



Universidade de Brasília - UnB
Faculdade de Tecnologia
Engenharia Elétrica

***Deep Learning* para Processamento de
Linguagem Natural: Extração de Significado em
Redes Sociais**

Autor: Gabriel Bayomi Tinoco Kalejaiye
Orientador: Prof. Alexandre R. S. Romariz

Brasília, DF
2017



Gabriel Bayomi Tinoco Kalejaiye

***Deep Learning* para Processamento de Linguagem
Natural: Extração de Significado em Redes Sociais**

Monografia submetida ao curso de graduação em (Engenharia Elétrica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Elétrica).

Universidade de Brasília - UnB

Faculdade de Tecnologia

Orientador: Prof. Alexandre R. S. Romariz

Brasília, DF

2017

Gabriel Bayomi Tinoco Kalejaiye

***Deep Learning* para Processamento de Linguagem Natural: Extração de Significado em Redes Sociais**

Monografia submetida ao curso de graduação em (Engenharia Elétrica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Elétrica).

Trabalho aprovado. Brasília, DF, 07 de Julho de 2017:

Prof. Alexandre R. S. Romariz
Orientador

Prof. Daniel Guerreiro
Convidado

Prof. João Paulo Leite
Convidado

Brasília, DF
2017

Este trabalho é dedicado à minha mãe, que independente de qualquer dificuldade pessoal, sempre esteve plenamente disponível para apoiar, ajudar e entregar amor aos seus filhos.

Agradecimentos

Seria impossível compreender a lista de agradecimentos em apenas uma página; foram infinitos colegas, professores, familiares e amigos que fizeram parte dessa caminhada e a todas elas refiro o meu imensurável agradecimento.

Entretanto, um grupo específico de companheiros de curso moldou toda a minha história de graduação, criando amizades que durarão uma vida inteira.

Portanto, um agradecimento especial ao eterno grupo:

- *André Costa*

- *Arthur Carvalho*

- *Gabriel Castellano*

- *Gustavo Cid*

- *Henrique Orefice*

- *João Antonio*

- *Leonardo Albuquerque*

- *Leticia Brito*

- *Luiz Felipe*

- *Pedro Campos*

- *Renata Lopes*

- *Stefano Dantas*

Resumo

O Processamento de Linguagem Natural é uma área de intenso crescimento e interesse acadêmico, principalmente com as recentes evoluções nas capacidades computacionais disponíveis. Devido ao enorme número de exceções, nuances e complexidades de língua, a exploração automática de significados em textos não é uma tarefa trivial. Em ambientes modernos como redes sociais, essa problemática é amplificada, já que a informação, em geral, se encontra de maneira não-estruturada. Entretanto, a partir do uso de redes neurais artificiais, é possível criar aplicações inteligentes que possuem a capacidade de avaliar informação textual automaticamente. Esse projeto explora três diferentes áreas dentro da interseção de *Deep Learning* e Processamento de Linguagem Natural. Para cada uma dessas áreas, uma rede social diferente foi escolhida e os diferentes parâmetros comparados. O processo de aprendizado de análise sentimental da rede IMDB se mostrou superior às metas estabelecidos, chegando a 85% de acurácia. Na aplicação de normalização de perguntas da rede social Quora, mesmo sem *benchmarks* estabelecidos, foram comparados diferentes modelos para a avaliação de normalização. Por fim, a aplicação de um *chatbot*, a partir de informações do Twitter e de conjuntos de dados pré-estabelecidos, mostrou-se interessante, porém longe ainda de um modelo que de fato pode ser confundido com um ser humano.

Palavras-chaves: rede neural recorrente, processamento de linguagem natural, *deep learning*, redes sociais

Abstract

Natural Language Processing is one of the fastest growing areas of computer science, specially with the recent evolution of computational processing techniques and equipments. Due to the tremendous amount of exceptions, rules and complexities, the automatic exploration of meaning in textual information is a nontrivial task. Social networks usually add another layer of complexity to the problem, due to the high level of unstructured data available. However, artificial neural networks are usually a great tool to derive meaning from unstructured information. This project intends to explore three different areas on the intersection between Deep Learning and Natural Language Processing. For each area, a different application was built, using a different social network as background. The Sentimental Analysis model built with IMDB data was superior than the Kaggle benchmark, reaching 86% of accuracy. The model for the question normalization task of Quora, even without pre-established benchmarks, were compared between different vector-comparison strategies. Finally, a chatbot application, using Twitter and public data sets, had interesting results, but still far from a human-like model.

Key-words: natural language processing, deep learning, recurrent neural networks, social networks.

Lista de ilustrações

Figura 1	– Crescimento do uso de <i>Internet</i> no mundo. Fonte: Daze Info	17
Figura 2	– Crescimento do armazenamento de dados no mundo. Fonte: Oracle	18
Figura 3	– Crescimento do uso de <i>Deep Learning</i> no Google. Fonte: Google Inc	19
Figura 4	– Ilustração de Perceptron Simples. Fonte: <i>Nature of Code</i>	24
Figura 5	– Representação gráfica de sigmóides. Fonte: sebastianraschka.com	25
Figura 6	– Feed Forward Neural Network: exemplificação. Fonte: Duanqu Tech	26
Figura 7	– Representação da descida do Gradiente. Fonte: sebastianraschka.com	28
Figura 8	– <i>Underfitting/Overfitting</i> . Fonte: kevinbinz.com	28
Figura 9	– Modelo sintático clássico de NLP. Fonte: Stanford University CS224d	29
Figura 10	– Representação gráfica de um mapeamento semântico de palavras. Fonte: Stanford University CS224d.	31
Figura 11	– Word2Vec: Algoritmo para janela de vizinhança. Fonte: Chris McCormick.	32
Figura 12	– Word2Vec: Modelo de Feed Forward Neural Network. Fonte: Chris McCormick.	33
Figura 13	– Word2Vec, matriz para atribuição de valores. Fonte: Chris McCormick.	33
Figura 14	– Modelo simples de RNN. Fonte: Christopher Olah.	34
Figura 15	– Modelo genérico de RNN. Fonte: Christopher Olah.	35
Figura 16	– Modelo LSTM para RNN. Fonte: Christopher Olah.	35
Figura 17	– Representação da Cell State. Fonte: Christopher Olah.	35
Figura 18	– Representação de Gates. Fonte: Christopher Olah.	36
Figura 19	– LSTM operação do cell state. Fonte: Christopher Olah.	36
Figura 20	– LSTM operação do cell state. Fonte: Christopher Olah.	36
Figura 21	– LSTM operação do cell state. Fonte: Christopher Olah.	37
Figura 22	– Saída da rede LSTM. Fonte: Christopher Olah.	37
Figura 23	– Representação do modelo Seq2Seq. Fonte: Christopher Olah.	38
Figura 24	– Típico <i>Review</i> do portal IMDB, Screenshot de tela.	39
Figura 25	– Exemplo do banco de dados IMDB	40
Figura 26	– Mapeamento de significado em <i>Word2Vec</i> , Screenshot de tela.	42
Figura 27	– Teste do modelo <i>Word2Vec</i>	43
Figura 28	– Teste do modelo <i>Word2Vec</i>	43
Figura 29	– Comparação entre métodos, IMDB	43
Figura 30	– Comparação entre <i>benchmark</i> e modelo, IMDB	44
Figura 31	– Aplicativo desenvolvido, IMDB	44
Figura 32	– Resultado correto, IMDB	45
Figura 33	– Resultado correto, IMDB	45

Figura 34 – Resultado correto gerado	45
Figura 35 – Resultado incorreto, IMDB	46
Figura 36 – Exemplo de <i>post</i> do Quora, Screenshot de tela.	47
Figura 37 – Exemplo de entrada no banco de dados	48
Figura 38 – Similaridade de cosseno	49
Figura 39 – Distância Euclidiana	49
Figura 40 – Entropia Cruzada	49
Figura 41 – Comparação entre métodos por entropia cruzada, Quora	51
Figura 42 – Comparação entre métodos por acurácia, Quora	51
Figura 43 – Aplicação desenvolvida, Quora	52
Figura 44 – Resultado correto, Quora	52
Figura 45 – Resultado correto, Quora	53
Figura 46 – Resultado incorreto, Quora	53
Figura 47 – Clickthrough Rate por Celebridade. Fonte: Twitter Inc.	55
Figura 48 – Exemplo de Chatbot. Fonte: Nordstrom.	56
Figura 49 – <i>Web Scraping</i> para o Twitter	59
Figura 50 – Tweepy API	59
Figura 51 – Limpeza de informação do Tweet	59
Figura 52 – Tweet Original	60
Figura 53 – Tweet após a limpeza	60
Figura 54 – Aplicação em processo de loading.	62
Figura 55 – Cornell Movie Dialogs - Exemplo 1	62
Figura 56 – Cornell Movie Dialogs - Exemplo 2	63
Figura 57 – Twitter Aleatório 7k - Exemplo 1	63
Figura 58 – Twitter Aleatório 7k - Exemplo 2	64
Figura 59 – Twitter Celebridades 400-3k - Exemplo 1	64
Figura 60 – Twitter Celebridades 400-3k - Exemplo 2	65
Figura 61 – Comparação Bot, Filmes - Exemplo 1	66
Figura 62 – Comparação Bot, Filmes - Exemplo 2	67
Figura 63 – Comparação Bot, Twitter - Exemplo 1	67
Figura 64 – Comparação Bot, Twitter - Exemplo 2	68

Sumário

1	INTRODUÇÃO	17
1.1	Contextualização e Motivação	17
1.1.1	<i>Deep Learning</i>	18
1.1.2	Processamento de Linguagem Natural (NLP)	19
1.2	Objetivos do projeto	19
1.3	Definições do projeto	20
1.3.1	Extração de significado	20
1.3.2	IMDB e Análise Sentimental	20
1.3.3	Quora e Normalização Textual	20
1.3.4	<i>Twitter e Chat Bots</i>	21
2	FUNDAMENTOS DE REDES NEURAS ARTIFICIAIS	23
2.1	Redes Neurais Artificiais	23
2.2	Perceptron: uma ANN simples	23
2.3	<i>Feed Forward Neural Networks (FFNN): perceptron com múltiplas camadas</i>	25
2.3.1	<i>Feed Forward</i>	26
2.3.2	<i>Backpropagation: otimização, underfitting e overfitting</i>	27
3	DEEP LEARNING PARA PROCESSAMENTO DE LINGUAGEM NATURAL (NLP)	29
3.1	Princípios de NLP	29
3.2	Avanços de NLP com <i>Deep Learning</i>	30
3.3	<i>Word Embeddings</i>	30
3.4	<i>Word2Vec</i>	31
3.4.1	Treinamento do modelo	32
3.5	<i>Rede Neural Recorrente (RNN): aprendizado com memória</i>	33
3.6	Long Short Term Memory (LSTM) Networks	34
3.7	Seq2Seq: Machine Translation com RNNs	37
4	DESENVOLVIMENTO: IMDB DATA SET	39
4.1	Contextualização	39
4.1.1	Descrição do banco de dados	39
4.1.2	Análise Sentimental	40
4.2	Benchmarking	40
4.3	Metodologia	40

4.4	Especificação técnica	41
4.5	Resultados	42
4.5.1	<i>Word Embeddings</i>	42
4.5.2	Resultados finais e comparações	43
4.5.3	Aplicação desenvolvida	44
4.5.4	Exemplificação de resultados corretos	44
4.5.5	Exemplificação de resultados incorretos	46
5	DESENVOLVIMENTO: QUORA DATA SET	47
5.1	Contextualização	47
5.1.1	Descrição do banco de dados	47
5.1.2	Normalização	48
5.2	Comparação vetorial	48
5.3	Metodologia e Entropia Cruzada	49
5.4	Especificação técnica	50
5.5	Resultados	50
5.5.1	Resultados finais e comparações	50
5.5.2	Aplicação desenvolvida	51
5.5.3	Exemplificação de resultados corretos	52
5.5.4	Exemplificação de resultados incorretos	53
6	DESENVOLVIMENTO: TWITTER CHAT BOT	55
6.1	Contextualização	55
6.1.1	Twitter	55
6.1.2	Chatbots	56
6.1.3	Modelos de Chatbots	56
6.2	Benchmark e Exemplo do Google	57
6.3	Metodologia	58
6.4	Especificação técnica	60
6.5	Resultados	61
6.5.1	Cornell Movie Dialogs	62
6.5.2	Twitter Aleatório 7k	63
6.5.3	Twitter Celebidades 400-3k	64
6.5.4	Comparação de métodos	65
7	CONCLUSÕES	69
	REFERÊNCIAS	71

APÊNDICES	73
APÊNDICE A – CÓDIGO - SCRAPER PARA INFORMAÇÕES DO TWITTER - BOT	75
APÊNDICE B – CÓDIGO - IMDB	83
APÊNDICE C – CÓDIGO - QUORA	91

1 Introdução

1.1 Contextualização e Motivação

O advento de técnicas de aprendizado computacional possibilitaram a criação de modelos computacionais robustos que garantem grande eficiência para a resolução de problemas complexos. Por mais que muitos métodos não sejam necessariamente novos, se comparados a outras áreas da tecnologia, o crescimento exponencial das bases de dados disponíveis conferiram uma nova possibilidade na área de sistemas inteligentes: criar correlações e extrair significados de extensas estruturas de informação. Um dos fatores que mais contribuiu para a expansão da quantidade de dados disponíveis é o uso acelerado da *Internet* nos últimos anos (SHAPIRO; GENTZKOW; BOXELL, 2017):

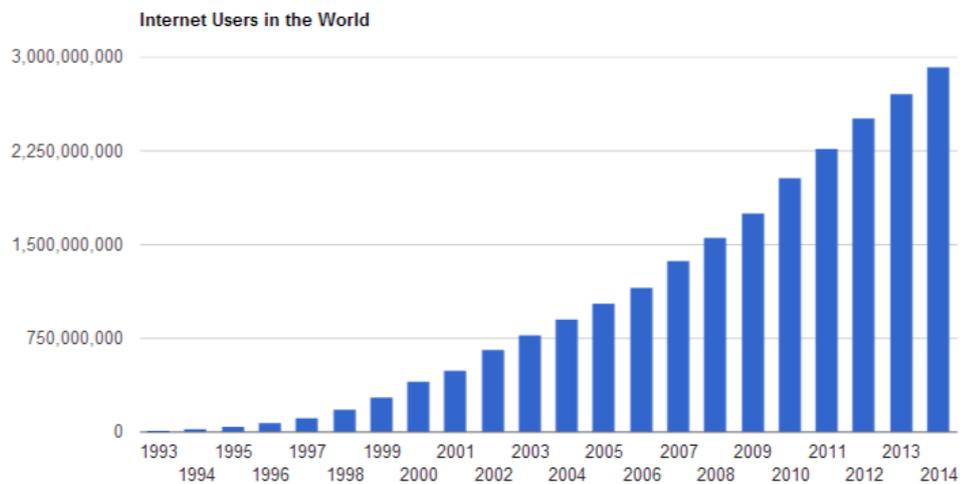


Figura 1 – Crescimento do uso de *Internet* no mundo. Fonte: Daze Info

Esse aumento do crescimento de capacidade de dados foi acompanhado pelo crescimento relativo de dados não-estruturados, que são os conjuntos de dados não relacionais, como documentos textos, imagens, vídeos, etc.

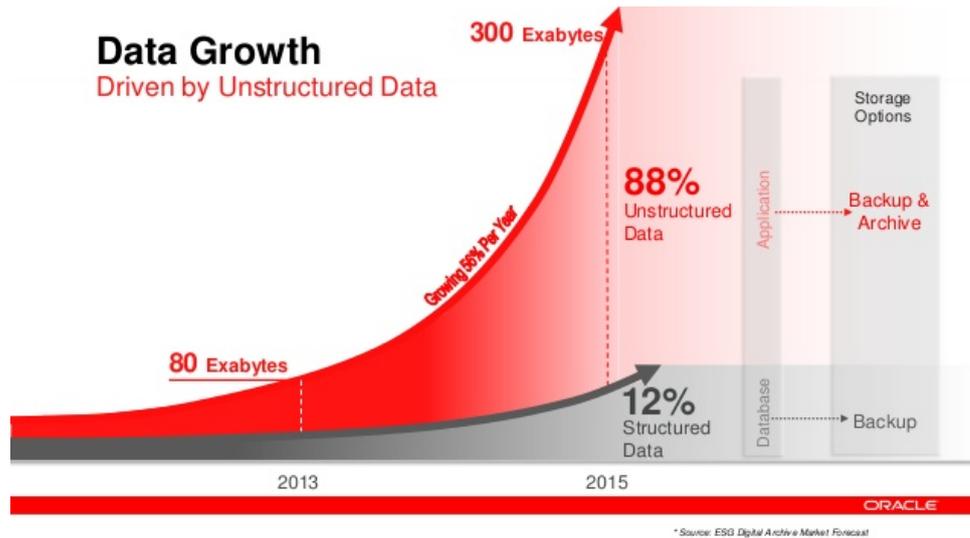


Figura 2 – Crescimento do armazenamento de dados no mundo. Fonte: Oracle

Entre a gama quase infinita de possíveis dados não-estruturados, as informações textuais são uma das mais interessantes a serem analisadas: são complexas, possuem nuances e duplos significados e também necessitam de contextualização.

Quando adiciona-se o contexto de redes sociais ao de dados textuais, a situação é ainda mais caótica, já que é necessário lidar com erros gramaticais, elevado nível de sarcasmo, gírias e outros tipos de desorganização estrutural. Este trabalho tem como foco demonstrar como é possível extrair significado quantitativo de redes sociais, mesmo em ambientes desorganizados e complexos, a partir do Processamento de Linguagem Natural e de métodos de *Deep Learning*.

1.1.1 *Deep Learning*

Grande parte do crescimento da base de dados mundial está relacionado ao crescente uso de redes sociais. Várias novas informações são geradas a cada segundo, de forma desordenada. Felizmente, modelos de *Deep Learning* são especialmente poderosos para desafios como esses. A partir do uso de funções não-lineares, problemas não-triviais com variáveis não linearmente separáveis são passíveis de serem solucionados (DU; SHANKER, 2013).

Deep Learning normalmente é definido como o uso de Redes Neurais Artificiais com múltiplas camadas de profundidade. Esses modelos não são necessariamente novos, mas ganharam uma grande quantidade de novos adeptos industriais e acadêmicos todos os anos, com o aumento radical de dados disponíveis para criação de modelos e a capacidade computacional para processá-los. Podemos verificar o crescimento do uso de *Deep Learning* no Google para os últimos anos:

Growing Use of Deep Learning at Google

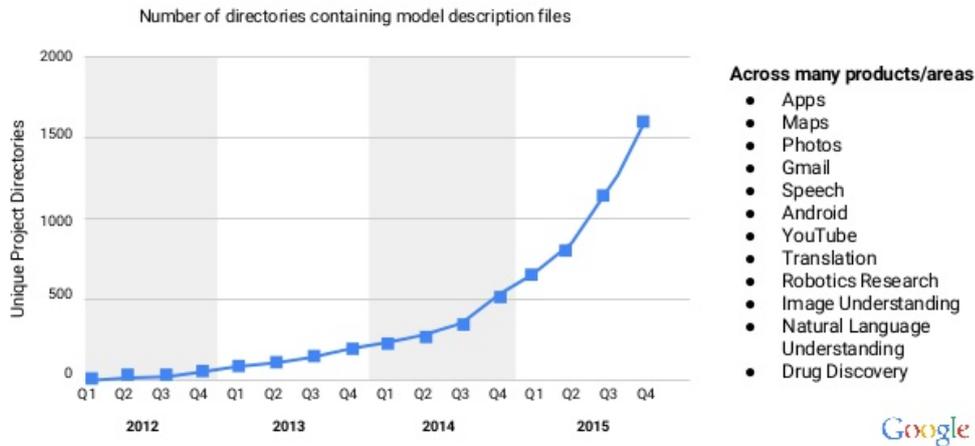


Figura 3 – Crescimento do uso de *Deep Learning* no Google. Fonte: Google Inc

Posteriormente, neste trabalho, serão descritas as principais características e exemplos de redes neurais artificiais utilizadas para deep learning.

1.1.2 Processamento de Linguagem Natural (NLP)

Processamento de Linguagem Natural (NLP) consiste no desenvolvimento de modelos computacionais para a realização de tarefas que dependem de dados expressos em alguma língua natural. Com a recente utilização intensiva dos modelos de aprendizado de *Deep Learning*, o estudo acadêmico de NLP foi fortemente modificado. Para a criação de aplicações de NLP, normalmente, utilizavam-se estruturas sintáticas, estruturais e morfológicas para derivar modelos computacionais capazes de gerar significado (PEREIRA, 2013). Entretanto, com o grande número de possíveis exceções e a quantidade massiva de dados textuais não estruturados (*tweets, posts de Facebook, etc*), o desenvolvimento de alguns desses modelos se tornaram impraticáveis.

Entretanto, novamente, as técnicas de *Deep Learning* são de grande valia para tratar de dados complexos e não estruturados. As principais técnicas de *Deep Learning* para NLP serão destacadas e exemplificadas em aplicações neste trabalho.

