UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Jorge Luis Murakami Chami

Implementação de autocomplementos para consultas utilizando rede neural LSTM

Uberlândia, Brasil

2022

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Jorge Luis Murakami Chami

Implementação de autocomplementos para consultas utilizando rede neural LSTM

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Marcelo de Almeida Maia

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2022

Agradecimentos

Agradeço a meus pais, meu irmão e meus amigos pelo apoio até o último momento.

Resumo

O grande volume de dados gerado pela sociedade atualmente permitiu que tecnologias de Ciência de Dados e Inteligência Artificial apresentassem soluções eficientes para problemas cada vez mais complexos. Nesse sentido, redes neurais vem dominando o mercado e a pesquisa em vários campos, incluindo recuperação da informação e processamento de linguagem natural. Nesse sentido, redes neurais baseadas em células LSTM se apresentam como uma possível ferramenta na geração de autocomplementos de consultas. Uma rede LSTM foi treinada sobre um conjunto de perguntas e respostas da linguagem Java, a fim de construir um módulo de autocomplementos de consultas para o mecanismo de busca CROKAGE. Ao final do treinamento, ficou evidente que as sugestões geradas não serviam de forma eficiente para integrar à plataforma CROKAGE, e que seria necessário uma base de dados mais extensa para que o modelo atingisse resultados satisfatórios.

Palavras-chave: Recuperação da informação, aprendizagem profunda, processamento de linguagem natural, RNN, LSTM.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

A evolução tecnológica e, mais especificamente, a disseminação do uso da Internet trouxeram mudanças significativas para diversos aspectos da vida em sociedade. Entre elas, é possível destacar a democratização do acesso à informação, pela facilidade de comunicação entre indivíduos do mundo todo e a criação de incontáveis plataformas para divulgação e discussão de dados, artigos, problemas e possíveis soluções relativas às mais variadas áreas do conhecimento.

É importante notar, porém, que essa mesma democratização do acesso resultou em uma gigantesca quantidade de dados disponíveis, tornando essencial a criação e aperfeiçoamento de mecanismos que buscassem as informações mais relevantes para as demandas específicas de cada usuário. Na ciência da computação, o ramo responsável por abordar problemas dessa natureza chama-se Recuperação de Informação, e sua aplicação de maior destaque são os mecanismos de busca. Nas últimas décadas, muito se conquistou no sentido de melhorar a experiência dos usuários ao utilizar um serviço de busca, como o retorno de resultados personalizados (FANG; YU; MENG, 2004) e a sugestão de autocomplementos para a busca (WEININGER et al., 2013).

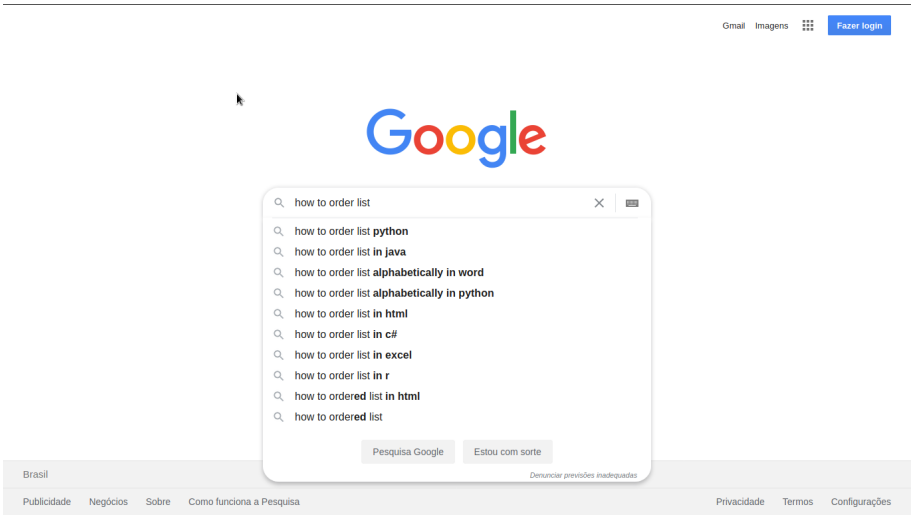


Figura 1 – Sugestão de autocomplementos na plataforma *Google*.

