.org/pdf/1301.3781.pdf>. Acesso em: 06 jun.2019.

MIKOLOV, T. et al. Distributed representations of words and phrases and their compositionality. In: **Advances in neural information processing systems**. [s.n.], 2013. p. 3111–3119. Disponível em: <https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>. Acesso em: 06 jun.2019.

MIKOLOV, T.; YIH, W.-t.; ZWEIG, G. Linguistic regularities in continuous space word representations. In: **Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**. [s.n.], 2013. p. 746–751. Disponível em: <https://www.aclweb.org/anthology/N13-1090>. Acesso em: 06 jun.2019.

MIRZA, M.; OSINDERO, S. Conditional generative adversarial nets. **arXiv preprint arXiv:1411.1784**, 2014. Disponível em: <https://arxiv.org/abs/1411.1784>. Acesso em: 06 jun.2019.

NALLAPATI, R. et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. **arXiv preprint arXiv:1602.06023**, 2016. Disponível em: <https://arxiv.org/pdf/1602.06023.pdf>. Acesso em: 06 jun.2019.

OLAH, C. et al. The building blocks of interpretability. **Distill**, 2018. Disponível em: <https://distill.pub/2018/building-blocks>. Acesso em: 06 jun.2019.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. In: **International conference on machine learning**. [s.n.], 2013. p. 1310–1318. Disponível em: <https://arxiv.org/abs/1211.5063>. Acesso em: 06 jun.2019.

PASZKE, A. et al. Automatic differentiation in pytorch. In: **NIPS-W**. [s.n.], 2017. Disponível em: <https://openreview.net/forum?id=BJJsrmfCZ>. Acesso em: 06 jun.2019.

RADEV, D. R.; HOVY, E.; MCKEOWN, K. Introduction to the special issue on summarization. **Computational linguistics**, v. 28, n. 4, 2002. Disponível em: <https://www.aclweb.org/anthology/J02-4001>. Acesso em: 06 jun.2019.

SALIMANS, T. et al. Improved techniques for training gans. In: **Advances in neural information processing systems**. [s.n.], 2016. p. 2234–2242. Disponível em: <https://arxiv.org/pdf/1606.03498.pdf>. Acesso em: 06 jun.2019.

SCHUSTER, M.; PALIWAL, K. K. Bidirectional recurrent neural networks. **IEEE Transactions on Signal Processing**, IEEE, v. 45, n. 11, p. 2673–2681, 1997. Disponível em: <https://www.researchgate.net/profile/Mike_Schuster/publication/3316656_Bidirectional_recurrent_neural_networks/links/56861d4008ae19758395f85c.pdf>. Acesso em: 06 jun.2019.

SEE, A. Code for the acl 2017 paper "get to the point: Summarization with pointer-generator networks". In: . [s.n.], 2017. Disponível em: <https://github.com/abisee/pointer-generator>. Acesso em: 06 jun.2019.

SEE, A.; LIU, P. J.; MANNING, C. D. Get to the point: Summarization with pointer-generator networks. **arXiv preprint arXiv:1704.04368**, 2017. Disponível em: <https://arxiv.org/pdf/1704.04368.pdf>. Acesso em: 06 jun.2019.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. In: **Advances in neural information processing systems**. [s.n.], 2014. p. 3104–3112. Disponível em: <https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf>. Acesso em: 06 jun.2019.

TALATHI, S. S.; VARTAK, A. Improving performance of recurrent neural network with relu nonlinearity. **arXiv preprint arXiv:1511.03771**, 2015. Disponível em: <https://arxiv.org/pdf/1511.03771.pdf>. Acesso em: 06 jun.2019.

TRASK, A.; MICHALAK, P.; LIU, J. sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings. **arXiv preprint arXiv:1511.06388**, 2015. Disponível em: <https://arxiv.org/pdf/1511.06388.pdf>. Acesso em: 06 jun.2019.

TU, Z. et al. Modeling coverage for neural machine translation. **arXiv preprint arXiv:1601.04811**, 2016. Disponível em: <https://arxiv.org/pdf/1601.04811.pdf>. Acesso em: 06 jun.2019.

VINYALS, O.; FORTUNATO, M.; JAITLY, N. Pointer networks. In: **Advances in Neural Information Processing Systems**. [s.n.], 2015. p. 2692–2700. Disponível em: <https://papers.nips.cc/paper/5866-pointer-networks.pdf>. Acesso em: 06 jun.2019.

VÖLSKE, M. et al. Tl; dr: Mining reddit to learn automatic summarization. In: **Proceedings of the Workshop on New Frontiers in Summarization**. [s.n.], 2017. p. 59–63. Disponível em: <https://www.aclweb.org/anthology/W17-4508>. Acesso em: 06 jun.2019.

WONG, K.-F.; WU, M.; LI, W. Extractive summarization using supervised and semi-supervised learning. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1**. 2008. p. 985–992. Disponível em: <https://www.aclweb.org/anthology/C08-1124>. Acesso em: 06 jun.2019.