1.2 Objetivos do projeto

- *Explorar* as técnicas modernas de Deep Learning, a partir da literatura acadêmica da área.
- *Compreender* a forma como as técnicas de aprendizado extraem significado no Processamento de Linguagem Natural.

- *Comparar* técnicas de aprendizado em termos de precisão e eficiência.
- *Explorar* a forma como as redes sociais influenciam a análise de dados na *Internet*.
- *Discutir* os temas mais importantes da literatura moderna em Deep Learning para NLP.
- *Criar* uma aplicação capaz de realizar análise textual binária para Análise Sentimental e Normalização, a partir de redes sociais.
- *Gerar* um *chatbot* a partir de informações de redes sociais.

1.3 Definições do projeto

1.3.1 Extração de significado

A missão principal deste projeto é conseguir extrair significado de redes sociais a partir de redes neurais artificiais. Este trabalho foi dividido em três diferentes técnicas para extração de significado e, para cada técnica, uma diferente rede social foi selecionada. Esta divisão é relacionada a seguir.

1.3.2 IMDB e Análise Sentimental

IMDB é a maior rede social de filmes e seriados do mundo. Apesar de ser menos interativa do que algumas outras mais populares, como o Facebook ou o Twitter, existe a possibilidade de usuários avaliarem e comentarem sobre seus filmes e seriados. A partir de uma análise de várias avaliações feitas por usuários, seria possível, automaticamente, declarar se a avaliação foi positiva ou negativa? Esse é o campo de estudo de Análise Sentimental que serve para compreender, a partir de dados textuais, o significado relacionado a uma sentença, parágrafo ou texto. Este estudo será apresentado, metodologicamente, neste trabalho.

1.3.3 Quora e Normalização Textual

Quora é uma rede social de perguntas e respostas. Usuários podem responder, perguntar, avaliar respostas, sugerir modificações e realizar comentários. Entretanto, muitas vezes as perguntas são frequentemente repetidas. Seria possível, automaticamente, compreender o contexto específico de uma pergunta e compará-lo a outra, de forma a resolver esse problema? O estudo de como obter a normalização de perguntas similares é o segundo ponto tratado neste trabalho.

1.3.4 *Twitter e Chat Bots*

Criar *Chat Bots* automáticos é uma das aplicações mais recentemente exploradas pelo meio industrial e acadêmico na área de Processamento de Linguagem Natural. A ideia de simular a inteligência humana em perguntas e respostas é antiga, mas a capacidade de criar modelos minimamente capazes de fazê-lo é extremamente recente. A ideia principal da terceira parte do desenvolvimento do projeto é fazer a coleta de dados do *Twitter* para criar um *bot* que consiga, organicamente, realizar conversas com um ser humano.

2 Fundamentos de Redes Neurais Artificiais

Este capítulo contém os fundamentos necessários sobre redes neurais artificiais para a compreensão das técnicas adotadas no projeto.

2.1 Redes Neurais Artificiais

Uma rede neural artificial (ANN) é um paradigma de processamento de informação, inspirado na biologia do sistema nervoso humano. A ideia central é simular, de maneira computacional, a forma como o cérebro processa e retém informação. A característica marcante de redes neurais artificiais é a habilidade de extrair significado de complicadas e incompletas estruturas de dados. Essas redes são organizadas em camadas de neurônios que são as células primárias de uma ANN. Cada uma dessas células é representada matematicamente por uma função de ativação em relação a suas entradas. A interconexão entre neurônios, o processo de aprendizado e a conversão dos valores de entrada de cada neurônio para uma função de ativação final são características fundamentais desses modelos (MAIND; WANKAR, 2014). As vantagens e características principais de redes neurais artificiais, em relação à habilidade de interpretar informação, são:

- **Aprendizado adaptativo:** a capacidade de aprender a partir de treinamento baseado em tentativas anteriores ou modelos previamente conhecidos.
- **Auto-organização:** redes neurais podem criar seu próprio sistema de representação e informação durante o aprendizado .
- **Operações em tempo real em paralelo:** a paralelização de redes neurais é prática e fácil comparada a outros métodos de processamento de informação.

2.2 Perceptron: uma ANN simples

O perceptron simples é uma das melhores representações didáticas para os fundamentos de funcionamentos de uma rede neural. Para um determinado problema matemático, onde se sabe que para um vetor de entrada X , teremos uma saída y , o perceptron de camada única consiste, basicamente, de uma função φ , tal que \tilde{y} , dado por

$$\tilde{y} = \varphi\left(\sum_{i=1}^K w_i x_i + b\right) \quad (2.1)$$

$$\tilde{y} = \varphi(W^T X + b) \quad (2.2)$$

φ é uma previsão de y . W representa a matriz de pesos para cada entrada do vetor X . É adicionado também um valor b , chamado de bias, tornando o modelo mais robusto. A ideia central é que, a partir do conhecimento de pares de vetores X_{old} e suas respostas y_{old} no passado, é possível, para um novo vetor X_{new} , encontrar uma previsão adequada $\tilde{y} \approx y_{new}$. Isso pode ser representado graficamente da seguinte forma:

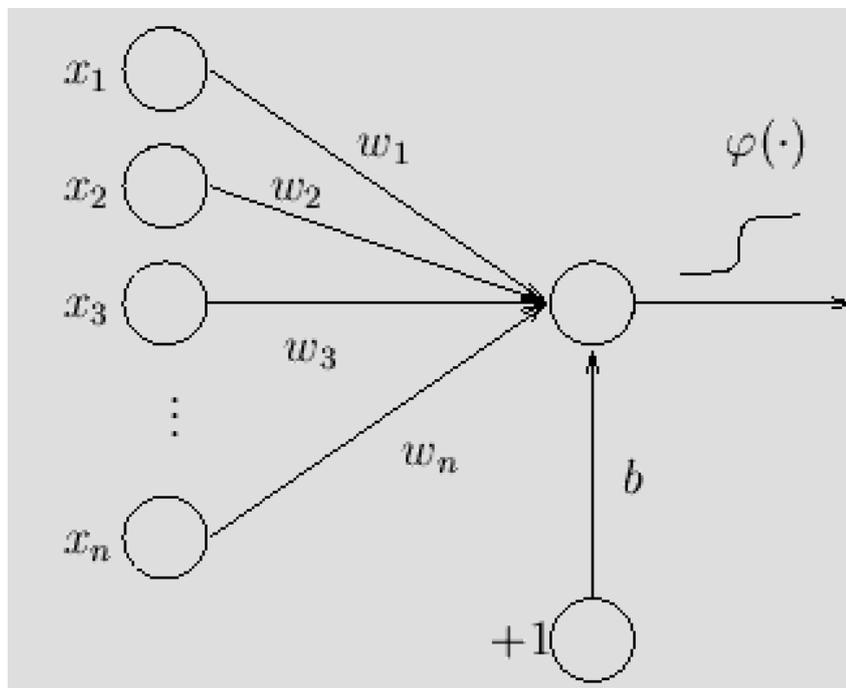


Figura 4 – Ilustração de Perceptron Simples. Fonte: *Nature of Code*

A partir desse modelo e escolhendo uma função φ adequada, o problema computacional se resume em encontrar os valores da matriz W e de b que otimizam o valor de \tilde{y} para um determinado problema.

Em um exemplo prático, pode-se considerar uma problemática em que é necessário decidir se um determinado livro será bom ($y=1$) ou ruim ($y=0$) para um determinado leitor, baseado nas avaliações prévias desse mesmo usuário de vários outros livros. Para isso, seria possível transformar as características de cada livro (tanto dos anteriores como do novo) em um vetor X . Posteriormente, seria utilizado o conhecimento prévio da avaliação dos livros já lidos para adaptar a matriz W de forma adequada, focando em garantir que \tilde{y} seja próximo do y já conhecido para os casos anteriores. Depois, o modelo gerado poderá avaliar conjuntos novos de livros.

2.3 Feed Forward Neural Networks (FFNN): perceptron com múltiplas camadas

Apesar da importância do perceptron simples para a fundamentação e entendimento de redes neurais, esse modelo é pouco efetivo para resolução de problemas mais complexos. Entretanto, é possível ampliar o modelo, utilizando a mesma lógica fundamental, para construir um perceptron de múltiplas camadas (ZAKARIA; WANKAR, 2014).

Para isso, é necessário entender a melhor forma de escolher funções de ativação adequadas. A função de ativação φ pode variar, dependendo da necessidade do modelo computacional, mas, em geral, uma função não-linear é escolhida, de forma a se tornar um instrumento mais poderoso para resolução de problemas não-triviais.

Como o próprio nome já diz, essas funções, normalmente, simulam a ativação (1) ou não-ativação (0 ou -1) de um nó. Essa característica pode ser vista em várias das funções não-lineares utilizadas em redes neurais artificiais. Exemplos clássicos são as funções sigmóides, como:

- Função Logística:

$$\text{logit}(x) = \varphi_{\text{log}}(x) = \frac{1}{1 + e^{-x}} \quad (2.3)$$

- Tangente Hiperbólica:

$$\text{tanh}(x) = \varphi_{\text{tanh}}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.4)$$

O comportamento delas é descrito graficamente, como:

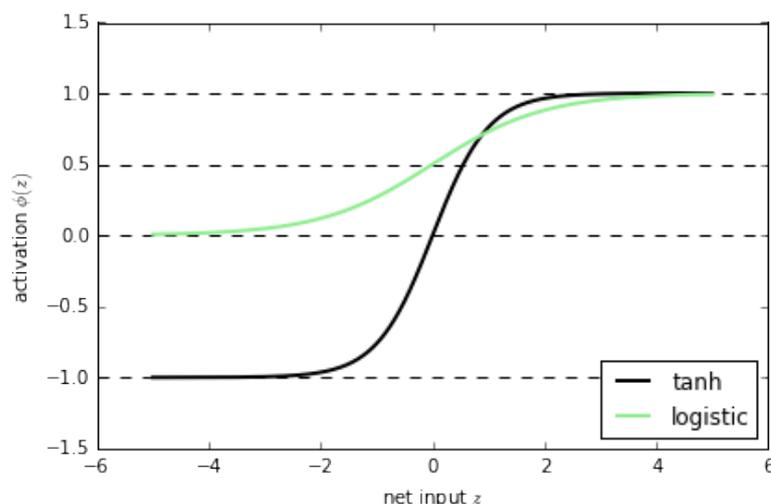


Figura 5 – Representação gráfica de sigmóides. Fonte: sebastianraschka.com

A FFNN consiste, basicamente, em se apropriar da característica de ativação de um nó (perceptron simples) e utilizá-lo como entrada para uma nova camada de nós idênticos. Dessa forma, o modelo será formado por uma série de nós em paralelo, fazendo com que o vetor de entrada X passe por várias camadas de funções de ativação até chegar ao seu valor de previsão final. As camadas intermediárias são referenciadas como camada escondida ou *hidden layers* e a camada final como camada de saída ou *output layer*, que representa o valor \hat{y} . Essa ideia pode ser ilustrada com o desenho a seguir:

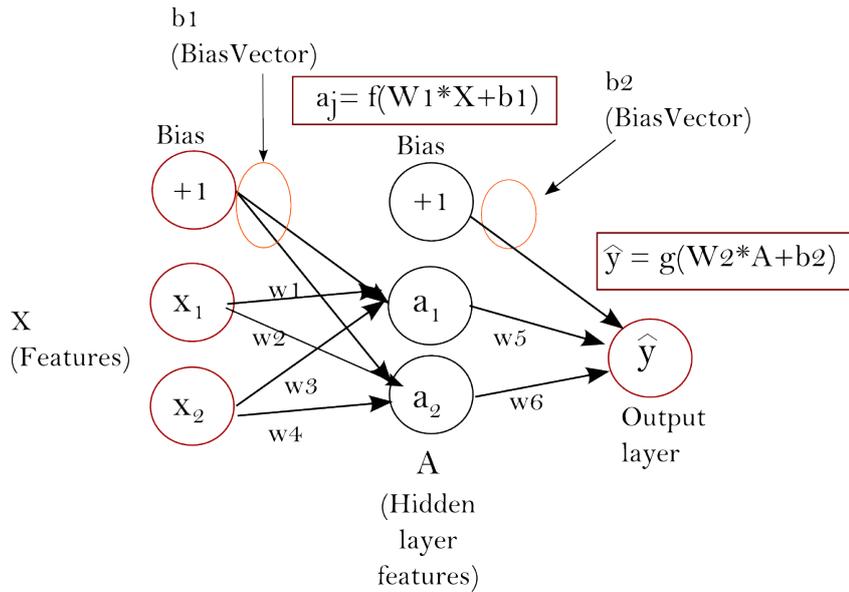


Figura 6 – Feed Forward Neural Network: exemplificação. Fonte: Duanqu Tech

É possível dividir o processo de aprendizado em redes supervisionadas em duas etapas: *Feed Forward* e *Back Propagation*. Na etapa de *Feed Forward*, encontra-se uma previsão de valor de classe y para uma entrada X . Na etapa de *Back Propagation*, é comparada a saída real com a saída prevista e corrigidos os valores de W a partir do erro entre a previsão e o valor real (MACUKOW, 2013).

2.3.1 Feed Forward

No modelo apresentado anteriormente, é estabelecido, em um nó k da camada escondida, com função de ativação φ para uma entrada X :

$$\tilde{h}_k = \varphi\left(\sum_{i=1}^N w_i^k x_i + b_k\right) \quad (2.5)$$

$$\tilde{h}_k = \varphi(W_k^T X + b) \quad (2.6)$$

E na entrada de cada nó da camada de saída, têm-se:

$$\widetilde{H} = [\tilde{h}_1, \tilde{h}_2, \dots, \tilde{h}_k] \quad (2.7)$$

$$\widetilde{H} = [\varphi(W_1^T X + b_1), \varphi(W_2^T X + b_2), \dots, \varphi(W_k^T X + b_k)] \quad (2.8)$$

E para um nó j da camada de saída \tilde{y} :

$$\tilde{y}_j = \varphi\left(\sum_{i=1}^K \tilde{w}_i \tilde{h}_i + b_j\right) \quad (2.9)$$

$$\tilde{y}_j = \varphi(\widetilde{W}_j^T \widetilde{H} + b_j) \quad (2.10)$$

A classe de saída será representada:

$$\tilde{y} = [\tilde{y}_1, \tilde{y}_2, \dots, \tilde{y}_j] \quad (2.11)$$

$$\tilde{y} = [\varphi(\widetilde{W}_1^T \widetilde{H} + b_1), \varphi(\widetilde{W}_2^T \widetilde{H} + b_2), \dots, \varphi(\widetilde{W}_j^T \widetilde{H} + b_j)] \quad (2.12)$$

A partir dessa configuração, é produzido um modelo mais complexo para obter previsões de valores com redes neurais. Novamente, ao se escolherem as funções φ adequadas, o problema computacional se resume em encontrar os valores da matriz W e de b que otimizem o valor de \tilde{y} . Porém, é menos intuitivo encontrar um método que otimize os valores de W . Para isso, o algoritmo de *backpropagation* é de grande valia nos estudos de ANNs e será apresentado a seguir.

2.3.2 Backpropagation: otimização, *underfitting* e *overfitting*

A etapa de *Feed Forward* cria uma previsão de resultado a partir das funções lineares intermediárias aplicadas na camada de entrada. A partir daí, durante a etapa de treinamento, o valor real é comparado com o valor previsto que resultará no sinal de erro. A partir do sinal de erro, será possível corrigir os valores da matriz W .

Cada nó contribuiu, de forma ponderada, para o erro final da rede. A etapa de *Backpropagation* define o quanto cada nó contribuiu para garantir a correção adequada dos valores da matriz W (MAZUR, 2015). Diversos métodos podem ser utilizados com esse intuito. A maioria desses métodos utiliza o conceito de Descida por Gradiente para realizar essa otimização.

A ideia básica da Descida por Gradiente é, a partir da definição da função de erro, encontrar o ponto de mínimo global da curva (DAO; VEMURI,). Ou seja, seguir a direção do gradiente, como é ilustrado na figura a seguir:

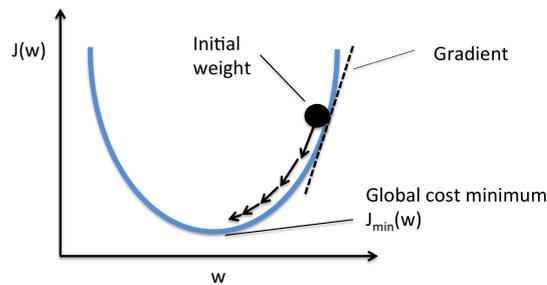


Figura 7 – Representação da descida do Gradiente. Fonte: sebastianraschka.com

A partir dessa lógica, cada método de otimização irá variar na forma de encarar o problema (DAO; VEMURI,). O método escolhido nesse trabalho foi o otimizador de Adam.

Underfitting/Overfitting

Um dos maiores perigos em modelos de aprendizado é adentrar em casos de *underfitting* ou *overfitting*. *Underfitting* ocorre quando a rede não é complexa o suficiente e o resultado final não consegue criar uma representação fidedigna da realidade.

Overfitting, por outro lado, ocorre quando o modelo é muito bem ajustado para os dados de treinamento, mas não consegue prever com acurácia qualquer conteúdo fora do escopo de treinamento.

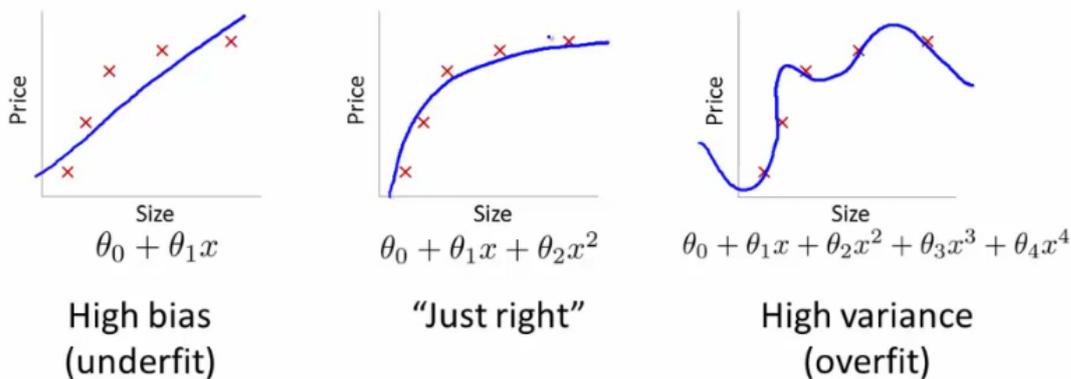


Figura 8 – *Underfitting/Overfitting*. Fonte: kevinbinz.com

Validação Cruzada

Uma técnica comum para evitar *underfitting/overfitting* é dividir os dados em treinamento e teste. Dessa forma, é possível treinar o modelo com a primeira parcela dos dados e depois testá-los na outra parcela, conferindo se houve *underfitting* ou *overfitting*. Caso positivo, os parâmetros do modelo devem ser alterados. Uma decisão comum de validação cruzada é de 80% para treinamento e 20% para testes.

3 *Deep Learning* para Processamento Natural de Linguagem (NLP)

Este capítulo contém os fundamentos necessários sobre o uso de *Deep Learning* com redes neurais para a compreensão das técnicas de processamento natural de linguagem adotadas no projeto.

3.1 Princípios de NLP

O campo de Processamento de Linguagem Natural (NLP) trata da forma como computadores podem analisar e extrair significado da linguagem natural humana. Isso pode ser feito de maneira distintas e, com o passar dos anos, a academia muda as diferentes formas e conceitos usados para abordar esses métodos, tanto do ponto de vista da linguística como do ponto de vista computacional. Em uma análise clássica de NLP, pode-se usar a morfologia, sintaxe e outras características para trazer significado matemático para textos. Dessa forma, o formato em que o texto está escrito interfere diretamente na forma como se captura o resultado. Pode-se obter diagramas, dividindo os diferentes segmentos de uma frase em suas propriedades gramaticais (SHAPIRO; GENTZKOW; BOXELL, 2017). Para a frase *'The first fifteen minutes were dry, but by the end I really enjoyed the lecture'*:

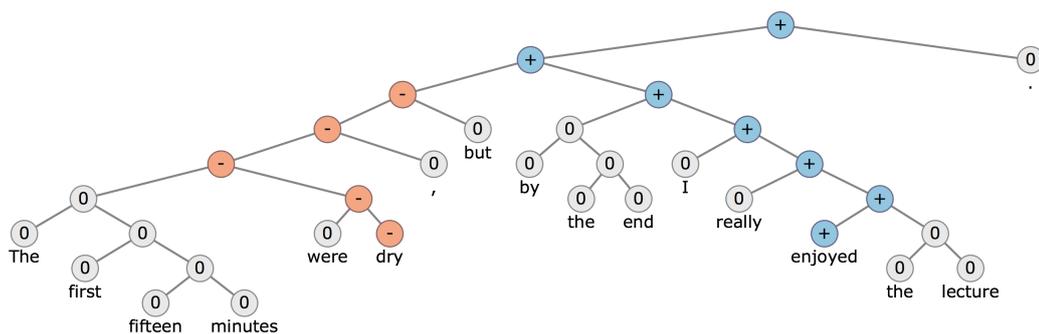


Figura 9 – Modelo sintático clássico de NLP. Fonte: Stanford University CS224d

Por mais que essa abordagem seja interessante, mesmo nas mais simples linguagens, é extremamente complicado obter divisões como essas e existem inúmeras exceções que dificultam imensamente o processo computacional de análise. Isso ajuda a concluir que métodos de aprendizado de máquinas (que funcionam muito bem para aprender sobre exceções) podem ser utilizados para a otimização de resultados.