Entre a comunidade de desenvolvimento de *software*, destaca-se o fórum *StackOverflow* destinado a discussão e resolução de problemas relacionados a computação e programação em geral. Aberto para profissionais e amadores entusiastas de desenvolvimento, o site conta com centenas de milhares de tópicos de discussão sobre diferentes bibliotecas, linguagens, algoritmos e conceitos de computação. No entanto, o fato de a informação advir das respostas dos próprios usuários, nem sempre as perguntas são respondidas de forma clara e com riqueza de detalhes, tanto em aspectos teóricos quanto práticos. Isso

resulta num cenário em que a ferramenta de busca incluída na própria plataforma não traz resultados tão satisfatórios quanto poderia para as questões dos usuários. Muitas vezes, os usuários chegam até as respostas por mecanismos de busca tradicionais como o Google em vez de utilizar o serviço oferecido pelo *StackOverflow*.

Para começar a solucionar esse problema, surgiu o CROKAGE (SILVA et al., 2019), ferramenta de busca para resolução de dúvidas de desenvolvimento de *software* que opera sobre a base de dados de perguntas e respostas do *StackOverflow*. O diferencial do CROKAGE está na construção das soluções para problemas buscados. A ferramenta agrega respostas de usuários que contém soluções teóricas a outras respostas que contém soluções práticas e trechos de código-fonte, gerando um compilado rico em detalhes e exemplos para abordar os problemas investigados. Até o momento, o CROKAGE se restringe a perguntas sobre a linguagem Java e não implementa serviços adicionais de experiência de usuário, como busca personalizada e sugestão de autocomplementos para consultas.

Além da renovação da importância da Recuperação de Informação, os últimos anos trouxeram também grandes inovações nas áreas de Inteligência Artificial. A evolução do hardware e a popularização das redes neurais permitiram com que modelos estatísticos aproveitassem da enorme quantidade de dados disponível para atingir resultados inéditos em várias tarefas das mais diversas áreas de atuação. A sobreposição entre essas duas áreas da computação resultou no surgimento da chamada Recuperação Neural de Informação (MITRA; CRASWELL et al., 2018).

Neste projeto, pretende-se validar a eficácia da utilização de técnicas de aprendizagem profunda para a sugestão de auto-complementos de consultas em mecanismos de busca. Para tanto, será implementado um modelo de aprendizado de máquina baseado em redes neurais de recorrência (RNN), que poderá ser integrado ao CROKAGE. Por fim, será feita uma análise qualitativa dos resultados do modelo.

1.1 Objetivos

O objetivo geral deste trabalho é construir uma solução de autocomplementos de consultas para a ferramenta de busca CROKAGE cujas sugestões extrapolem recomendações em nível estritamente sintático. A hipótese é que algoritmos de aprendizado de máquina e aprendizagem profunda obtenham resultados que levem em consideração o contexto e a semântica das consultas, em contraste à modelos estatísticos de Recuperação de Informação que focam em características léxicas e sintáticas, como modelos de n-gramas. Os objetivos específicos deste trabalho são:

- Investigar métodos de sugestão de autocomplementos de consultas existentes na

literatura;

- Implementar módulos de sugestão de autocomplementos de consultas treinados sobre a base de dados de dúvidas sobre a linguagem Java do fórum *StackOverflow*;
- Fazer uma análise qualitativa sobre os resultados gerados para avaliar a incorporação do módulo de autocomplementos de consulta ao CROKAGE;

1.2 Justificativa

A principal contribuição deste trabalho seria a implementação de um módulo eficiente de sugestão de autocomplementos de consultas e a sua integração à ferramenta CROKAGE. O modelo será útil para a experiência de usuários ao utilizar a ferramenta, de forma a facilitar a delimitação da demanda e sua tradução em uma consulta que resulte na recuperação dos documentos mais relevantes. Além disso, os modelos descritos são de fácil reutilização para aplicação em outras tarefas semelhantes, podendo ser integrados a mecanismos de busca diferentes, caso seja confirmada a eficácia do método utilizado.

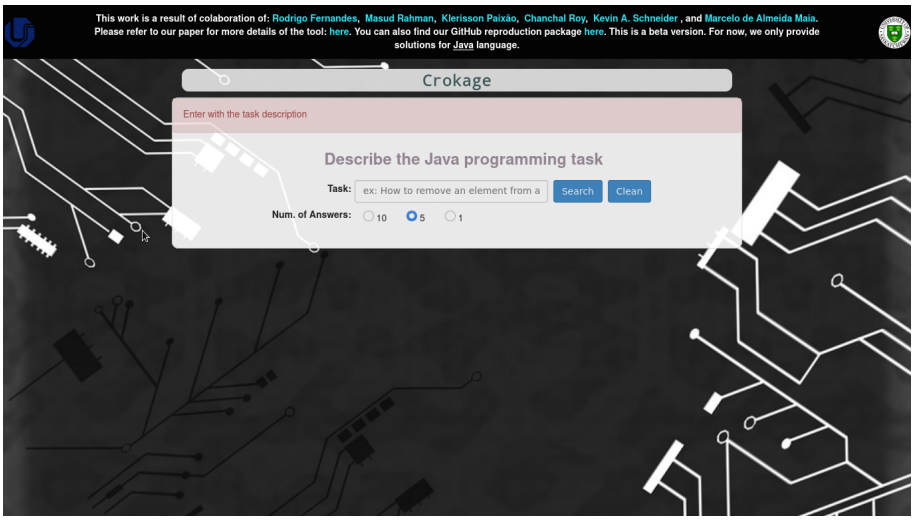


Figura 2 – Página da ferramenta de buscas CROKAGE.

2 Fundamentação Teórica

Este capítulo apresenta os conceitos e técnicas de Recuperação de Informação, Redes Neurais e Processamento de Linguagem Natural necessários para a compreensão do trabalho realizado.

2.1 Recuperação de Informação

A Recuperação da Informação é a área que surge para lidar com representação, armazenamento, organização e acesso à informações em um conjunto de dados não estruturados. É importante salientar que, diferente de uma simples consulta de dados, usuários de um sistema de Recuperação de Informação estão preocupados em captar informações, contidas nos dados, que satisfaçam suas necessidades (BAEZA-YATES; RIBEIRO-NETO et al., 1999).

Em recuperação de texto (que engloba a maior parte das aplicações de RI, além de ser o foco deste trabalho), uma busca ou consulta não consiste em estruturas complexas de texto, e sim em palavras-chave que sumarizem o que está sendo procurado (HAN; PEI; KAMBER, 2011). O objetivo do sistema é trazer documentos, podendo ter formatos e tamanhos variados, que atendam aos requisitos da consulta. Em uma base de dados, todos os termos que podem ser relevantes para uma consulta compõem um vocabulário.

A frequência de cada termo em cada documento é importante para calcular a relevância deste para uma consulta e, portanto, justifica a utilização de alguma estrutura de dados (como uma matriz) para mapear essa relação. No entanto, à medida que uma base de dados e seu vocabulário crescem, é importante que o texto seja processado a fim de diminuir o custo computacional de se construir a referida estrutura. Nesse sentido, é comum que se desconsiderem palavras muito comuns e genéricas do idioma (denominadas *stopwords*) (LUHN, 1960), como artigos e algumas preposições. Além disso, é feita a extração de radicais das palavras (LOVINS, 1968), a fim de eliminar palavras similares e conjugações, preservando o significado geral.