3.2 Avanços de NLP com *Deep Learning*

A tendência atual em NLP é a utilização de Deep Learning para o entendimento de contextos e detalhes da comunicação humana, de forma automatizada. Atualmente, isso ocorre normalmente em três etapas diferentes. Primeiro, obtém-se o mapeamento de cada palavra em determinado contexto para um vetor multidimensional.

Segundo, transformam-se estruturas textuais (parágrafos, frases, etc) em um novo vetor a partir da combinação dos vetores das palavras que o compõe. Finalmente, têm-se a terceira etapa, onde o processo de aprendizado acontece e que pode ter vários objetivos finais diferentes, desde análise sentimental até classificação multinível (ou combinação de métodos diferentes, como será visto a seguir).

3.3 *Word Embeddings*

Representação de palavras é parte central do estudo do processamento natural de linguagem. Um *Word Embedding* se refere a um método de transformar um conjunto de palavras, um vocabulário, normalmente referido como *corpus*, em um conjunto-espelho de vetores (LEVY; GOLDBERG, 2014). A ideia principal é garantir que cada palavra tenha um vetor único referente a mesma. A partir do momento que cada palavra possui um vetor de referência, é possível moldar um modelo de aprendizado capaz de aprender e avaliar sentenças, parágrafos ou textos.

A forma mais trivial de se pensar em *word embeddings* é utilizar um método simples de *one-hot encoder* para cada palavra. Por exemplo, em um *corpus* de 5 palavras como 'sanduíche', 'garoto', 'pizza', 'atacar' e 'vida', seria possível criar uma representação da seguinte forma:

sanduíche = [1 0 0 0 0]

garoto = [0 1 0 0 0]

pizza = [0 0 1 0 0]

atacar = [0 0 0 1 0]

vida = [0 0 0 0 1]

Por mais que essa escolha garanta um vetor único para cada palavra, é fácil perceber que seria muito pouco efetivo para um *corpus* de um problema real, com um número elevado de palavras no vocabulário. Para um *corpus* de 10.000 palavras, seria necessário retirar palavras menos necessárias ou gerar vetores de 10.000 posições.

Portanto, é preciso definir uma estrutura mais complexa que consiga gerar uma representação única e, ao mesmo tempo, escalável para uma quantidade grande de palavras. Uma opção interessante seria acoplar o significado de cada palavra na definição de seu vetor. Palavras semanticamente parecidas, como 'macarrão' e 'pizza' teriam uma geometria vetorial parecida. Dessa forma, é possível resolver o problema de escalabilidade e, ao mesmo tempo, extrair significado de cada vetor representativo. Em outras palavras, os vetores podem traduzir contexto. Com esse objetivo, vários algoritmos foram desenvolvidos para focar nessa problemática. Entre eles, esse trabalho foca no mais popular entre esses algoritmos: *Word2Vec*.

3.4 *Word2Vec*

Word2Vec é um modelo que cria um espaço vetorial único de dimensão, normalmente, na casa de centenas de unidades. Esse modelo utiliza a ideia de coocorrência. Em outras palavras, palavras que usualmente se encontram próximas, terão geometria vetorial próxima. É possível verificar que esse modelo gera uma aproximação geométrica similar a aproximação semântica.

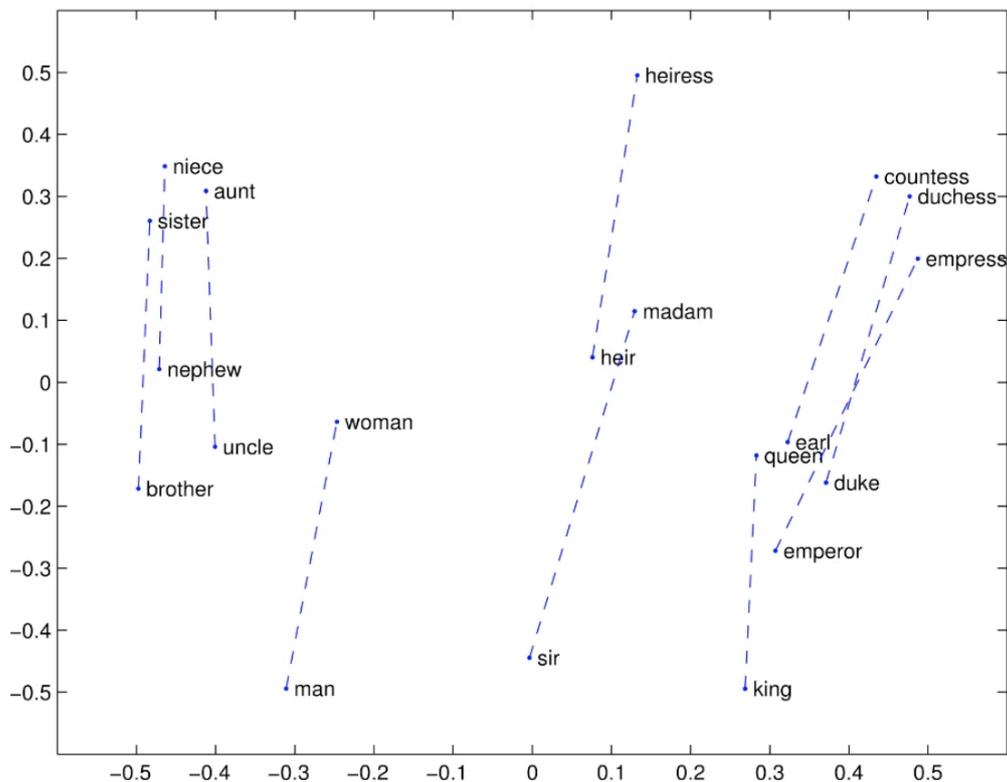


Figura 10 – Representação gráfica de um mapeamento semântico de palavras. Fonte: Stanford University CS224d.

Percebemos que distâncias geométricas similares possuem semânticas similares, deixando o modelo mais robusto.

Esse modelo utiliza o treinamento de uma rede neural FFNN. Porém, diferente de outros modelos, o objetivo é utilizar a matriz W de pesos como o espaço vetorial de representação de palavras (MCCORMICK, 2016).

3.4.1 Treinamento do modelo

Dado uma determinada palavra e sua vizinhança dentro de um *corpus* específico, é escolhida uma palavra aleatória vizinha. A rede irá definir qual a probabilidade para cada palavra do nosso vocabulário ser a palavra aleatoriamente escolhida.

O vetor de saída será relacionado com a probabilidade de encontrar cada palavra como vizinha de outra. Por exemplo, utilizando a palavra 'almoço' como entrada da rede, a probabilidade de se encontrar palavra 'comida' como vizinha será muito mais alta do que da palavra 'computador'. Isso será refletido nos vetores de saída da rede. Isso é possível a partir da introdução de pares de palavras, encontrados no corpus, como entradas da rede.

Considera-se um *corpus* constituído apenas da sentença: “The quick brown fox jumps over the lazy dog” e uma janela de vizinhança de 2 palavras. A partir disso, têm-se o seguinte modelo:

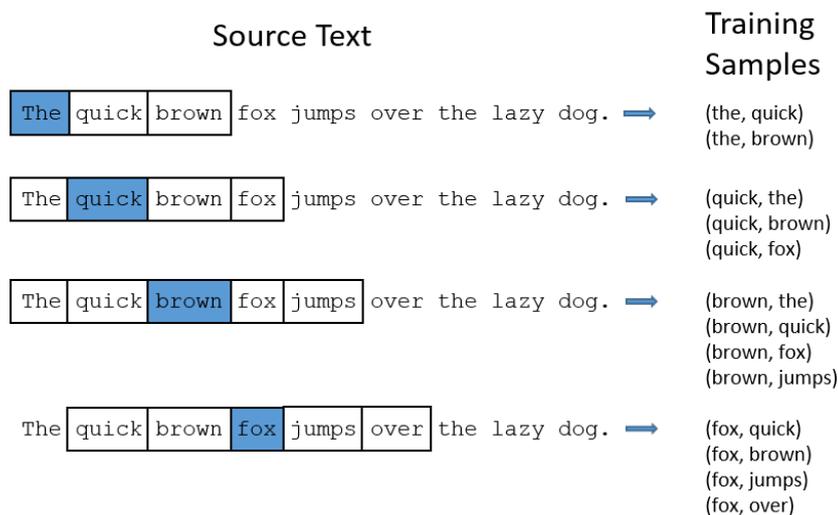


Figura 11 – Word2Vec: Algoritmo para janela de vizinhança. Fonte: Chris McCormick.

Anteriormente, foi descrito um método de one-hot encoder. Como, antes da criação do espaço vetorial *word2vec*, não temos uma representação adequada das palavras, é necessário utilizar o modelo *one-hot* para treinar a rede neural *word2vec*. Dessa forma, todo o treinamento posterior do modelo poderá usufruir de um *word embedding* mais adequado.

Considera-se, novamente, um vocabulário de 10.000 palavras. A partir desse vocabulário, criaremos um vetor *one-hot* para cada uma das palavras. Assim, monta-se o seguinte modelo FFNN:

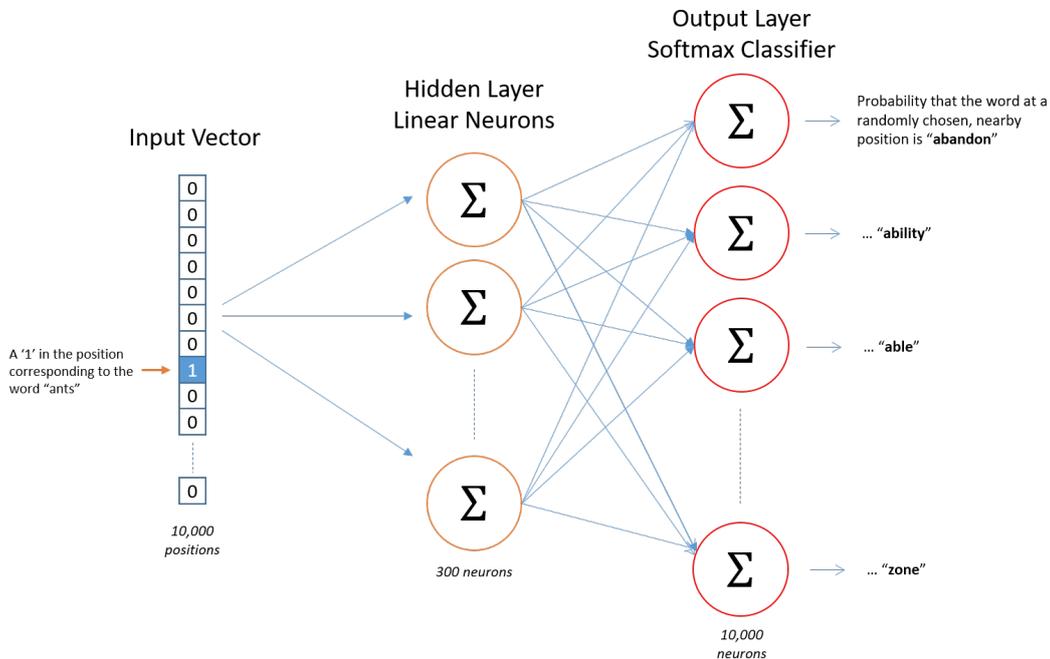


Figura 12 – Word2Vec: Modelo de Feed Forward Neural Network. Fonte: Chris McCormick.

Depois de treinado o modelo, é necessário recuperar o espaço vetorial da FFNN a partir do vetor peso W . Como todas as representações são *one-hot*, referencia-se apenas a parte da matriz relacionada com "1". Uma exemplificação simples encontra-se a seguir:

$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

Figura 13 – Word2Vec, matriz para atribuição de valores. Fonte: Chris McCormick.

3.5 Rede Neural Recorrente (RNN): aprendizado com memória

As outras propriedades descritas anteriormente também são válidas para redes neurais recorrentes. Entretanto, a principal diferença entre uma FFNN e uma RNN é a sua capacidade de guardar memória.

Pode-se imaginar um modelo onde é necessário classificar o que está acontecendo em um filme em cada momento que se passa. Não é claro como uma rede neural tradicional conseguiria utilizar informações anteriores sobre o filme para tratar informações futuras. Essa é a importância das RNNs criar um sistema de *feedback* capaz de criar um ciclo de informações persistentes, como ilustrado no diagrama a seguir:

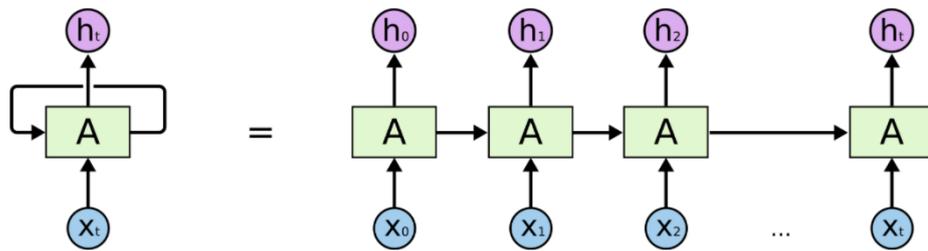


Figura 14 – Modelo simples de RNN. Fonte: Christopher Olah.

Essa estrutura demonstra que RNNs estão intimamente relacionadas com sequência e listas. Elas são a arquitetura natural a ser utilizada nesse tipo de estrutura. Neste trabalho, utilizaremos um subtipo de RNNs chamado de LSTM. Os detalhes sobre esse tipo de rede serão retratados mais a seguir.

3.6 Long Short Term Memory (LSTM) Networks

Um dos problemas de se trabalhar com linguagem é a criação de dependências de longo prazo, onde duas informações importantes para um determinado contexto estão distantes textualmente. RNNs, como visto anteriormente, possuem a capacidade de guardar memória. Em teoria, essas redes teriam a capacidade de resolver essa deficiência. Entretanto, percebe-se, na prática, RNNs clássicas não são efetivas nesse aspecto.

As redes LSTM foram introduzidas em 1997 (HOCHREITER; SCHMIDHUBER, 1997) e são um subtipo de RNNs que conseguem lidar com esse tipo de dependência de forma altamente eficiente. Todas as RNNs possuem uma estrutura de corrente que repete informações em *loop*. Porém, em RNNs tradicionais, essa repetição se refletirá em uma simples configuração de tangente hiperbólica, como por exemplo:

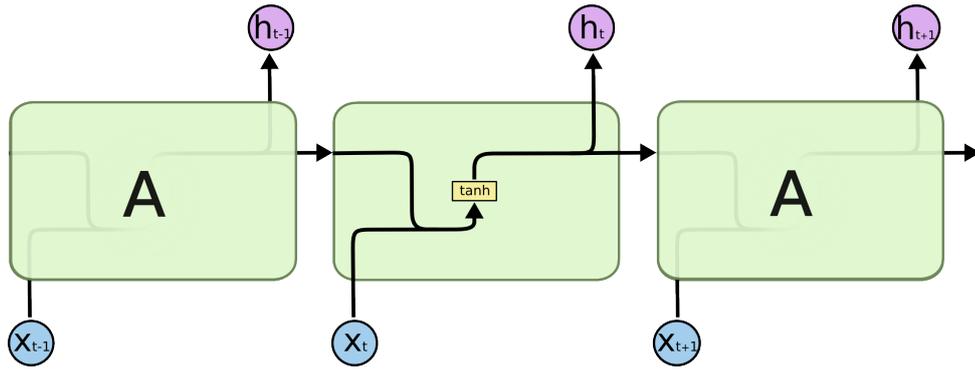


Figura 15 – Modelo genérico de RNN. Fonte: Christopher Olah.

LSTMs também possuem uma estrutura de *loop*, porém o módulo de repetição possui uma estrutura mais complexa. Como representado na figura seguinte:

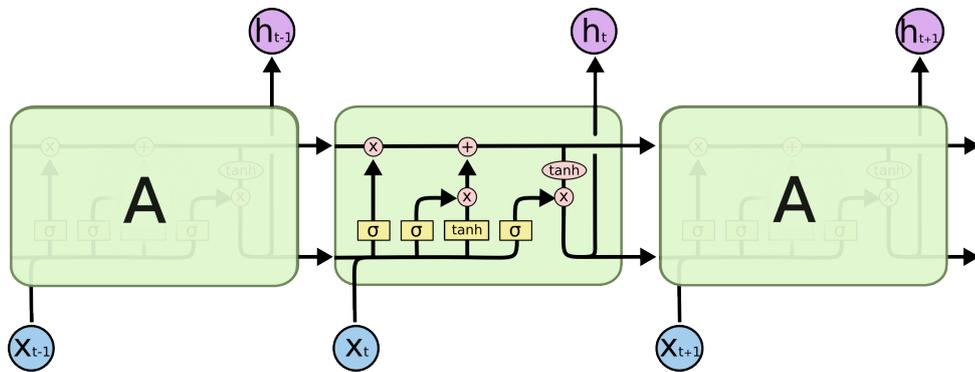


Figura 16 – Modelo LSTM para RNN. Fonte: Christopher Olah.

A característica principal de uma rede LSTM é o 'estado de célula', que é a linha horizontal no topo do diagrama (HOCHREITER; SCHMIDHUBER, 1997). O estado de célula facilita que a informação seja transmitida por toda a rede com poucas alterações importantes.

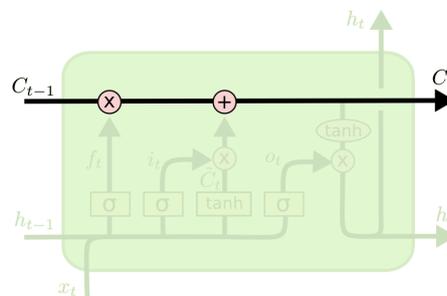


Figura 17 – Representação da Cell State. Fonte: Christopher Olah.

A LSTM pode remover ou adicionar informação para o estado de célula a partir de estruturas chamadas gates.

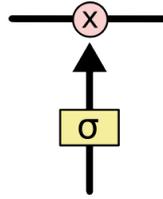


Figura 18 – Representação de Gates. Fonte: Christopher Olah.

A camada sigmoide descreve o quanto da informação é permitida a passar para os próximos estágios. O diagrama pode ser ilustrado dessa forma:

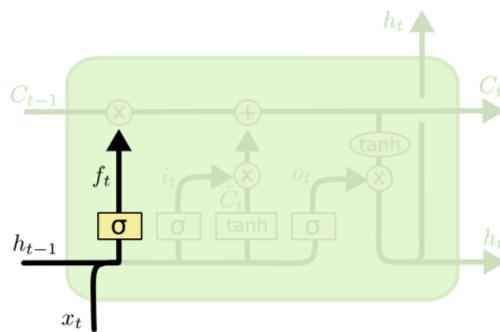


Figura 19 – LSTM operação do cell state. Fonte: Christopher Olah.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (3.1)$$

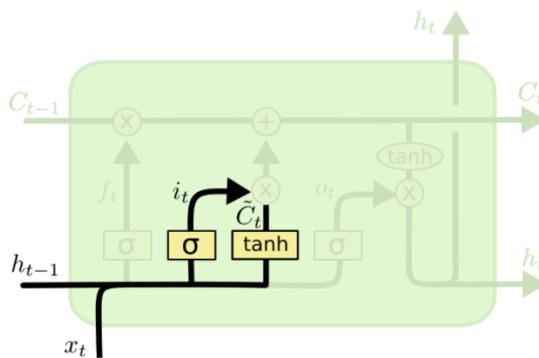


Figura 20 – LSTM operação do cell state. Fonte: Christopher Olah.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (3.2)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (3.3)$$

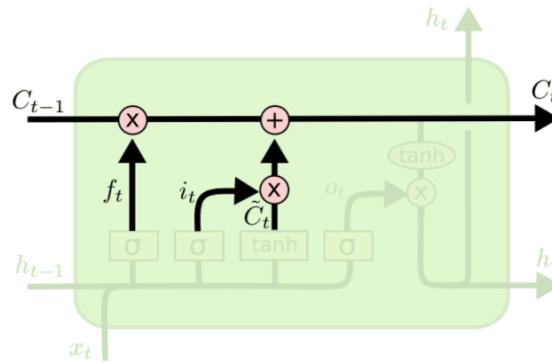


Figura 21 – LSTM operação do cell state. Fonte: Christopher Olah.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (3.4)$$

Isso garante que não só a saída h_t da célula, mas também informações mais antigas da rede (OLAH, 2015). A saída h_t é definida como:

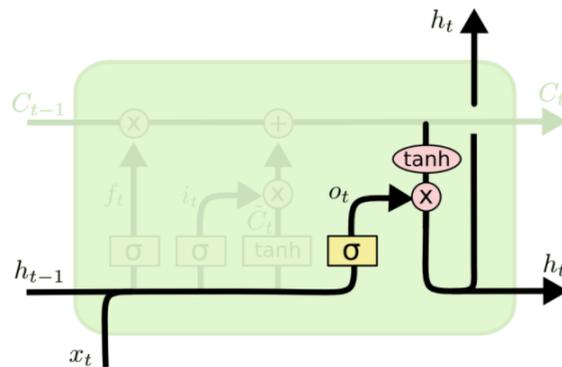


Figura 22 – Saída da rede LSTM. Fonte: Christopher Olah.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3.5)$$

$$h_t = o_t * \tanh(C_t) \quad (3.6)$$

3.7 Seq2Seq: Machine Translation com RNNs

Modelos *Sequence to Sequence (Seq2Seq)* são muito importantes para o campo de *Machine Translation*. A ideia é que a partir de uma sequência de palavras, pode-se criar uma outra sequência de saída referente a primeira (SUTSKEVER; VINYALS; LE, 2014).