Além do formato e processamento de consultas, técnicas de Recuperação de Informação podem implementar diferentes alternativas para representação dos termos, medição de similaridade entre documentos e até os próprios critérios de correspondência entre documentos e consultas. Para o restante deste trabalho, é importante mencionar a técnica de *word embeddings*, que será definida com mais detalhes na Seção 2.3.

2.1.1 CROKAGE

Como exposto anteriormente, a ferramenta CROKAGE (*CROwd Knowledge Answer GEnerator* visa resolver tarefas e dúvidas de programação na linguagem Java, agregando explicações teóricas de soluções a trechos de código a partir dos dados disponíveis no fórum *StackOverflow* (SILVA et al., 2019). A construção da ferramenta envolveu a utilização de técnicas avançadas para a resolução de diversas tarefas de Recuperação da Informação.

Na construção e preparação do conjunto de documentos (denominado *corpus*), foram utilizados *word embeddings* para conversão do texto em representações numéricas, além da criação de índices para facilitar a busca posterior de respostas relevantes. O levantamento de soluções candidatas é feito utilizando a função de classificação *BM25* (ROBERTSON; WALKER, 1994). Posteriormente, tais soluções são submetidas ao cálculo de quatro métricas de similaridade (similaridades léxica, semântica e duas outras métricas não relacionadas ao texto em si), que definem os melhores pares pergunta-resposta para a tarefa especificada.

2.2 Redes Neurais

Redes Neurais Artificiais são modelos computacionais que tentam imitar a estrutura do cérebro humano para processar dados e resolver tarefas de Inteligência Artificial e Aprendizado de Máquina. Apesar de as primeiras implementações de RNAs datarem da metade do século XX (MCCULLOCH; PITTS, 1943), tal técnica veio se tornar popular apenas no fim da primeira década do século seguinte. O aumento da capacidade de processamento das máquinas modernas, em conjunto à grande quantidade de dados disponíveis, possibilitou que surgissem novas arquiteturas de redes neurais artificiais, capazes de superar até o desempenho humano em diversas tarefas (HE et al., 2015; MNIH et al., 2015).

Para entender o funcionamento das Redes Neurais Artificiais modernas, é fundamental entender o conceito de percéptron de múltiplas camadas. O restante desta seção explicará os fundamentos deste algoritmo, além de apresentar a arquitetura principal de RNAs mais relevante para o restante do trabalho.

2.2.1 Percéptrons

Um percéptron por si só já é um classificador binário, além de ser a unidade básica de qualquer RNA e, em geral, é composto por três componentes principais, sendo eles um conjunto de pesos sinápticos, uma função de transferência e uma função de ativação. A operação de um percéptron j, como mostrado na Figura 3 pode ser descrita em três etapas:

1. Multiplicação dos valores de entrada $x_0..x_n$ pelos respectivos valores dos pesos sinápticos $w_0j..w_nj$;

2. Aplicação da função de transferência Σ (geralmente uma simples soma) aos valores resultantes da multiplicação anterior;

3. Aplicação da função de ativação ϕ ao resultado da função de transferência net_j, utilizando algum tipo de limiar θ_j.

O resultado o_j (geralmente 0 ou 1) então pode ser usado para cumprir a tarefa de classificação binária para determinado conjunto de dados. Vale notar que é possível também cumprir também a tarefa de classificação multiclasses, desde que se escolha uma função de ativação cujo resultado final indique a classe a qual o resultado pertence.

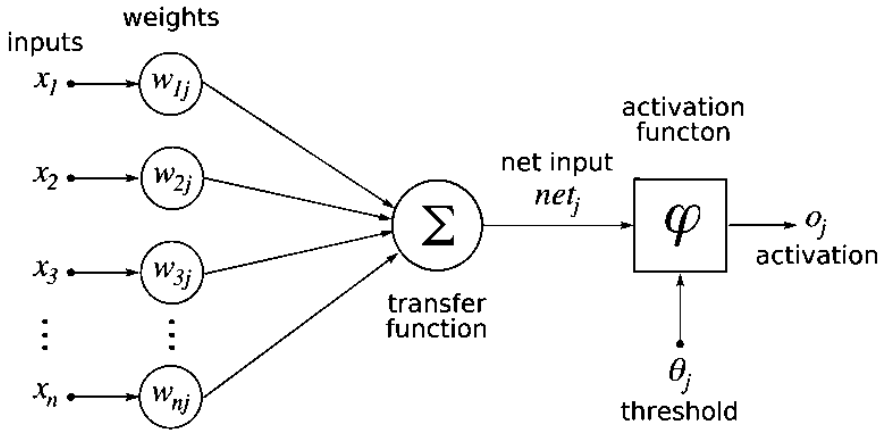


Figura 3 – Modelo de neurônio artificial - Extraída de (COMMONS, 2005)

A partir do percéptron simples proposto em (ROSENBLATT, 1958), o próximo passo naturalmente seria a construção de percéptrons de múltiplas camadas. A evolução deste sobre o percéptron simples está no fato de que todos os valores de uma determinada entrada passam por todos os neurônios de uma chamada "camada oculta"que, por fim, produz o resultado final. A Figura 4 facilita a visualização deste processo, no qual todos os valores de cada nó na camada de entrada (*Input Layer*) estão conectados a todos os nós da camada oculta (*Hidden Layer*).

A partir da estrutura apresentada na Figura 4, percebe-se que é tarefa relativamente fácil adicionar mais camadas ocultas entre a entrada e saída, promovendo uma maior abstração dos valores iniciais e permitindo com que o os percéptrons encontrem relações estatísticas cada vez mais sofisticadas entre os dados. Tal ideia resume a área de Aprendizagem Profunda, que domina diversos setores de pesquisa na atualidade, com RNAs - a partir daqui, os termos "Redes Neurais Artificiais"e "Percéptrons de Múltiplas Camadas"serão utilizados de forma equivalente - de dezenas ou até centenas de camadas (HE et al., 2016; HUANG et al., 2017).

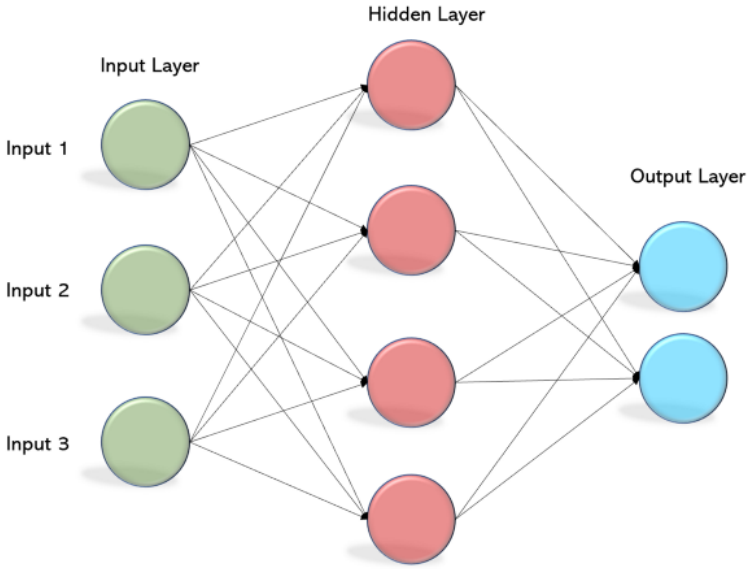


Figura 4 – Modelo de Percéptron de Múltiplas Camadas - Extraída de (MOHANTY, 2019)

2.2.1.1 Treinamento de uma RNA

Para construir um classificador baseado em Redes Neurais Artificiais, primeiramente é necessário treinar a rede sobre um conjunto de dados. Esse treinamento é um processo de otimização que visa atualizar automaticamente os pesos sinápticos em todas as camadas para reduzir o erro, e consiste de duas etapas: *feedforward* e *backpropagation*.