Isso pode ser usado para diferentes utilidades. A sequência de saída pode ser desde uma tradução até uma resposta para uma pergunta. Esses modelos são muito utilizados para a criação de *chatbots*.

Modelos *Seq2Seq* são formados por duas RNNs - uma RNN *encoder* e outra RNN *decoder*. A ideia principal se refere a leitura de uma sequência de valores de tamanho variável na RNN *encoder*, gerando um vetor de tamanho fixo como resultado, normalmente chamado de contexto. A RNN *decoder* utiliza o vetor de tamanho fixado gerado para criar uma nova sequência de tamanho variável como saída. As duas RNNs são treinadas juntas para garantir um sincronismo na criação de contextos para determinadas entradas. Isso garante que as saídas estão diretamente relacionadas às variáveis de entradas. É possível visualizar essa estrutura na ilustração a seguir:

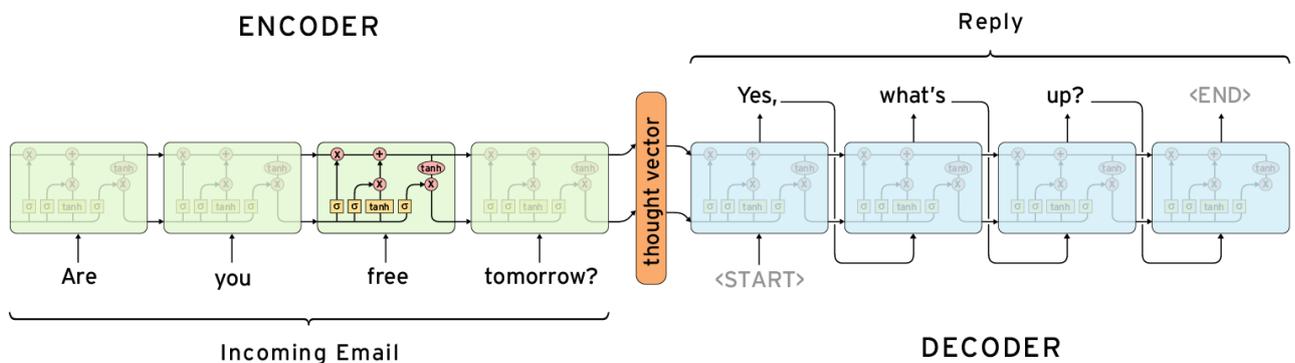


Figura 23 – Representação do modelo Seq2Seq. Fonte: Christopher Olah.

O objetivo do treinamento de uma *Seq2Seq* é maximizar a probabilidade da sequência de saída condicionada pela sequência de entrada ser coerente.

4 Desenvolvimento: IMDB Data Set

4.1 Contextualização

O IMDB é o maior banco de dados/rede social com informações e discussões sobre filmes e séries do mundo. Em 2017, o IMDB registrou mais de 4,4 milhões de títulos, 8 milhões de artistas e mais de 75 milhões de usuários. O portal permite que usuários editem e adicionem novos materiais ao banco de dados. Uma das partes mais importantes do IMDB é a seção de comentários, onde cada usuário pode colocar a sua avaliação sobre determinada série ou filme. Essa seção é apresentada da seguinte maneira:

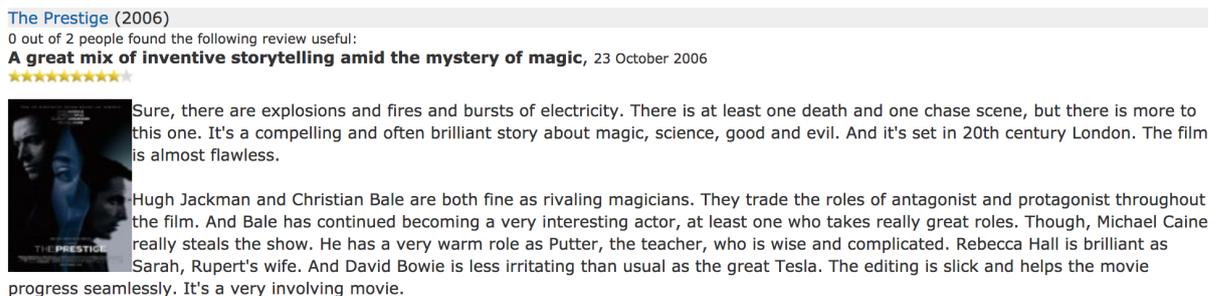


Figura 24 – Típico *Review* do portal IMDB, Screenshot de tela.

Apesar de a quantidade de estrelas selecionada para cada filme ser um método quantitativo importante para avaliação do conteúdo, não é possível visualizar nenhum método explícito que avalie o texto que foi escrito por cada usuário do IMDB. Por esse motivo, foi disponibilizado no site Kaggle um tutorial-desafio para que engenheiros, cientistas de dados, cientistas de computação e outros interessados pudessem utilizar diferentes técnicas de aprendizado para avaliar, textualmente, o significado de uma série de avaliações do site: <https://www.kaggle.com/c/word2vec-nlp-tutorial>.

4.1.1 Descrição do banco de dados

Cada entrada do banco de dados disponível possui o seguinte formato:

id	Sentiment	review
identidade única	positivo(1) ou negativo(0)	avaliação do IMDB

É possível exemplificar com a seguinte entrada:

id	sentiment	review
9352_10	1	Most people, especially young people, may not understand this film. It looks like a story of loss, when it is actually a story about being alone. Some people may never feel loneliness at this level. Cheadles character Johnson reflected the total opposite of Sandler's character Fineman. Where Johnson felt trapped by his blessings, Fineman was trying to forget his life in the same perspective. Jada is a wonderful additive to the cast and Sandler pulls tears. Cheadle had the comic role and was a great supporter for Sandler. I see Oscars somewhere here. A very fine film. If you have ever lost and felt alone, this film will assure you that you're not alone. Jerry

Figura 25 – Exemplo do banco de dados IMDB

4.1.2 Análise Sentimental

O objetivo do desafio recai, então, em uma das áreas mais proeminentes nas técnicas de aprendizado e modelagem matemática: *Sentiment Analysis*. Como o próprio nome supõe, o objetivo principal desse estilo de aprendizado é entender qual o tipo de expressão emocional que determinado texto representa: angústia, tristeza, felicidade, raiva. No caso do banco de dados do IMDB, a avaliação é bem simples e consiste apenas em determinar se determinado texto contém uma avaliação positiva ou negativa sobre o filme.

Por mais simples que essa missão seja para um ser humano, não é, necessariamente intuitivo para a máquina. Muitas marcações normalmente utilizadas para expressar algo negativo podem ser contextualizadas em uma determinada sentença de forma que represente algo positivo, o exato oposto. Por exemplo: 'Não acredito que o filme acabou tão rápido. Saí da sala de cinema com a terrível sensação de querer ainda mais. Espero ver de novo em breve!'. A frase anterior, por mais que contenha várias marcações negativas como 'Não' e 'terrível', representa uma avaliação positiva do filme. Essa nuance precisa ser captada pelo modelo matemático de avaliação.

4.2 Benchmarking

Como se trata de um desafio-tutorial, a equipe do Kaggle, em conjunto com o IMDB, proporcionou um código de *benchmark* para a comparação com outros modelos. Dessa forma, será possível comparar os resultados deste trabalho com o *benchmark* utilizado no desafio e verificar os possíveis *trade-offs* entre cada modelo e as razões teóricas que explicam as diferenças.

4.3 Metodologia

- A primeira etapa do processo foi utilizar o banco de dados de avaliações do IMDB para criar um *corpus*. A partir disso, é possível gerar um *word embedding*, tendo assim um vetor representativo único para cada palavra do vocabulário do *corpus*.

- Posteriormente, foi escolhido o algoritmo de Word2Vec para gerar os vetores únicos de cada palavra do vocabulário.
- Dois diferentes corpus foram utilizados para o *Word2Vec*. O primeiro foi construído a partir das próprias palavras de avaliações do IMDB e o outro foi o modelo pré-gerado do *Google News*, com mais de 3 bilhões de palavras de treinamento.
- Cada avaliação foi transformada em um vetor de entrada, criado a partir das palavras pertencentes em cada texto. A partir do centro geométrico de cada palavra-vetor da avaliação, um vetor médio, chamado de centróide, era criado. Dessa forma, cada avaliação consiste de um vetor de entrada para o modelo, associado a um vetor binário de saída entre 0 ou 1.
- Posteriormente, foi utilizada uma rede neural LSTM para treinar o banco de dados a partir dos resultados já conhecidos. Utilizou-se um modelo de validação cruzada para dividir o banco de dados em treinamento e teste, com o objetivo de evitar *overfitting*.
- Também foram utilizadas outras técnicas, como *Random Forests* e *Softmax* com objetivo de gerar comparações de resultados.
- Os parâmetros foram reajustados de forma a otimizar a porcentagem de resultados positivos.

4.4 Especificação técnica

A seguir, são explicitadas as configurações técnicas do projeto depois de todos os testes de modelo, com a versão final utilizada e apresentada como resultado na seção seguinte.

Primeiramente, a decisão de linguagem e *framework*:

Linguagem de programação	Python
Versão	2.7
Framework ANN	TensorFlow
Versão	0.12.1
Framework Word2Vec	Gensim
Versão	1.0.0

Em seguida, a configuração do servidor Amazon utilizado:

Configuração	Amazon Cloud m4.10xlarge
Memória	40 GiB
SSD	160 GB
Banda	4000 Mbps
Otimização	Cython
Tempo de execução	20 horas

Por fim, a configuração da rede neural LSTM utilizada:

Número de Épocas	100
Número de Classes	2
Tamanho do batch	60
Tamanho do Chunk	25
Quantidade de chunks	12
Tamanho da rede	500 unidades

4.5 Resultados

4.5.1 *Word Embeddings*

A partir do algoritmo *Word2Vec*, gerou-se um vetor único para cada palavra do vocabulário. Utilizou-se o *Word2Vec* do *Google News* e outro feito a partir dos próprios dados do IMDB. Para demonstrar que o mapeamento possui significado semântico, é possível adquirir o seguinte gráfico, utilizando os dados do *Google News*:

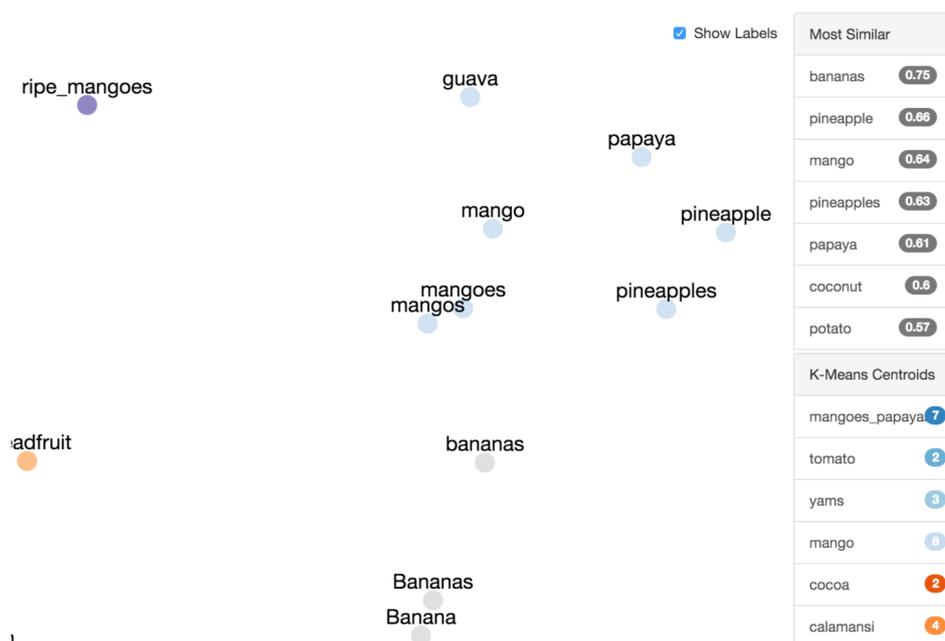


Figura 26 – Mapeamento de significado em *Word2Vec*, Screenshot de tela.

É interessante também utilizar os dados do IMDB para testar o mapeamento semântico, fazendo uso de funções próprias do modelo *Word2Vec*:

```
model.doesnt_match("france germany england london ".split())
london
```

Figura 27 – Teste do modelo *Word2Vec*.

```
model.most_similar("depp")
[(u'johnny', 0.6756786704063416), (u'dylan', 0.6571077108383179), (u'brandon', 0.6503952741622925), (u'
227621078491), (u'lawrence', 0.6362581253051758), (u'gallagher', 0.6349369287490845), (u'macdonald', 0
```

Figura 28 – Teste do modelo *Word2Vec*.

4.5.2 Resultados finais e comparações

É importante comparar técnicas distintas, utilizando *word embeddings* distintos, para verificar qual método possui o melhor resultado. Esse experimento foi realizado e é ilustrado no gráfico seguinte:

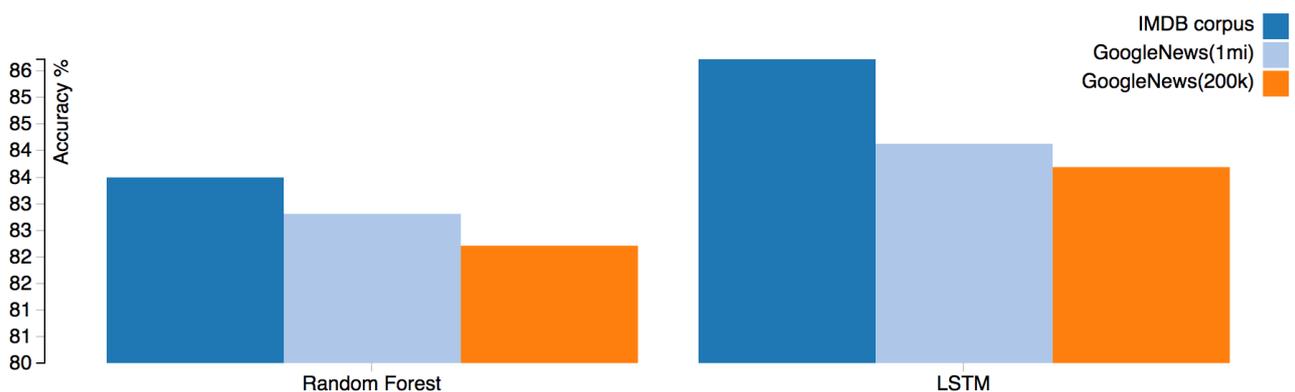


Figura 29 – Comparação entre métodos, IMDB

Assim, é possível escolher o melhor modelo e compará-lo com o *benchmark* do IMDB e com um método simples (computacionalmente menos complexo) como o *softmax*, para entender se é, de fato, necessário utilizar as técnicas apresentadas neste trabalho.



Figura 30 – Comparação entre *benchmark* e modelo, IMDB

É possível perceber que existe um aumento significativo em acurácia com o auxílio das técnicas de *Deep Learning* e NLP dentro do modelo. É possível perceber que ao utilizar o vocabulário formado pelo Word2Vec do *corpus* das próprias palavras do IMDB, o resultado é superior se comparado com a versão do Google News (85% vs 83%), mesmo com o vocabulário e complexidade sendo maior no segundo. Uma possível explicação é que os mapeamentos semânticos extras que o *corpus* do Google News apresenta pouco adiciona para o entendimento do significado das avaliações do IMDB, causando apenas um nível adicional de ruído. De qualquer forma, é possível notar que o aumento do número de palavras vetorizadas utilizadas gera um acréscimo, mesmo que pequeno, na taxa de acerto de classificação.

4.5.3 Aplicação desenvolvida

Além do modelo matemático, foi desenvolvida, durante o projeto, uma aplicação capaz de utilizar textos arbitrários para avaliação e verificar se o conteúdo é positivo ou negativo. Um *screenshot* da aplicação sendo utilizada pode ser encontrado a seguir:

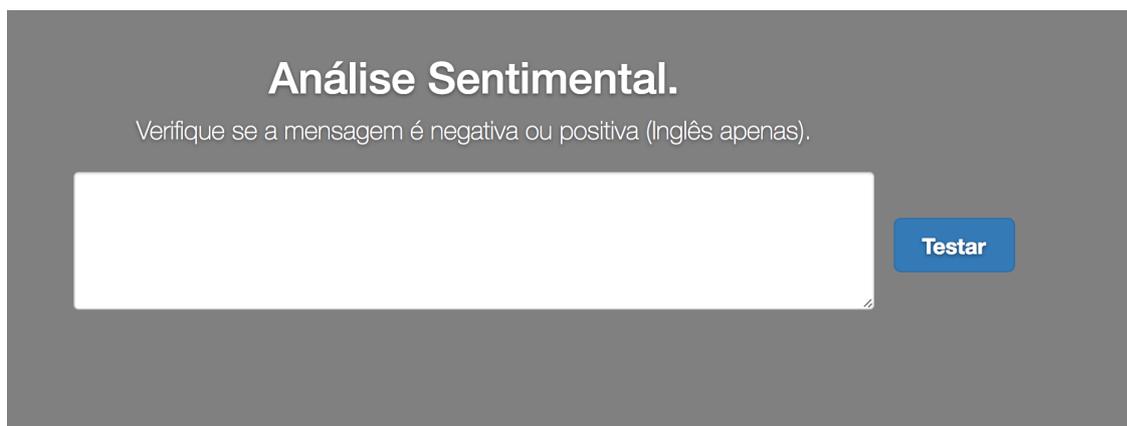


Figura 31 – Aplicativo desenvolvido, IMDB

4.5.4 Exemplificação de resultados corretos

Podemos demonstrar exemplos do funcionamento correto do modelo.