Na etapa de *feedforward*, os dados são processados sequencialmente pela rede, como em uma predição comum. Os dados do vetor de entrada são processados pelos neurônios da primeira camada oculta, e os resultados desse processamento alimentam os neurônios da segunda camada oculta, e assim por diante até alcançar a camada de saída (com a predição de uma classe para cada elemento do conjunto de dados). Esses resultados então são comparados com os valores de referência do conjunto de de dados, e o erro é calculado.

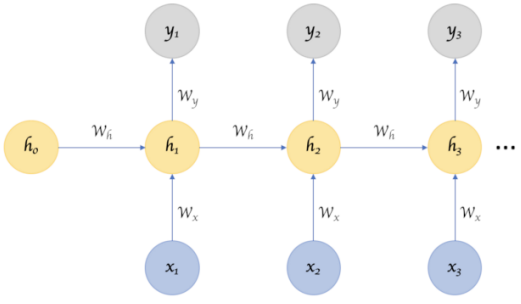
O passo seguinte, de *backpropagation*, utiliza o erro calculado para encontrar um valor de ajuste ideal para os pesos de cada neurônio em cada camada. Para isso, ele utiliza os conceitos de cálculo de gradientes e regra da cadeia para encontrar a influência de cada neurônio no valor do erro e, assim, fazer o ajuste de acordo. Em resumo, o processo de *backpropagation* consiste na aplicação do método do gradiente (LEMARÉCHAL, 2012) para redução do erro em função dos pesos sinápticos de toda a rede.

As duas etapas se repetem por um número máximo de iterações ou até que o erro convirja para um valor aceitável pré-determinado.

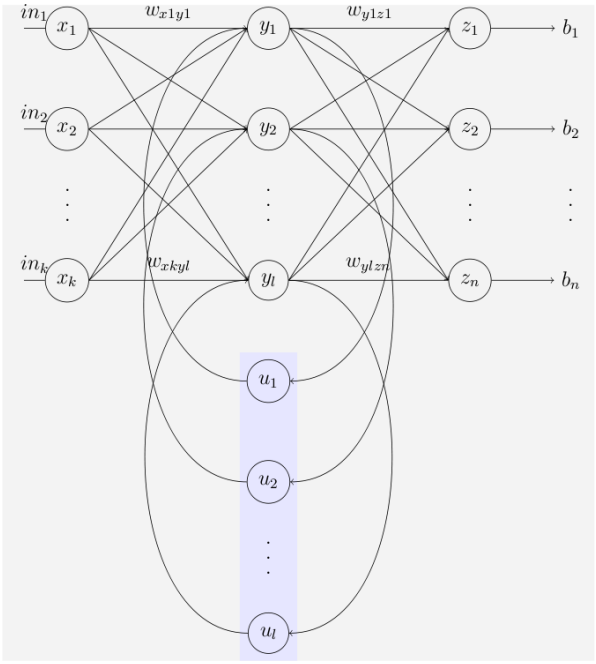
2.2.2 Redes Neurais de Recorrência

Ao tratar de dados cujos elementos influenciam uns nos outros em uma linha temporal (como sequências de texto, áudio e vídeo), uma RNA comum não apresentaria uma performance tão boa devido ao fato de não conseguir captar essas dependências temporais de forma efetiva. Para isso, uma variação da arquitetura básica de redes neurais foi projetada para incluir um elemento de memória no processamento das bases de dados. São as chamadas Redes Neurais de Recorrência.

Nas primeiras Redes Neurais de Recorrência (RUMELHART; HINTON; WILLIAMS, 1986), o elemento de memória se dava na própria saída de cada camada oculta, em um esquema de "retroalimentação"da camada oculta da rede, como é possível observar na Figura 5b. Dito isso, é importante lembrar que o dados que alimentam o treinamento de uma rede de recorrência são sequenciais. Isso significa que o resultado do processamento na camada oculta h_1 para um elemento x_1 é utilizado como entrada para esse mesmo processamento h_2 em conjunto com o elemento x_2, e assim por diante. Por causa disso, é comum que Redes Neurais de Recorrência sejam modeladas em sua forma "desdobrada"no tempo, conforme mostra a Figura 5a. É importante salientar que, apesar de parecer maior e mais complexa à medida que o tempo passa, as matrizes de pesos sinápticos são as mesmas em cada passo temporal, ou seja, a complexidade não é necessariamente maior do que em uma RNA comum.