Primeiramente, dois exemplos utilizando um valor contido dentro do banco de dados e não utilizado para o treinamento do modelo, apenas para testes:

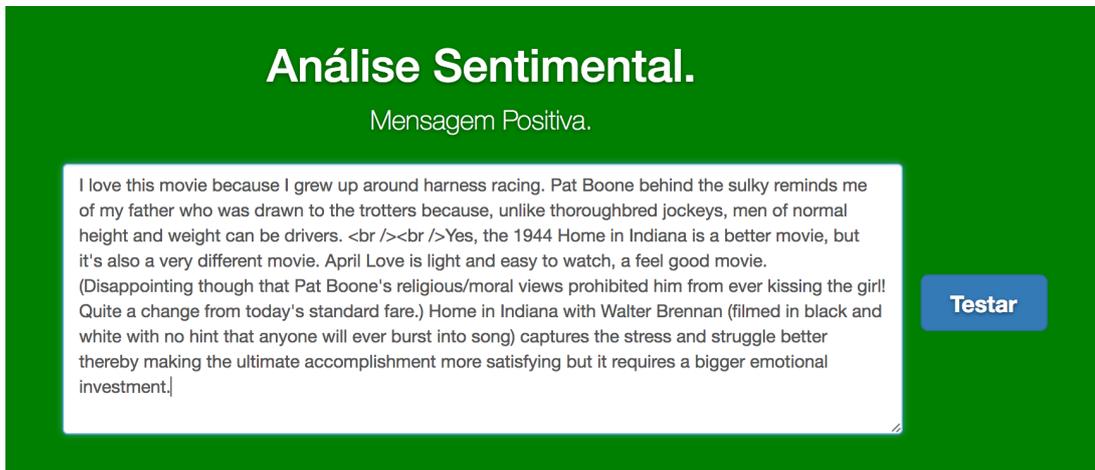


Figura 32 – Resultado correto, IMDB



Figura 33 – Resultado correto, IMDB

Posteriormente, um exemplo com um valor arbitrário utilizado pela aplicação:

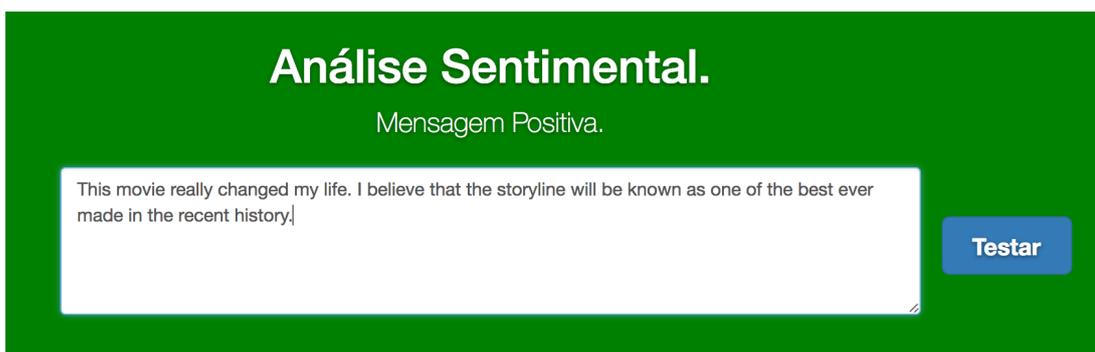
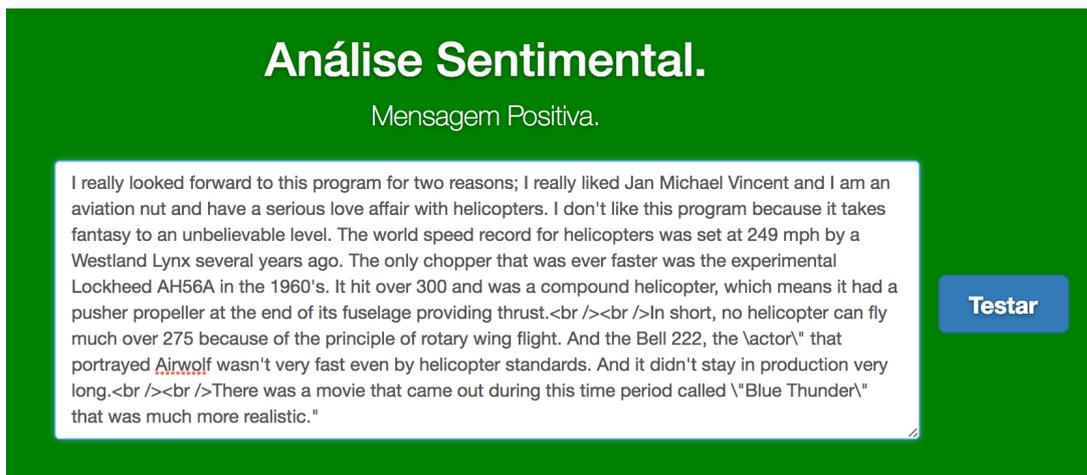


Figura 34 – Resultado correto gerado

4.5.5 Exemplificação de resultados incorretos

Finalmente, é importante explicitar casos em que a aplicação não funciona corretamente.

Neste exemplo, a classificação correta deveria ter sido negativa. Esse texto é um valor contido dentro do banco de dados e não utilizado para o treinamento do modelo, apenas para testes (validação cruzada):



The image shows a web interface for sentiment analysis. At the top, it says "Análise Sentimental." followed by "Mensagem Positiva." Below this is a text input area containing a paragraph of text. To the right of the text area is a blue button labeled "Testar".

Análise Sentimental.
Mensagem Positiva.

I really looked forward to this program for two reasons; I really liked Jan Michael Vincent and I am an aviation nut and have a serious love affair with helicopters. I don't like this program because it takes fantasy to an unbelievable level. The world speed record for helicopters was set at 249 mph by a Westland Lynx several years ago. The only chopper that was ever faster was the experimental Lockheed AH56A in the 1960's. It hit over 300 and was a compound helicopter, which means it had a pusher propeller at the end of its fuselage providing thrust. In short, no helicopter can fly much over 275 because of the principle of rotary wing flight. And the Bell 222, the "factor" that portrayed *Airwolf* wasn't very fast even by helicopter standards. And it didn't stay in production very long. There was a movie that came out during this time period called "Blue Thunder" that was much more realistic."

Testar

Figura 35 – Resultado incorreto, IMDB

5 Desenvolvimento: Quora Data Set

5.1 Contextualização

Quora é uma rede social cujo produto é a agregação de conhecimento, com o funcionamento baseado em perguntas e respostas. O Quora é semelhante ao Yahoo! Respostas, no entanto mantém e agrupa em seu banco de dados, informações mais coerentes, além de organizar e direcionar as páginas as respostas encontradas, como se fosse uma enciclopédia virtual. O que diferencia esta das demais ferramentas de busca é que pode ser editável e com a possibilidade de seguir todas as modificações realizadas pelos demais usuários. Um típico *post* do Quora se encontra no seguinte formato:

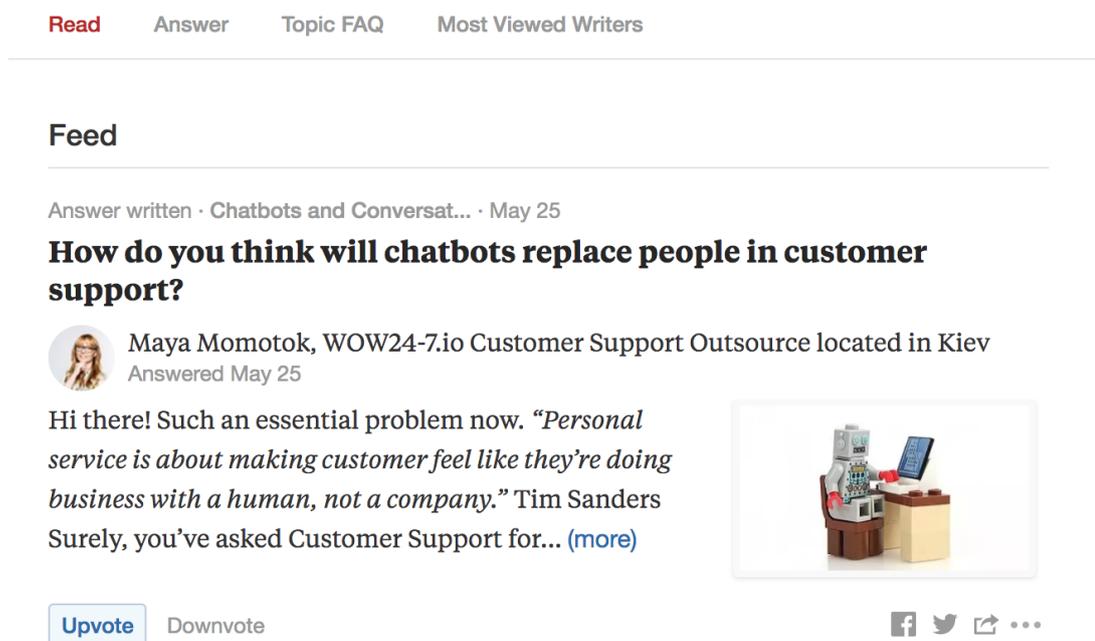


Figura 36 – Exemplo de *post* do Quora, Screenshot de tela.

Um dos grandes problemas enfrentados pelo Quora é a quantidade enorme de perguntas repetidas na plataforma. Isso não só desestimula os usuários, mas também cria diversos problemas em gastos excessivo com armazenamento de informação. Dessa forma, foi criado um desafio no Kaggle com o objetivo de diminuir esse problema. Este trabalho procura uma estratégia de resolução.

5.1.1 Descrição do banco de dados

Cada entrada do banco de dados disponível possui o seguinte formato:

id	qid1 qid2	question1 question2	is_duplicate
identidade única do par	identidade única de cada pergunta	conteúdo de cada pergunta	avaliação do IMDB

Exemplifica-se com a seguinte entrada:

id	qid1	qid2	question1	question2	is_duplicate
0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0

Figura 37 – Exemplo de entrada no banco de dados

5.1.2 Normalização

O objetivo do desafio recai, então, em uma das áreas mais difíceis e ainda abertas nas técnicas de aprendizado e modelagem matemática: Normalização. Verificar se duas sentenças, frases, parágrafos ou conjuntos de palavras são iguais ou expressam as mesmas coisas é uma tarefa difícil, já que pequenas variações de vocabulário podem possuir significados diametralmente opostos.

Essa função não é necessariamente sempre intuitiva para um ser humano. Por mais que seja possível perceber similaridades, não é uma percepção óbvia e necessita análise. Isso significa algo ainda mais complexo para uma máquina: entender nuances complexas.

5.2 Comparação vetorial

Diferente do desafio IMDB, a entrada não é de apenas um conjunto textual e, sim, a comparação entre duas diferentes perguntas. Para isso, é necessário processar o banco de dados de forma a transformar as duas entradas em apenas uma.

A ideia principal é comparar o modelo de Word2Vec + LSTM com classificadores simples vetoriais, como a similaridade de cosseno e a distância euclidiana.

Será referenciado o modelo Word2Vec + LSTM como *Difference Vector*. E este será diretamente comparado com os outros dois modelos mais simples. Dessa forma, será possível entender o quão valioso um método mais robusto computacionalmente será, comparativamente. Três métodos diferentes de comparação vetorial foram utilizados:

- Vetor diferença: a diferença entre os dois vetores, uma entrada multidimensional da diferença entra a representação vetorial de uma sentença e outra (modelo Word2Vec + LSTM).

- Similaridade de cosseno: uma entrada unidimensional que mede o cosseno do ângulo entre vetores, considerando o seu produto interno. É dado pela seguinte fórmula:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Figura 38 – Similaridade de cosseno

- Distância euclidiana: uma entrada unidimensional da distância entre dois pontos. A distância euclidiana (ou distância métrica) entre os pontos em um espaço euclidiano n-dimensional, é definida como:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}.$$

Figura 39 – Distância Euclidiana

5.3 Metodologia e Entropia Cruzada

A metodologia, depois do processamento do banco de dados para transformar uma dupla de vetores em um vetor de similaridade, é idêntica à metodologia utilizada no banco de dados do IMDB. Entretanto, será medida a qualidade do modelo também pela entropia cruzada. Para uma predição p_i para uma saída y_i com N amostras, a entropia cruzada é dada pela seguinte função:

$$-\frac{1}{N} \sum_{i=1}^N [y_i \log p_i + (1 - y_i) \log (1 - p_i)].$$

Figura 40 – Entropia Cruzada

A principal diferença dessa medição, em comparação com a acurácia, é que a entropia cruzada é um critério mais robusto, pois considera a probabilidade atribuída a cada saída. Se um modelo, por exemplo, credita que certa entrada X possui probabilidade 0.51 de ser da classe Y e outro modelo atribui 0.99 para essa probabilidade, os dois seriam contabilizados da mesma maneira em uma medida simples de acurácia.

5.4 Especificação técnica

A seguir, são explicitadas as configurações técnicas do projeto depois de todos os testes de modelo, com a versão final utilizada e apresentada como resultado na seção seguinte.

Primeiramente, a decisão de linguagem e *framework*:

Linguagem de programação	Python
Versão	2.7
Framework ANN	TensorFlow
Versão	0.12.1
Framework Word2Vec	Gensim
Versão	1.0.0

Em seguida, a configuração do servidor Amazon utilizado:

Configuração	Amazon Cloud m4.10xlarge
Memória	40 GiB
SSD	160 GB
Banda	4000 Mbps
Otimização	Cython
Tempo de execução	16 horas

Por fim, a configuração da rede neural LSTM utilizada:

Número de Épocas	70
Número de Classes	2
Tamanho do batch	60
Tamanho do Chunk	25
Quantidade de chunks	12
Tamanho da rede	440 unidades

5.5 Resultados

5.5.1 Resultados finais e comparações

Aqui, é feita a comparação entre os três métodos escolhidos, em termos da entropia cruzada:



Figura 41 – Comparação entre métodos por entropia cruzada, Quora

Também podemos fazer a comparação em termos da acurácia.



Figura 42 – Comparação entre métodos por acurácia, Quora

É possível perceber que existe um aumento significativo em acurácia com o auxílio das técnicas de *Deep Learning* e NLP dentro do modelo. Percebe-se que o modelo mais robusto, a partir de word embeddings e redes neurais recorrentes, apresenta um resultado muito superior, tanto em termos da entropia cruzada (0.59 vs 2.2) quanto de acurácia (69% vs 58%). Isso comprova a efetividade do mapeamento semântico e das redes neurais com capacidade de memória.

5.5.2 Aplicação desenvolvida

Além do modelo matemático, foi desenvolvida, durante o projeto, uma aplicação capaz de utilizar textos arbitrários para avaliação e verificar se o conteúdo é positivo ou negativo. Um *screenshot* da aplicação sendo utilizada pode ser encontrado a seguir:

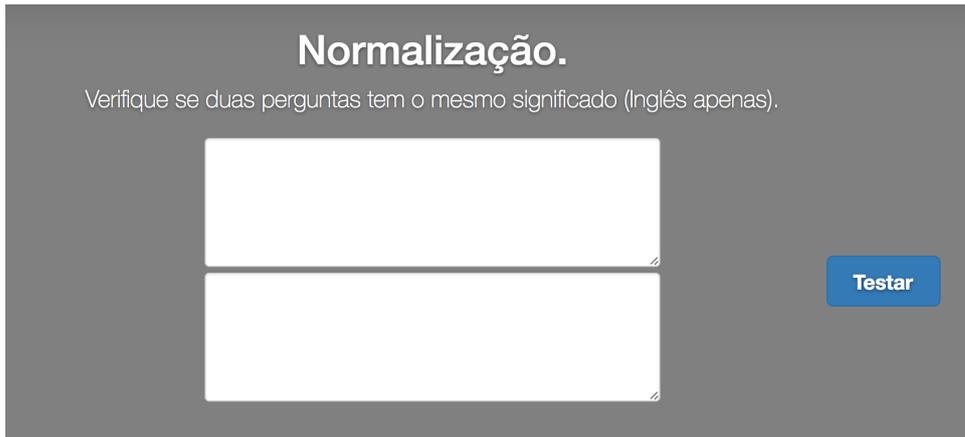


Figura 43 – Aplicação desenvolvida, Quora

5.5.3 Exemplificação de resultados corretos

É possível demonstrar exemplos do funcionamento correto do modelo.

Primeiramente, dois exemplos utilizando um valor contido dentro do banco de dados e não utilizado para o treinamento do modelo, apenas para testes:

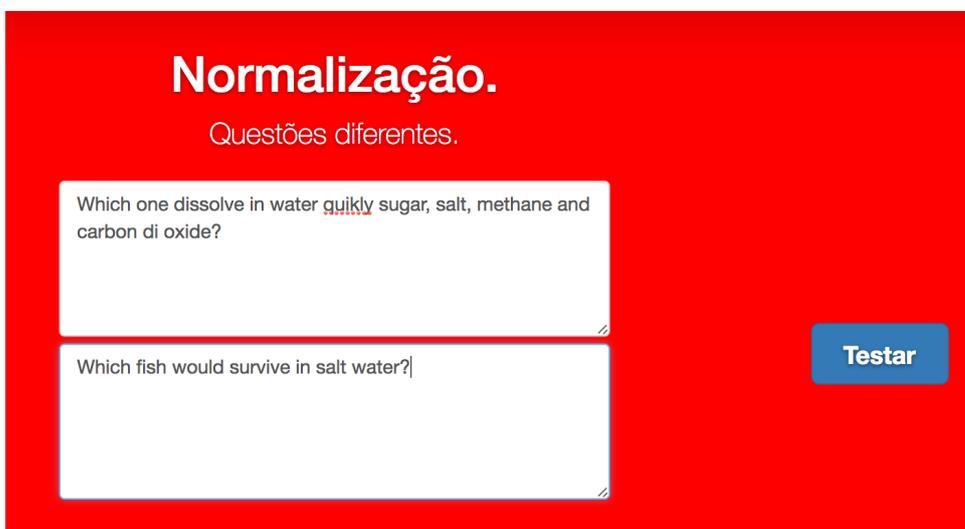


Figura 44 – Resultado correto, Quora

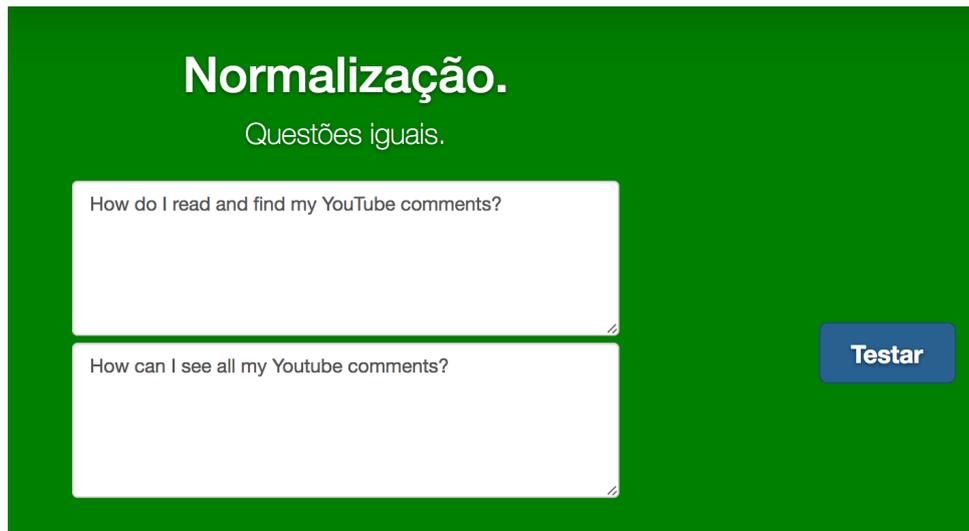


Figura 45 – Resultado correto, Quora

5.5.4 Exemplificação de resultados incorretos

Finalmente, é importante explicitar casos em que a aplicação não funciona corretamente.

Neste exemplo, a classificação correta deveria ter sido de que as perguntas são idênticas. Novamente, um caso utilizando um valor contido dentro do banco de dados e não utilizado para o treinamento do modelo, apenas para testes:

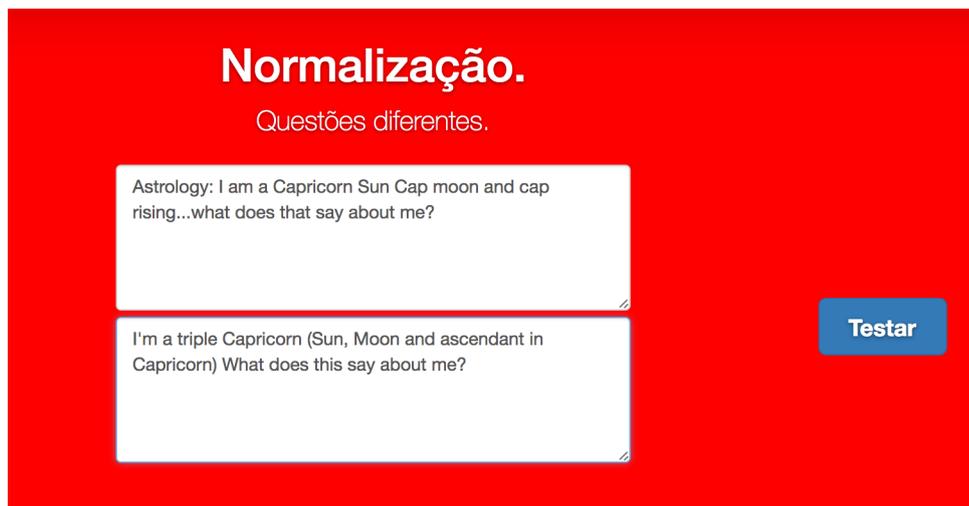


Figura 46 – Resultado incorreto, Quora

6 Desenvolvimento: Twitter Chatbot

6.1 Contextualização

6.1.1 Twitter

O Twitter é uma das maiores redes sociais do mundo, com mais de mais de 300 milhões de usuários ativos. A ferramenta foi criada em Março de 2006 por Jack Dorsey, Noah Glass, Biz Stone e Evan Williams a lançado em Julho do mesmo ano, ganhando popularidade de forma extremamente rápida. O serviço coordena mais de 300 milhões de mensagens, chamadas de *tweets*, por dia. Os *tweets* são compartilhados pelos usuários em seus perfis pessoais e podem ter, no máximo, 140 caracteres. A ideia principal dos fundadores do Twitter é que os usuários possam compartilhar o que estão pensando no momento, de forma interativa, rápida e curta.

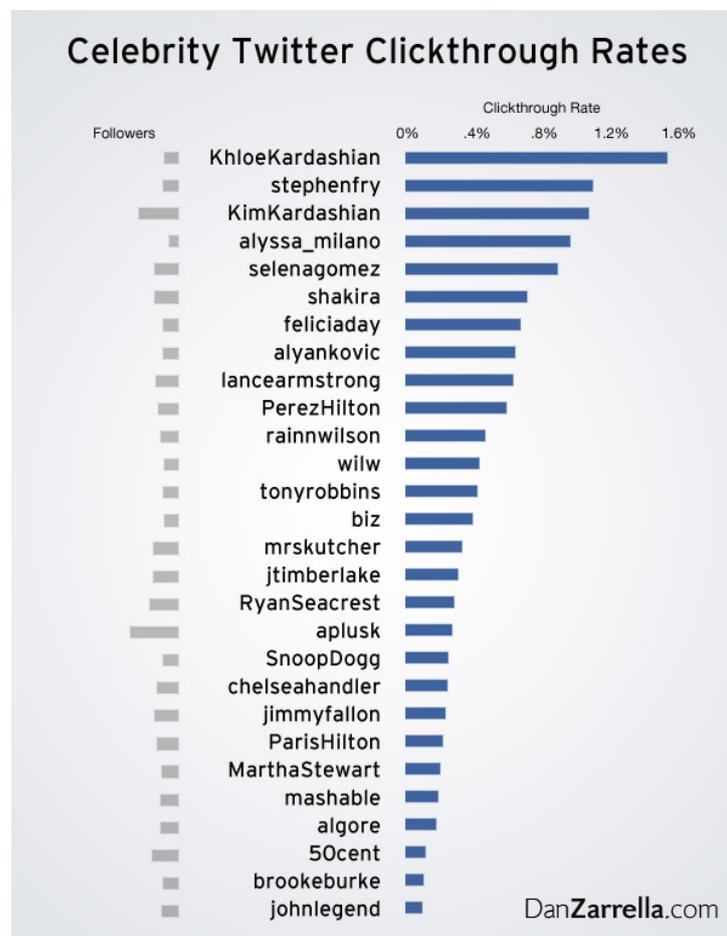


Figura 47 – Clickthrough Rate por Celebridade. Fonte: Twitter Inc.

Com o crescimento da importância das redes sociais para o a imagem pública, celebridades começaram a usar ferramentas como o Twitter para garantir o grande alcance de suas marcas pessoais. O objetivo é garantir que os usuários cliquem nos *links* compartilhados ou repassem informações sobre marcas ou propagandas. O gráfico acima mostra o poder de persuasão de algumas celebridades conhecidas do Twitter, avaliado pela porcentagem das pessoas que clicam nos *tweets* compartilhados.

6.1.2 Chatbots

Um chatbot é um serviço desenhado em torno de regras e/ou inteligência artificial, em que um usuário interage com um computador por meio de uma interface de chat. Esse serviço possui grandes vantagens, entre elas garantir uma interação direta e personalizada para um usuários e diminuir a quantidade de gastos com centros ineficientes de *call-centers* ou *help desks*. Várias empresas e redes sociais, como Facebook, Slack, Instagram, investiram em projetos como esses nos últimos anos.

Se um usuário, por exemplo, deseja comprar um sapato da Nordstrom, poderia, diretamente, interagir com um *bot* de auxílio, ao invés de navegar dentro das infinitas opções de uma página *web*.

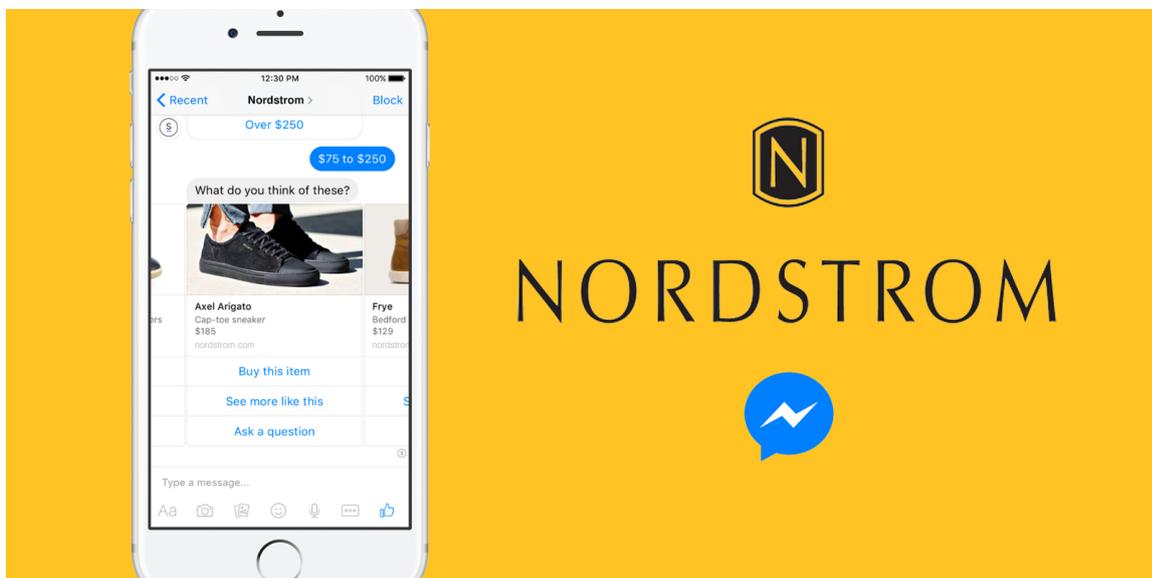


Figura 48 – Exemplo de Chatbot. Fonte: Nordstrom.

6.1.3 Modelos de Chatbots

O objetivo dessa seção do trabalho é conseguir utilizar dados do Twitter para gerar um *chatbot* eficiente. Existem dois estilos principais de *chatbots*: modelos baseados em seleção e os modelos baseados em geração (ZHAO; CALLAN, 2008). Cada um possui diferenças em termos de complexidade e eficiência.

Modelos baseados em seleção

Modelos baseados em seleção são fáceis de entender e adaptar. Basicamente, funcionam com uma série de respostas pré-definidas e aprendem a selecionar a melhor resposta possível, dependendo da pergunta solicitada. Como não é necessário gerar nenhum texto novo, apenas selecionar o adequado, alguns problemas como erros gramaticais são evitados. Além disso, possuem alta capacidade de atender os requisitos empresariais em termos de efetividade. Entretanto, a capacidade de resposta para mecanismos de seleção depende da quantidade de respostas disponíveis e podem, facilmente, se tornarem impraticáveis em termos de escalabilidade, dependendo da aplicação.

Modelos baseados em geração

Modelos geradores são difíceis de se produzir e não dependem de nenhum conjunto externo de respostas pré-estabelecidas. Esses modelos dependem inteiramente de métodos de Inteligência Artificial, Tradução de Máquina e Processamento de Linguagem Natural. Modelos geradores criam respostas novas para uma determinada pergunta, a partir do aprendizado de pares de respostas e perguntas anteriores. Não existe um consenso acadêmico na melhor forma de construir *chatbots* geradores, mas é uma das áreas de pesquisas dentro de inteligência artificial mais estudada pelo seu alto potencial em resolver o problema de escalabilidade de modelos de seleção. Empresas como Google também desenvolveram artigos relacionados ao tema.

6.2 Benchmark e Exemplo do Google

Não existe um *Benchmark* adequado e universalmente conhecido para se verificar a qualidade de respostas de um *bot*. O Teste de Turing, por mais que universalmente conhecido, ainda não foi conceitualmente atingido e, portanto, não seria adequado para comparar a eficiência entre modelos. Portanto, a forma escolhida de verificar a eficiência de modelos foi a partir da comparação de funcionamento entre diferentes perspectivas. De qualquer forma, em 2015, pesquisadores do Google criaram um chatbot baseado em diálogos de filmes (com um data set mais robusto e completo do que o utilizado neste trabalho). Alguns diálogos podem ser vistos aqui:

Human: is moral and ethics the same?

Machine: yes sir.

Human: what are the things that i do to be moral?

Machine: i don't know.

Human: give me an example of moral action

Machine: I'm not a moralist

Para este trabalho, foram utilizados três conjuntos de dados diferentes para realizar a mesma função de *corpus*:

- Cornell Movie Dialogues: Conjunto de pares de diálogos entre personagens de filmes famosos, desenvolvido pela universidade de Cornell.
- Twitter Aleatório 7k: Conjunto de 7k pares de tweets aleatórios de um dia específico.
- Twitter Celebidades 400-3k: Conjunto de 3 mil pares de tweets selecionados dentro de 400 mil *tweets* de celebridades influentes na plataforma.

Dessa forma, é possível realizar duas comparações: comparação entre os modelos utilizando o Twitter com o modelo de diálogos de filmes e a comparação entre modelos diferentes utilizando o Twitter.

6.3 Metodologia

- A primeira etapa do processo foi a seleção dos bancos de dados a serem utilizados. Como reportado anteriormente, três diferentes conjuntos foram utilizados. Os dois primeiros são facilmente encontrados na *internet* e são utilizados de diversas maneiras diferentes para atividades acadêmicas interessantes. O segundo banco de dados foi criado para o desenvolvimento deste trabalho e será disponibilizada posteriormente para que outros usuários façam experimentos com o mesmo. As etapas para geração do *corpus* próprio são descritas a seguir.
- Para a geração do *corpus* Twitter Celebidades 400-3k, primeiramente foram mineirados 400 mil *ids* de *tweets* de celebridades famosas do Twitter. Exemplos de perfis utilizados: 'Kim Kardashian', 'Britney Spears', 'Justin Bieber'. A maioria dos usuários possuem dezenas de milhares de *tweets*, essencial para a construção do modelo. As *ids* foram recolhidas a partir de um web scraping de duração de 24h, como demonstrado a seguir:

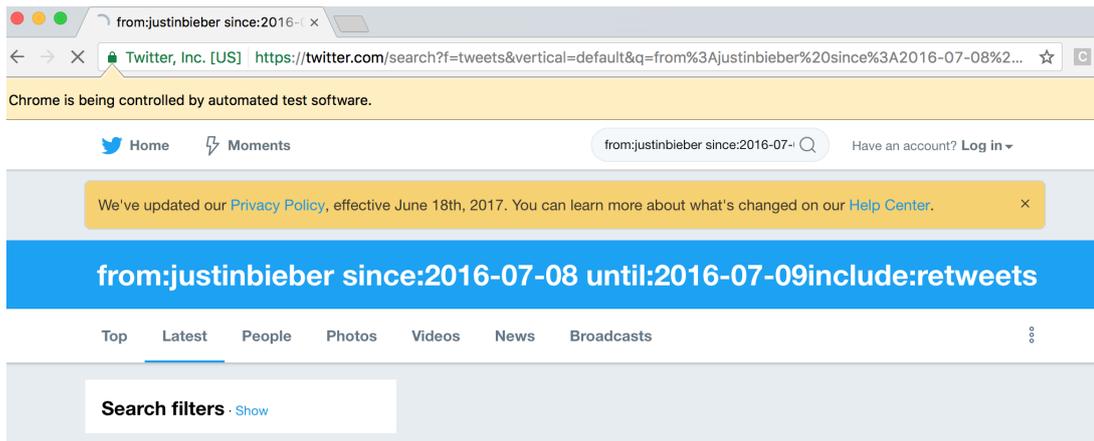


Figura 49 – *Web Scraping* para o Twitter

- Posteriormente, foi utilizada a *framework* Tweepy para recolher os tweets a partir da busca por *ids*. Esse processo durou em torno de 8 horas:

```
Gabriels-MacBook-Pro:twitter_collect gbayomi$ python tweet.py
total ids: 2394
currently getting 0 - 100
currently getting 100 - 200
currently getting 200 - 300
currently getting 300 - 400
■
```

Figura 50 – Tweepy API

- Em seguida, foi necessário selecionar os *tweets* que representam um par de pergunta e resposta e limpar cada um deles:

```
text = text.replace("\", "xxapxx")
new_text = ' '.join(re.sub("@[A-Za-z0-9+]|([\^0-9A-Za-z \t])|(\w+:\w+\wS+)", " ", text).split())
new_text = new_text.replace("xxapxx", "\")

words = new_text.split(' ')
count_words = len(words)
```

Figura 51 – Limpeza de informação do Tweet

Com isso, um *tweet* com o seguinte formato



Figura 52 – Tweet Original

é transformado em uma nova string de texto da seguinte forma:

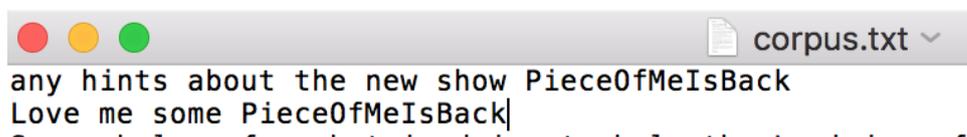


Figura 53 – Tweet após a limpeza

A última etapa do processo de limpeza é a união de todos os arquivos em um arquivo único de texto (.txt).

- Em seguida, foi feita a conversão de texto em vetores, utilizando dessa vez o método de *one-hot encoder*, que é menos efetivo, mas menos complexo computacionalmente. Isso foi feito devido a limitações de tempo e memória para se criar um modelo *Word2Vec*.
- Com os 3 modelos de banco de dados no mesmo formato, é possível criar um modelo Seq2Seq com LSTMS, anteriormente descritos neste trabalho. A ideia é utilizar os pares pergunta-resposta como treinamento da rede. Assim, para uma nova entrada (pergunta), a rede Seq2Seq resultará em uma saída (resposta) adequada.
- Utilizou-se um *mini-batch* de 32 pares e 40 mil épocas para o treinamento do modelo.

6.4 Especificação técnica

A seguir, são explicitadas as configurações técnicas do projeto depois de todos os testes de modelo, com a versão final utilizada e apresentada como resultado na seção seguinte.

Primeiramente, a decisão de linguagem e *framework*:

Linguagem de programação	Python
Versão	3.0
Framework ANN	TensorFlow
Versão	1.0.0
Framework Word2Vec	Gensim
Versão	2.2.0

Em seguida, a configuração do servidor Amazon utilizado:

Configuração	Amazon Cloud m4.10xlarge
Memória	40 GiB
SSD	160 GB
Banda	4000 Mbps
Otimização	Cython
Tempo de execução	36 horas

Por fim, a configuração da rede neural LSTM Seq2Seq utilizada:

Número de Épocas	200
Tamanho da rede	250 unidades

6.5 Resultados

Foi criado um aplicativo, como demonstrado a seguir, que auxilia nos testes do *chatbot*:

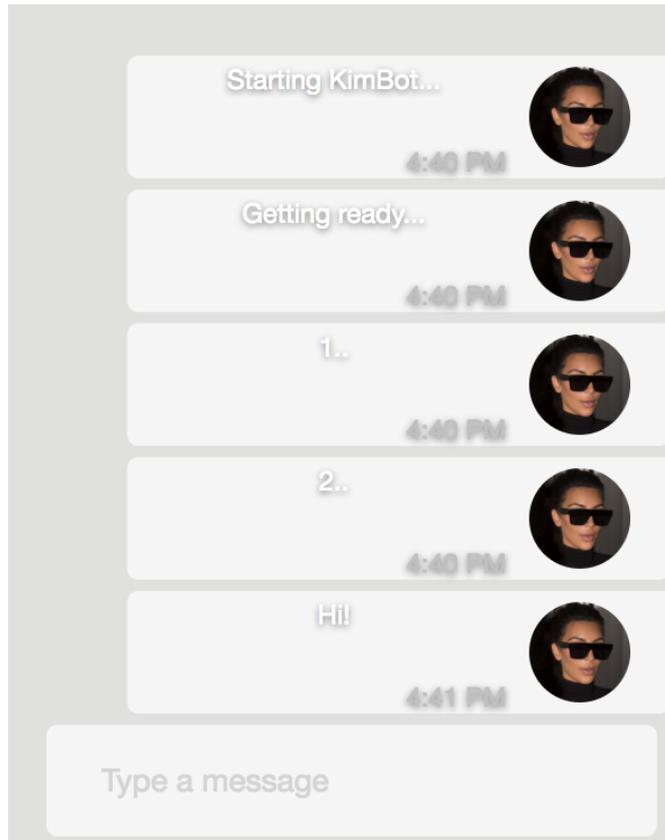


Figura 54 – Aplicação em processo de loading.

6.5.1 Cornell Movie Dialogs

Os seguintes resultados foram proporcionados pelo conjunto de dados do *Cornell Movie Dialogs*:

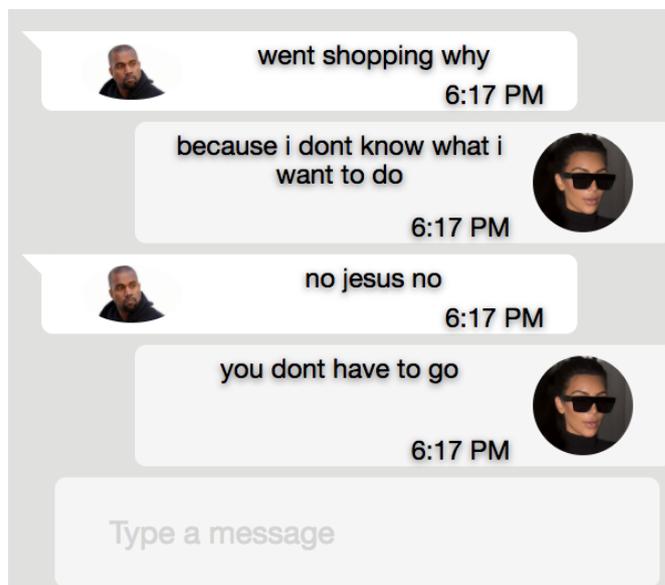


Figura 55 – Cornell Movie Dialogs - Exemplo 1

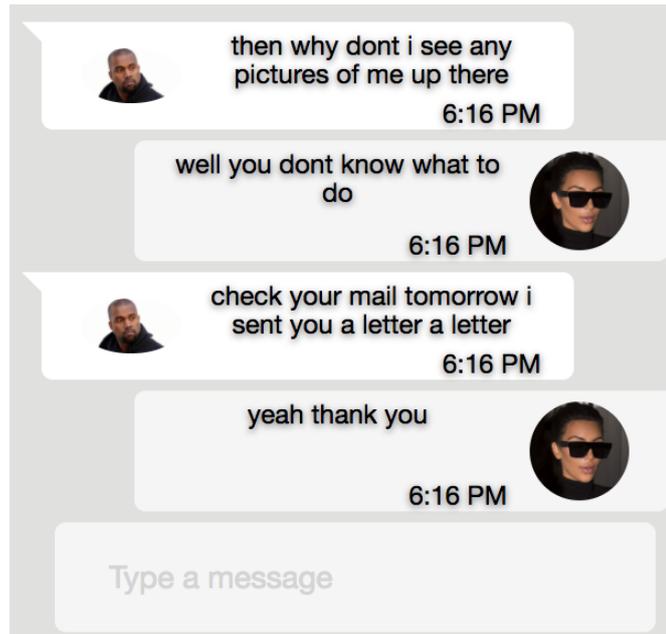


Figura 56 – Cornell Movie Dialogs - Exemplo 2

6.5.2 Twitter Aleatório 7k

Os seguintes resultados foram proporcionados pelo conjunto de dados do Twitter Aleatório 7k:

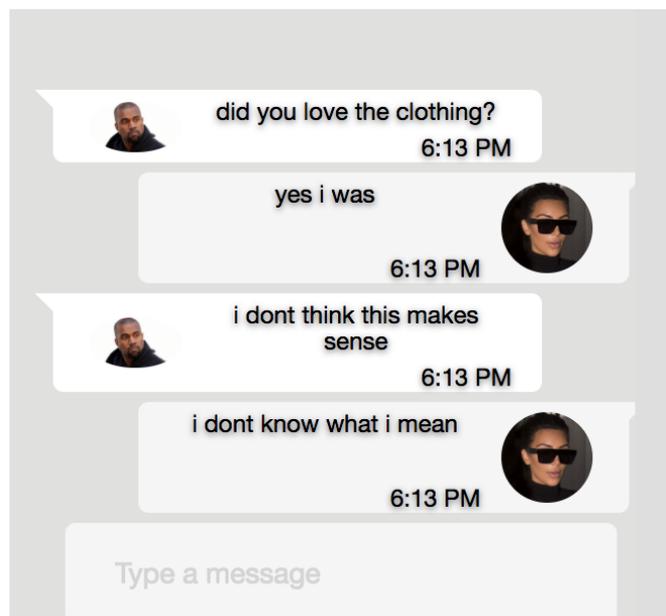


Figura 57 – Twitter Aleatório 7k - Exemplo 1

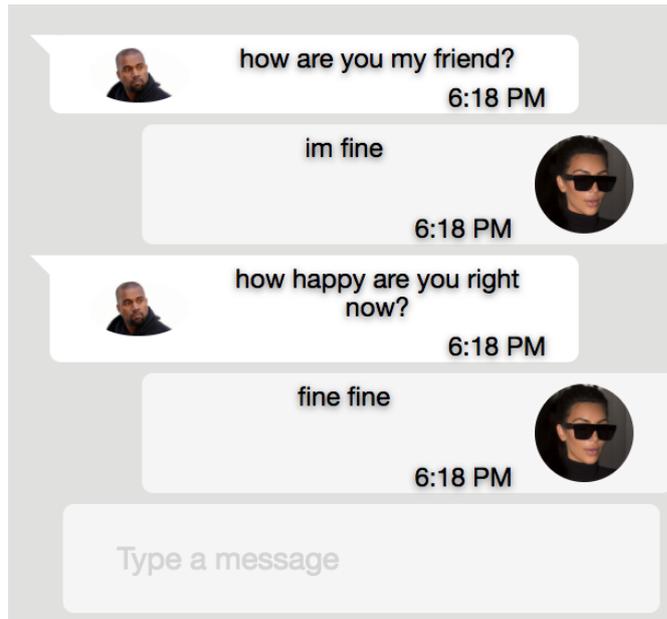


Figura 58 – Twitter Aleatório 7k - Exemplo 2

6.5.3 Twitter Celebridades 400-3k

Os seguintes resultados foram proporcionados pelo conjunto de dados do Twitter Aleatório Twitter Celebridades 400-3k:

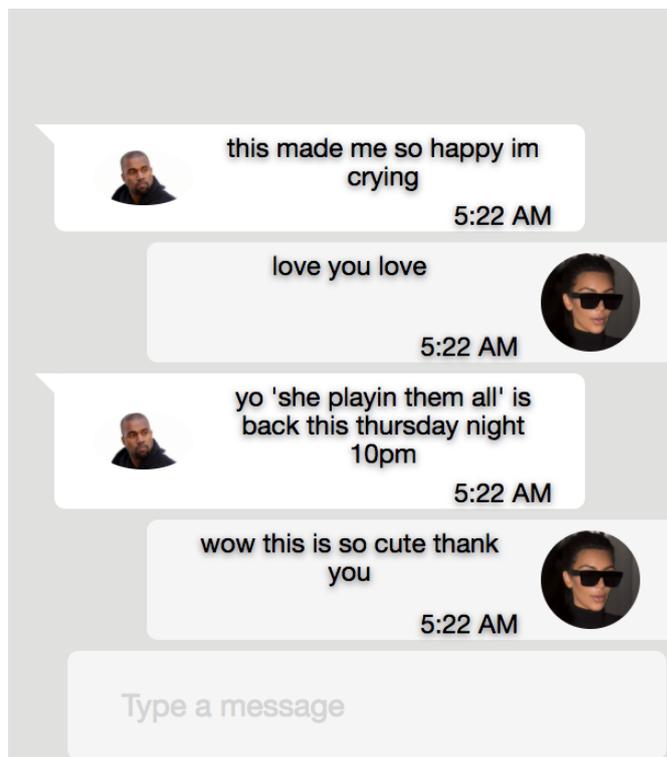


Figura 59 – Twitter Celebridades 400-3k - Exemplo 1



Figura 60 – Twitter Celebridades 400-3k - Exemplo 2

6.5.4 Comparação de métodos

Por fim, é possível mandar mensagens similares para o *bot* e verificar se o comportamento é similar em cada um dos conjuntos de dados.

Primeiramente, é interessante comparar o resultado obtido pelo Google em 2015 com uma base de dados de filmes com o modelo obtido neste trabalho, utilizando um modelo matemático e base de dados diferentes (mas também baseada em filmes):

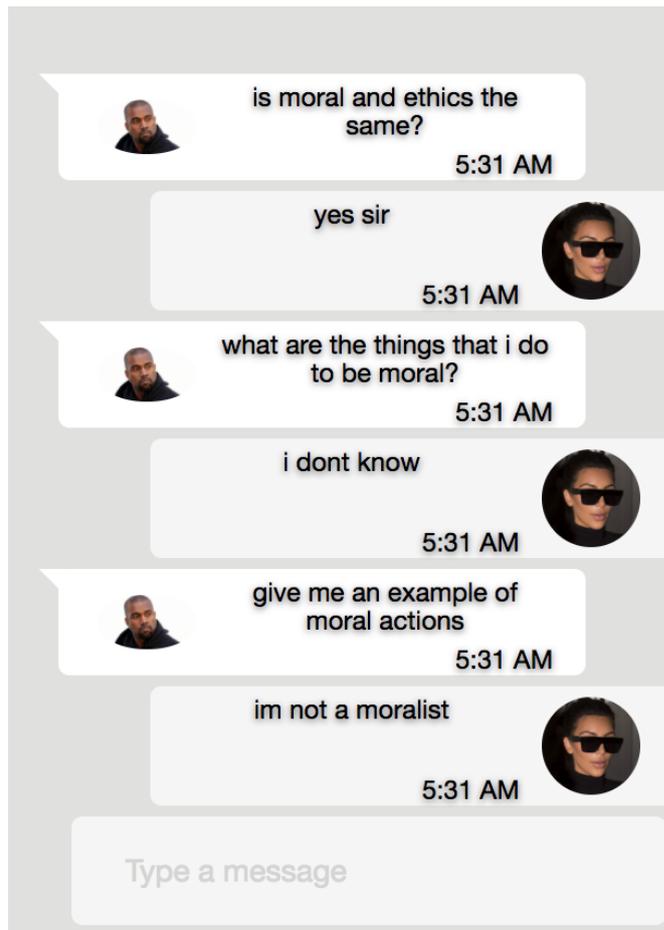


Figura 61 – Comparação Bot, Filmes - Exemplo 1

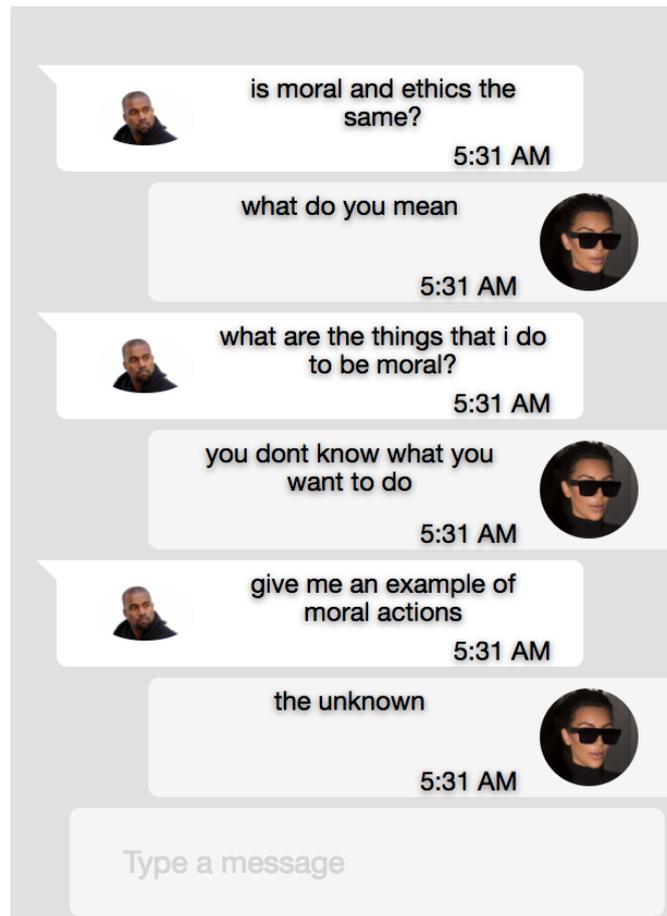


Figura 62 – Comparação Bot, Filmes - Exemplo 2

Agora, é importante comparar também os dois modelos que utilizam o Twitter:

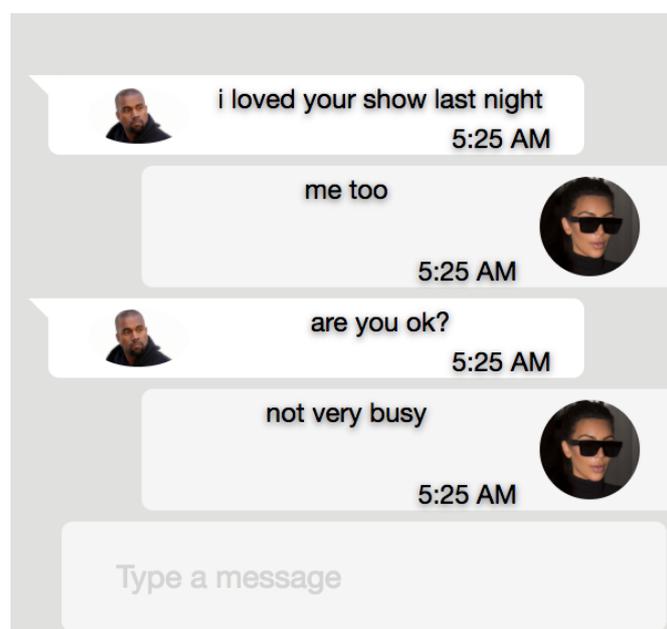


Figura 63 – Comparação Bot, Twitter - Exemplo 1

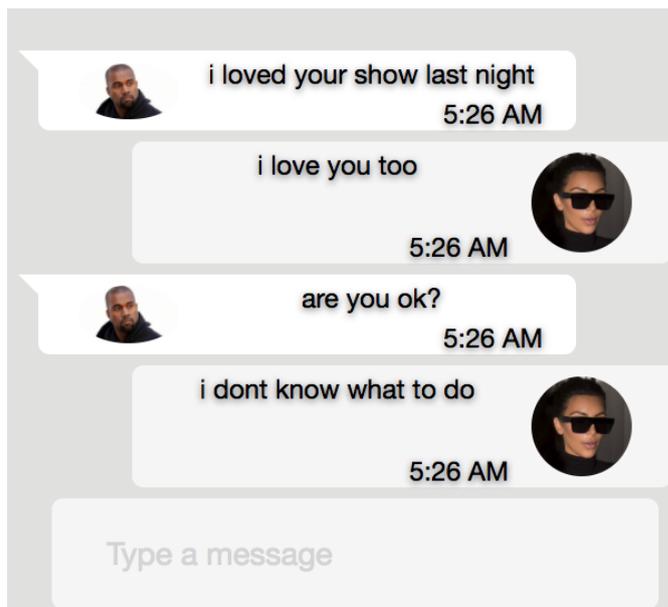


Figura 64 – Comparação Bot, Twitter - Exemplo 2

É possível perceber que a base de dados de diálogo de filmes é mais realista do que as bases dados que utilizam o Twitter. Isso pode ser explicado por inúmeras razões. Primeiramente, a *Cornell Movie Dialogs* possui mais de 200 mil pares, mais de 20 vezes maior que as outras duas juntas. Outro fator importante é que o Twitter não é uma ferramenta com foco em diálogo. Normalmente, as interações no Twitter são repetitivas e não apresentam muito conteúdo, pela própria estrutura da rede que permite apenas 140 caracteres. O modelo do Google é mais realista do que os outros, muito provavelmente pela quantidade de dados utilizado pelo modelo e métodos de pré-processamento utilizados. Entre as duas bases de dados que utilizam o Twitter, o comportamento foi semelhante, mas a base de dados de celebridades possui um comportamento um pouco mais realista, mesmo que com menos pares de informação. Isso, provavelmente, deriva da forma como os dados foram tratados: de 400 mil para apenas 3 mil *tweets*.

Qualitativamente, é possível perceber que provavelmente nenhum dos *chatbots* conseguiria passar em um teste como o de Turing, já que são facilmente identificados como não-humanos. Entretanto, alguns diálogos aparentam similaridade com a forma de contato humano e os campos semânticos se mostram, claramente, relacionados, mesmo que algumas frases não façam sentido completo.

7 Conclusões e Possíveis Trabalhos Futuros

Este trabalho demonstrou as principais aplicações de redes neurais artificiais para a área de Processamento de Linguagem Natural (NLP), focando na exploração de significado em redes sociais. Primeiramente, apresentou-se a fundamentação matemática necessária para o entendimento de Redes Neurais Recorrentes, em particular a rede LSTM, e os princípios fundamentais para a transformação de conjuntos de palavras em vetores.

Para ilustrar os conceitos apresentados, três diferentes aplicações foram geradas, utilizando três diferentes redes sociais. Os princípios trabalhados foram os de Análise Sentimental (IMDB), Normalização (Quora) e Chatbots (Twitter). A metodologia completa dos projetos foram apresentadas: obtenção de dados, vetorização de palavras, escolha de algoritmos e implementação de métodos de aprendizado.

As primeiras aplicações focam em análise textual binária. A primeira aplicação, focada em Análise Sentimental, obteve um resultado acima do benchmark estabelecido de 82%, chegando ao valor final de 86%. Além disso, foi possível comparar a diferença de acurácia entre métodos diferentes de mapeamento vetorial para palavras e técnicas de aprendizado, com diferenças de até 5%. A segunda aplicação mostrou a comparação de métodos diferentes para comparação vetorial. Dois métodos tradicionais foram comparados com o modelo criado a partir de LSTM e Word2Vec. Verificou-se que o modelo mais robusto se comporta de forma bem superior com 69% de acurácia, em comparação aos dois outros modelos mais simples que chegaram em 59%. Por último, foi demonstrada a aplicação de um chatbot. Três diferentes corpos textuais foram utilizados para alimentar a inteligência artificial do bot e a comparação entre os resultados é feita de forma subjetiva.

Implementações futuras poderiam incluir técnicas diferentes de *word embedding* para análise textual binária, outras formas de comparação de vetores e, também, outras variações de Redes Neurais Recorrentes, diferentes das LSTM. Além disso, seria possível implementar o chatbot com a utilização de técnicas de *word embedding* e com a capacidade de aprender durante a conversa e não só durante o período de treinamento.

A área de Processamento de Linguagem Natural com métodos de aprendizado profundo cresceram muito nos últimos anos. Com a combinação de modelos desenvolvidos no final da década de 90 e de modelos de pré-processamento textual desenvolvidos muito recentemente (como Word2Vec, em 2013), as capacidades de aplicações são muito poderosas. O futuro da área, como os trabalhos recentes demonstram, está na criação de métodos de pré-processamento cada vez mais eficientes para que a geração de significado a partir do aprendizado se torne mais rápida e robusta.

Referências

- DAO, V. N.; VEMURI, R. A performance comparison of different back propagation neural networks methods in computer network intrusion detection. University of California, Davis. Disponível em: <<http://web.cs.ucdavis.edu/~vemuri/papers/bp-intrusion%20detection.pdf>>. Citado 2 vezes nas páginas 27 e 28.
- DU, T.; SHANKER, V. K. Deep learning for natural language processing. University of Delaware, 2013. Disponível em: <<https://www.eecis.udel.edu/~vijay/fall13/snlp/lit-survey/DeepLearning.pdf>>. Citado na página 18.
- HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory neural computation. In: . Fakultat fur Informatik Technische Universitat Munchen: [s.n.], 1997. Disponível em: <<http://www.bioinf.jku.at/publications/older/2604.pdf>>. Citado 2 vezes nas páginas 34 e 35.
- LEVY, O.; GOLDBERG, Y. Dependency-based word embeddings. Computer Science Department Bar-Ilan University, 2014. Disponível em: <<https://levyomer.files.wordpress.com/2014/04/dependency-based-word-embeddings-acl-2014.pdf>>. Citado na página 30.
- MACUKOW, B. Neural networks – state of art, brief history, basic models and architecture. Faculty of Applied Mathematics and Information Science, Warsaw University of Technology, 2013. Disponível em: <<http://www.mini.pw.edu.pl/~macukow/wspolne/selection.pdf>>. Citado na página 26.
- MAIND, M. S. B.; WANKAR, M. P. Research paper on basic of artificial neural network. Datta Meghe Institute of Engineering, Technology Research, 2014. Disponível em: <<http://www.ijritcc.org/download/Research%20Paper%20on%20Basic%20of%20Artificial%20Neural%20Network.pdf>>. Citado na página 23.
- MAZUR, M. A step by step backpropagationexample. 2015. Disponível em: <<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>>. Citado na página 27.
- MCCORMICK, C. Word2vec tutorial - the skip-gram model. 2016. Disponível em: <<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>>. Citado na página 32.
- OLAH, C. Understanding lstm networks. In: . [s.n.], 2015. Disponível em: <<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>>. Citado na página 37.
- PEREIRA, S. do L. Processamento de linguagem natural. Universidade de São Paulo, 2013. Disponível em: <http://jeiks.net/wp-content/uploads/2013/10/Processamento_de_Linguagem_Natural.pdf>. Citado na página 19.
- SHAPIRO, J. M.; GENTZKOW, M.; BOXELL, L. Is the internet causing political polarization? evidence from demographics. Brown University, 2017. Disponível em:

<https://www.brown.edu/Research/Shapiro/pdfs/age-polars.pdf>. Citado 2 vezes nas páginas 17 e 29.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. In: . Sirte University: [s.n.], 2014. Disponível em: <https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf>. Citado na página 37.

ZAKARIA, M. A.-S. M.; WANKAR, M. P. Artificial neural network : A brief overview. Sirte University, 2014. Disponível em: <https://pdfs.semanticscholar.org/bfc1/480cf04a263a7a184c94578f7be5fd7f788c.pdf>. Citado na página 25.

ZHAO, L.; CALLAN, J. A generative retrieval model for structured documents. In: . Carnegie Mellon University: [s.n.], 2008. Disponível em: <http://www.cs.cmu.edu/~callan/Papers/cikm08-lezhao.pdf>. Citado na página 56.

Apêndices

APÊNDICE A – Código - Scraper para informações do Twitter - Bot

Créditos parciais para o usuário *bpb27*, onde parte do código de extração foi adaptado:

[<https://github.com/bpb27>](https://github.com/bpb27)

scraper.py

```

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException,
    StaleElementReferenceException
from time import sleep
import json
import datetime

def get_tweets(user):
    # edit these three variables
    #start = datetime.datetime(2009, 3, 1) # year, month, day
    #start = datetime.datetime(2017, 6, 9) # year, month, day
    start = datetime.datetime(2016, 6, 19) # year, month, day
    end = datetime.datetime(2017, 6, 20) # year, month, day

    # only edit these if you're having problems
    delay = 1 # time to wait on each page load before reading the page
    driver = webdriver.Chrome('driver/chromedriver') # options are Chrome()
        Firefox() Safari()

    # don't mess with this stuff
    twitter_ids_filename = 'json/' + user + '.json'
    days = (end - start).days + 1
    id_selector = '.time a.tweet-timestamp'
    tweet_selector = 'li.js-stream-item'

```

```

user = user.lower()
ids = []

def format_day(date):
    day = '0' + str(date.day) if len(str(date.day)) == 1 else str(date.day)
    month = '0' + str(date.month) if len(str(date.month)) == 1 else
        str(date.month)
    year = str(date.year)
    return '-'.join([year, month, day])

def form_url(since, until):
    p1 = 'https://twitter.com/search?f=tweets&vertical=default&q=from%3A'
    p2 = user + '%20since%3A' + since + '%20until%3A' + until +
        'include%3Aretweets&src=typd'
    return p1 + p2

def increment_day(date, i):
    return date + datetime.timedelta(days=i)

for day in range(days):
    d1 = format_day(increment_day(start, 0))
    d2 = format_day(increment_day(start, 1))
    url = form_url(d1, d2)
    print(url)
    print(d1)
    driver.get(url)
    sleep(delay)

    try:
        found_tweets = driver.find_elements_by_css_selector(tweet_selector)
        increment = 10

        while len(found_tweets) >= increment:
            print('scrolling down to load more tweets')
            driver.execute_script('window.scrollTo(0,
                document.body.scrollHeight);')
            sleep(delay)
            found_tweets =
                driver.find_elements_by_css_selector(tweet_selector)
            increment += 10

    print('{} tweets found, {} total'.format(len(found_tweets),

```

```
        len(ids)))

    for tweet in found_tweets:
        try:
            id =
                tweet.find_element_by_css_selector(id_selector).get_attribute('href')
            ids.append(id)
        except StaleElementReferenceException as e:
            print('lost element reference', tweet)

    except NoSuchElementException:
        print('no tweets on this day')

    start = increment_day(start, 1)

    try:
        with open(twitter_ids_filename) as f:
            all_ids = ids + json.load(f)
            data_to_write = list(set(all_ids))
            print('tweets found on this scrape: ', len(ids))
            print('total tweet count: ', len(data_to_write))
    except FileNotFoundError:
        with open(twitter_ids_filename, 'w') as f:
            all_ids = ids
            data_to_write = list(set(all_ids))
            print('tweets found on this scrape: ', len(ids))
            print('total tweet count: ', len(data_to_write))

    with open(twitter_ids_filename, 'w') as outfile:
        json.dump(data_to_write, outfile)

    print('all done here')
    driver.close()

celeb_list = ['justinbieber', 'iansomerhalder']

for celeb in celeb_list:
    get_tweets(celeb)
```

```
import tweepy
from tweepy import OAuthHandler
import re
import math
import json
from time import sleep
import cPickle as pickle
import os
from os.path import splitext

consumer_key = ''
consumer_secret = ''
access_token = ''
access_secret = ''

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)

api = tweepy.API(auth, wait_on_rate_limit=True)

def get_all_tweets(screen_name):
    #initialize a list to hold all the tweepy Tweets
    alltweets = []

    #make initial request for most recent tweets (200 is the maximum allowed
    count)
    new_tweets = api.user_timeline(screen_name = screen_name, count=200)

    #save most recent tweets
    alltweets.extend(new_tweets)

    #save the id of the oldest tweet less one
    oldest = alltweets[-1].id - 1

    #keep grabbing tweets until there are no tweets left to grab
    while len(new_tweets) > 0:
        print "getting tweets before %s" % (oldest)

        #all subsequent requests use the max_id param to prevent duplicates
        new_tweets = api.user_timeline(screen_name =
```

```
        screen_name,count=200,max_id=oldest)

    #save most recent tweets
    alltweets.extend(new_tweets)

    #update the id of the oldest tweet less one
    oldest = alltweets[-1].id - 1

    print "...%s tweets downloaded so far" % (len(alltweets))

    return alltweets

def get_lit_all_tweets(username):
    user = username
    user = user.lower()

    with open('json/' + username + '.json') as f:
        ids = json.load(f)

    print('total ids: {}'.format(len(ids)))

    all_data = []
    start = 0
    end = 100
    limit = len(ids)
    i = int(math.ceil(limit / 100))

    for go in range(i):
        print('currently getting {} - {}'.format(start, end))
        sleep(6) # needed to prevent hitting API rate limit
        id_batch = ids[start:end]
        start += 100
        end += 100
        tweets = api.statuses_lookup(id_batch)
        all_data.extend(tweets)

    return all_data

def clean_string(text):
    text = text.replace("\'", "xxapxx")
    new_text = ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z
```

```

        \t])|(\w+:\/\/\/\S+)", " ", text).split())
new_text = new_text.replace("xxapxx", "\\")

words = new_text.split(' ')
count_words = len(words)

if count_words > 1:
    return new_text
else:
    return None

def get_tweet_pairs(tweets):
    pairs = []
    for tweet_raw in tweets:
        if tweet_raw.is_quote_status:
            tweet_text = tweet_raw.text
            tweet = tweet_raw._json
            if 'quoted_status' in tweet:
                quoted_status = tweet['quoted_status']
                quoted_status_text = quoted_status['text']

                quoted_status_text_clean = clean_string(quoted_status_text)
                tweet_text_clean = clean_string(tweet_text)
                if ((quoted_status_text_clean != None) and (tweet_text_clean != None)):
                    pair = [quoted_status_text_clean, tweet_text_clean]
                    print pair
                    pairs.append(pair)
    return pairs

def save_object(obj, filename):
    with open(filename, 'wb') as output:
        pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)

def load_object(filename):
    with open(filename, 'rb') as _input:
        return pickle.load(_input)

"""
username = 'kimkardashian'
tweets = get_lit_all_tweets(username)
save_object(tweets, 'tweets/' + username + '.pkl')
pairs = get_tweet_pairs(username, tweets)

```

```
print 'total pairs: ' + str(len(pairs))
save_object(pairs, 'data/' + username + '.pkl')
"""

usernames = []
for file_ in os.listdir("json/"):
    file_name,extension = splitext(file_)
    if str(extension) == '.json':
        usernames.append(file_name)

for username in usernames:
    tweets = get_lit_all_tweets(username)
    save_object(tweets, 'tweets/' + username + '.pkl')
    pairs = get_tweet_pairs(tweets)
    print 'total pairs: ' + str(len(pairs))
    save_object(pairs, 'pairs/' + username + '.pkl')
```

APÊNDICE B – Código - IMDB

Créditos parciais para a plataforma Kaggle, onde parte do código de limpeza e vetorização foi adaptado de um tutorial:

[<https://www.kaggle.com/>](https://www.kaggle.com/)

E também para o tutorial por onde aprendi a criar redes LSTM:

[<https://pythonprogramming.net/neural-networks-machine-learning-tutorial/>](https://pythonprogramming.net/neural-networks-machine-learning-tutorial/)

play.py

```

from gensim.models import word2vec
import numpy as np # Make sure that numpy is imported
from sklearn.ensemble import RandomForestClassifier

import pandas as pd
# Import various modules for string cleaning
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
# Download the punkt tokenizer for sentence splitting
import nltk.data
#nltk.download()
import logging
from sklearn.cluster import KMeans
import time

import argparse
from rnn import train_rnn
import cPickle as pickle

#print model.doesnt_match("friends community sherlock her".split())
#print model.most_similar("awful")

def makeFeatureVec(words, model, num_features):
    # Function to average all of the word vectors in a given
    # paragraph
    #
    # Pre-initialize an empty numpy array (for speed)

```

```

featureVec = np.zeros((num_features,),dtype="float32")
#
nwords = 0.
#
# Index2word is a list that contains the names of the words in
# the model's vocabulary. Convert it to a set, for speed
index2word_set = set(model.index2word)
#
# Loop over each word in the review and, if it is in the model's
# vocaublary, add its feature vector to the total
for word in words:
    if word in index2word_set:
        nwords = nwords + 1.
        featureVec = np.add(featureVec,model[word])
#
# Divide the result by the number of words to get the average
featureVec = np.divide(featureVec,nwords)
return featureVec

def getAvgFeatureVecs(reviews, model, num_features):
# Given a set of reviews (each one a list of words), calculate
# the average feature vector for each one and return a 2D numpy array
#
# Initialize a counter
counter = 0.
#
# Preallocate a 2D numpy array, for speed
reviewFeatureVecs = np.zeros((len(reviews),num_features),dtype="float32")
#
# Loop through the reviews
for review in reviews:
    #
    # Print a status message every 1000th review
    if counter%1000. == 0.:
        print "Review %d of %d" % (counter, len(reviews))
    #
    # Call the function (defined above) that makes average feature vectors
    reviewFeatureVecs[counter] = makeFeatureVec(review, model, \
        num_features)
    #
    # Increment the counter

```

```
        counter = counter + 1.
    return reviewFeatureVecs

def review_to_wordlist( review, remove_stopwords=False ):
    # Function to convert a document to a sequence of words,
    # optionally removing stop words. Returns a list of words.
    #
    # 1. Remove HTML
    review_text = BeautifulSoup(review, "lxml").get_text()
    #
    # 2. Remove non-letters
    review_text = re.sub("[^a-zA-Z]", " ", review_text)
    #
    # 3. Convert words to lower case and split them
    words = review_text.lower().split()
    #
    # 4. Optionally remove stop words (false by default)
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        words = [w for w in words if not w in stops]
    #
    # 5. Return a list of words
    return(words)

# *****
# Calculate average feature vectors for training and testing sets,
# using the functions we defined above. Notice that we now use stop word
# removal.

def getDataVecs(train, test, model, num_features):
    clean_train_reviews = []
    for review in train["review"]:
        clean_train_reviews.append( review_to_wordlist( review, \
            remove_stopwords=True ))

    trainDataVecs = getAvgFeatureVecs( clean_train_reviews, model, num_features
        )

    print "Creating average feature vecs for test reviews"
    clean_test_reviews = []
    for review in test["review"]:
        clean_test_reviews.append( review_to_wordlist( review, \
```

```

        remove_stopwords=True ))

testDataVecs = getAvgFeatureVecs( clean_test_reviews, model, num_features )

return trainDataVecs, testDataVecs

def RFTraining(train, trainDataVecs, testDataVecs):
    # Fit a random forest to the training data, using 100 trees
    forest = RandomForestClassifier( n_estimators = 100 )

    print "Fitting a random forest to labeled training data..."
    forest = forest.fit( trainDataVecs, train["sentiment"] )

    # Test & extract results
    result = forest.predict( testDataVecs )

    # Write the test results
    output = pd.DataFrame( data={"id":test["id"], "sentiment":result} )
    output.to_csv( "docs/random_forest_results.csv", index=False, quoting=3 )

    return output

def TFTraining(train_vals, train_labels, test_vals):
    train_labels = np.eye(2)[train_labels.as_matrix(columns=None)]
    result = train_rnn(train_vals, train_labels, test_vals)

    # Write the test results
    output = pd.DataFrame( data={"id":test["id"], "sentiment":result} )
    output.to_csv( "docs/tensor_results.csv", index=False, quoting=3 )

    return output

def save_object(obj, filename):
    with open(filename, 'wb') as output:
        pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)

def load_object(filename):
    with open(filename, 'rb') as _input:
        return pickle.load(_input)

"""
#Train, test and model and number of features

```

```
train = pd.read_csv("data/labeledTrainData.tsv", header=0, \
                    delimiter="\t", quoting=3)

test = pd.read_csv("data/testData.tsv", header=0, delimiter="\t", \
                   quoting=3 )

model = word2vec.Word2Vec.load('features/300features_40minwords_10context')

num_features = len(model.syn0[1,:])

#get the data vectors
trainDataVecs, testDataVecs = getDataVecs(train, test, model, num_features)

save_object(test, 'test.pkl')
save_object(trainDataVecs, 'trainDataVecs.pkl')
save_object(train, 'train.pkl')
save_object(testDataVecs, 'testDataVecs.pkl')
"""

trainDataVecs = load_object('trainDataVecs.pkl')
train = load_object('train.pkl')
test = load_object('test.pkl')
testDataVecs = load_object('testDataVecs.pkl')

#Random Forest Training
#output1 = RFTraining(train, trainDataVecs, testDataVecs)

#Tensor Training
output2 = TFTraining(trainDataVecs, train["sentiment"], testDataVecs)
#print output2

#print output1
#print output2
```

rnn.py

```
import tensorflow as tf
from tensorflow.python.ops import rnn, rnn_cell
import numpy as np
import pickle
import idx2numpy as idx
```

```
#from tensorflow.examples.tutorials.mnist import input_data
#mnist = input_data.read_data_sets("/tmp/data/", one_hot = True)
"""
train_size = 60000
test_size = 10000
images_size = 784
n_classes = 10

train_x_pure = idx.convert_from_file("data/train-images-idx3-ubyte")
train_y_pure = idx.convert_from_file("data/train-labels-idx1-ubyte")

train_x = np.ones((train_size,images_size))
train_y = np.zeros((train_size,n_classes))

for i in range (0,train_size):
    train_x[i] = train_x_pure[i].flatten();
    train_y[i, train_y_pure[i]] = 1

test_x_pure = idx.convert_from_file("data/t10k-images-idx3-ubyte")
test_y_pure = idx.convert_from_file("data/t10k-labels-idx1-ubyte")

test_x = np.ones((test_size,images_size))
test_y = np.zeros((test_size,n_classes))

for i in range (0,test_size):
    test_x[i] = test_x_pure[i].flatten();
    test_y[i, test_y_pure[i]] = 1
"""

hm_epochs = 100
n_classes = 2

batch_size = 60
chunk_size = 25
n_chunks = 12
rnn_size = 500

x = tf.placeholder('float', [None, n_chunks,chunk_size])
y = tf.placeholder('float')
```

```

def recurrent_neural_network(x):
    layer = {'weights':tf.Variable(tf.random_normal([rnn_size,n_classes])),
            'biases':tf.Variable(tf.random_normal([n_classes]))}

    x = tf.transpose(x, [1,0,2])
    x = tf.reshape(x, [-1, chunk_size])
    x = tf.split(0, n_chunks, x)

    lstm_cell = rnn_cell.BasicLSTMCell(rnn_size,state_is_tuple=True)
    outputs, states = rnn.rnn(lstm_cell, x, dtype=tf.float32)

    output = tf.matmul(outputs[-1],layer['weights']) + layer['biases']

    return output

def train_neural_network_acc(train_x, train_y, test_x, test_y):
    prediction = recurrent_neural_network(x)
    cost =
        tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(prediction,y))
    optimizer = tf.train.AdamOptimizer().minimize(cost)

    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())

        for epoch in range(hm_epochs):
            i=0
            while i < len(train_x):
                start = i
                end = i+batch_size
                batch_x = np.array(train_x[start:end])
                batch_x = batch_x.reshape((len(batch_x),n_chunks,chunk_size))
                batch_y = np.array(train_y[start:end])

                sess.run([optimizer, cost], feed_dict={x: batch_x, y: batch_y})
                i+=batch_size

            print epoch

    correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))

```


APÊNDICE C – Código - Quora

Créditos parciais para a plataforma Kaggle, onde parte do código de limpeza e vetorização foi adaptado de um tutorial:

[<https://www.kaggle.com/>](https://www.kaggle.com/)

E também para o tutorial por onde aprendi a criar redes LSTM:

[<https://pythonprogramming.net/neural-networks-machine-learning-tutorial/>](https://pythonprogramming.net/neural-networks-machine-learning-tutorial/)

run.py

```

from gensim.models import word2vec
import numpy as np # Make sure that numpy is imported
from sklearn.ensemble import RandomForestClassifier

import pandas as pd
# Import various modules for string cleaning
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
# Download the punkt tokenizer for sentence splitting
import nltk.data
#nltk.download()
import logging
from sklearn.cluster import KMeans
import time

import argparse
#from rnn import train_rnn
import cPickle as pickle
from scipy.spatial.distance import cosine
from sklearn.metrics import log_loss
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cross_validation import train_test_split
from rnn import train_rnn

embedding_ref = 'gnews/'

```

```
def RFTraining(X_train, X_test, y_train, y_test):
    # Fit a random forest to the training data, using 100 trees
    forest = RandomForestClassifier( n_estimators = 100 )

    print "Fitting a random forest to labeled training data..."
    forest = forest.fit(X_train, y_train)

    #print ('RF accuracy: TRAINING', forest.score(X_test, y_test))

    accu = forest.score(X_test, y_test)
    probs = forest.predict_proba(X_test)
    score = log_loss(y_test, probs)

    return score, accu

def forester(X_train, X_test, y_train, y_test):
    # Fit a random forest to the training data, using 100 trees
    forest = RandomForestClassifier( n_estimators = 100 )

    print "Fitting a random forest to labeled training data..."
    forest = forest.fit(X_train, y_train)

    return forest

def save_object(obj, filename):
    with open(filename, 'wb') as output:
        pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)

def load_object(filename):
    with open(filename, 'rb') as _input:
        return pickle.load(_input)

def cos_sim(a1, a2):
    dists = []
    for q1,q2 in zip(a1,a2):
        try:
            dist = cosine(q1,q2)
            dists.append(dist)
        except:
            dist = 0.25
            dists.append(dist)
```

```
    return dists

def euclidean(a1, a2):
    dists = []
    for q1,q2 in zip(a1,a2):
        try:
            dist = np.nan_to_num(np.linalg.norm(q1-q2))
            dists.append(dist)
        except:
            dist = 0.25
            dists.append(dist)

    return dists

def diff(a1, a2):
    dists = []
    for q1,q2 in zip(a1,a2):
        dist = list(np.nan_to_num(np.array(q1 - q2)))
        dists.append(dist)

    return np.array(dists)

def array_sparse(x):
    x_ = []
    for el in x:
        temp = []
        temp.append(el)
        temp_np = np.array(temp)
        x_.append(temp_np)

    return np.array(x_)

def TFTraining(train_vals, train_labels, test_vals, test_labels):
    train_labels = np.eye(2)[train_labels]
    test_labels = np.eye(2)[test_labels]
    result = train_rnn(train_vals, train_labels, test_vals, test_labels)

    return result

train = load_object(embedding_ref + 'train.pkl')
question1_train_dv = load_object(embedding_ref + 'question1_train_dv.pkl')
```

```

question2_train_dv = load_object(embedding_ref + 'question2_train_dv.pkl')

distances = diff(question1_train_dv, question2_train_dv)
labels = np.array(train['is_duplicate'])

X_train, X_test, y_train, y_test = train_test_split(distances, labels,
    test_size=0.2, random_state=42)
forest = forester(X_train, X_test, y_train, y_test)
save_object(forest, 'wen/forest.pkl')
print 'saved'

#rf_array = RFTraining(X_train, X_test, y_train, y_test)

train = load_object(embedding_ref + 'train.pkl')
question1_train_dv = load_object(embedding_ref + 'question1_train_dv.pkl')
question2_train_dv = load_object(embedding_ref + 'question2_train_dv.pkl')

distances = diff(question1_train_dv, question2_train_dv)
labels = np.array(train['is_duplicate'])

X_train, X_test, y_train, y_test = train_test_split(distances, labels,
    test_size=0.2, random_state=42)
#rf_array = RFTraining(X_train, X_test, y_train, y_test)
#print 'Distance - log loss: ' + str(rf_array[0]) + ' accu: ' +
    str(rf_array[1])

print TFTraining(X_train, y_train, X_test, y_test)

distances = euclidean(question1_train_dv, question2_train_dv)
distances = array_sparse(distances)
labels = np.array(train['is_duplicate'])

X_train, X_test, y_train, y_test = train_test_split(distances, labels,
    test_size=0.2, random_state=42)
rf_array = RFTraining(X_train, X_test, y_train, y_test)

print 'Euclidean - log loss: ' + str(rf_array[0]) + ' accu: ' +
    str(rf_array[1])

```

```
distances = cos_sim(question1_train_dv, question2_train_dv)
distances = array_sparse(distances)
labels = np.array(train['is_duplicate'])

X_train, X_test, y_train, y_test = train_test_split(distances, labels,
    test_size=0.2, random_state=42)
rf_array = RFTraining(X_train, X_test, y_train, y_test)

print 'Cosine Similarity - log loss: ' + str(rf_array[0]) + ' accu: ' +
    str(rf_array[1])
```

rnn.py

```
import tensorflow as tf
from tensorflow.python.ops import rnn, rnn_cell
import numpy as np
import pickle
import idx2numpy as idx

#from tensorflow.examples.tutorials.mnist import input_data
#mnist = input_data.read_data_sets("/tmp/data/", one_hot = True)
"""
train_size = 60000
test_size = 10000
images_size = 784
n_classes = 10

train_x_pure = idx.convert_from_file("data/train-images-idx3-ubyte")
train_y_pure = idx.convert_from_file("data/train-labels-idx1-ubyte")

train_x = np.ones((train_size,images_size))
train_y = np.zeros((train_size,n_classes))

for i in range (0,train_size):
    train_x[i] = train_x_pure[i].flatten();
    train_y[i, train_y_pure[i]] = 1

test_x_pure = idx.convert_from_file("data/t10k-images-idx3-ubyte")
test_y_pure = idx.convert_from_file("data/t10k-labels-idx1-ubyte")

test_x = np.ones((test_size,images_size))
```

```

test_y = np.zeros((test_size,n_classes))

for i in range (0,test_size):
    test_x[i] = test_x_pure[i].flatten();
    test_y[i, test_y_pure[i]] = 1
"""

hm_epochs = 70
n_classes = 2

batch_size = 60
chunk_size = 25
n_chunks = 12
rnn_size = 440

x = tf.placeholder('float', [None, n_chunks,chunk_size])
y = tf.placeholder('float')

def recurrent_neural_network(x):
    layer = {'weights':tf.Variable(tf.random_normal([rnn_size,n_classes])),
            'biases':tf.Variable(tf.random_normal([n_classes]))}

    x = tf.transpose(x, [1,0,2])
    x = tf.reshape(x, [-1, chunk_size])
    x = tf.split(0, n_chunks, x)

    lstm_cell = rnn_cell.BasicLSTMCell(rnn_size,state_is_tuple=True)
    outputs, states = rnn.rnn(lstm_cell, x, dtype=tf.float32)

    output = tf.matmul(outputs[-1],layer['weights']) + layer['biases']

    return output

def train_neural_network_acc(train_x, train_y, test_x, test_y):
    prediction = recurrent_neural_network(x)
    cost =
        tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(prediction,y))
    optimizer = tf.train.AdamOptimizer().minimize(cost)

```

```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())

    for epoch in range(hm_epochs):
        i=0
        while i < len(train_x):
            start = i
            end = i+batch_size
            batch_x = np.array(train_x[start:end])
            batch_x = batch_x.reshape((len(batch_x),n_chunks,chunk_size))
            batch_y = np.array(train_y[start:end])

            sess.run([optimizer, cost], feed_dict={x: batch_x, y: batch_y})
            i+=batch_size

        print epoch

    correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct, 'float'))

    print accuracy.eval({x:test_x.reshape((-1, n_chunks, chunk_size)),
        y:test_y})

def train_rnn(train_x, train_y, test_x, test_y):
    prediction = recurrent_neural_network(x)
    cost =
        tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(prediction,y))
    optimizer = tf.train.AdamOptimizer().minimize(cost)

with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())

    for epoch in range(hm_epochs):
        i=0
        print 'Starting epoch ' + str(epoch+1) + ' out of ' + str(hm_epochs)
        while i < len(train_x):
            start = i
            end = i+batch_size
            batch_x = np.array(train_x[start:end])
            batch_x = batch_x.reshape((len(batch_x),n_chunks,chunk_size))
```

```
batch_y = np.array(train_y[start:end])

sess.run([optimizer, cost], feed_dict={x: batch_x, y: batch_y})
i+=batch_size

print 'Finished epoch: ' + str(epoch+1)

correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct, 'float'))

return accuracy.eval({x:test_x.reshape((-1, n_chunks, chunk_size)),
                      y:test_y})

#res = tf.argmax(prediction, 1)
#return sess.run(res, feed_dict={x:test_x.reshape((-1, n_chunks,
                      chunk_size))})

#train_x = mnist.train.images
#train_y = mnist.train.labels

#test_x = mnist.test.images
#test_y = mnist.test.labels

#train_neural_network_acc(train_x, train_y, test_x, test_y)
#train_rnn(train_x, train_y, test_x)
```