(a) Modelo de Rede Neural de Recorrência Desdobrada - Extraída de (VENKATACHALAM, 2019)



(b) Modelo de Rede Neural de Recorrência Simples - Extraída de (COMMONS, 2009)

Figura 5 – Representações de Redes Neurais de Recorrência

O problema de redes de recorrência mais simples é que, à medida que o treina-

mento ocorre, dados de processamento de passos muito anteriores no tempo passam a ter uma influência desprezível no treinamento, devido ao cálculo sucessivo dos gradientes em função dos pesos. Esse é o chamado problema do desaparecimento do gradiente (*vanishing gradient problem*). Uma solução para isso foi encontrada em 1995, com a criação de células de *Long Shot-Term Memory*, ou LSTM (HOCHREITER; SCHMIDHUBER, 1997). Em uma rede de recorrência simples, o cálculo feito é o mesmo de uma RNA comum, ou seja, multiplicação e soma dos pesos seguida de uma função de ativação. Uma célula LSTM é capaz de guardar informações a longo e curto prazo, efetivamente garantindo que dados de passos temporais anteriores mantenham alguma influência no processo de treinamento.

2.2.2.1 Funcionamento da célula LSTM

A Figura 6 apresenta uma representação do funcionamento interno de uma célula LSTM. Percebe-se que em qualquer célula, temos três entradas, que correspondem à entrada atual x_t e às memórias de curto prazo h_{t-1} e longo prazo c_{t-1}. Em cada nó, existem quatro etapas responsáveis pelo processamento dos dados e manutenção das memórias de curto e longo prazo, que resultam nas novas memórias h_t e c_t. Tais etapas são compostas por diferentes aplicações das funções sigmóides σ e tangente hiperbólica *tanh* a combinações lineares entre memórias anteriores de curto e longo prazo, dados da entrada atual e, principalmente, cinco matrizes diferentes de pesos que serão ajustadas no processo de *backpropagation*.

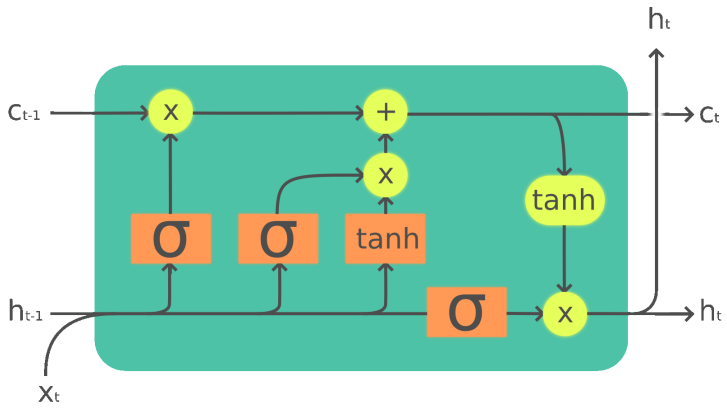


Figura 6 – Diagrama de Célula LSTM - Adaptada de (COMMONS, 2018)

2.3 Processamento de Linguagem Natural

A área de Processamento de Linguagem Natural, ou PNL, é a interseção entre as áreas de Linguística e Ciência da Computação, solucionando problemas relacionados ao entendimento e/ou a geração de construções em linguagem natural por computadores a partir de um conjunto de dados, ou *corpus*. Reconhecimento de fala, tradução automática

e análise de sentimentos são exemplos de tarefas comuns de PNL. É importante entender dados não estruturados em linguagem natural podem ser representados numericamente, a fim de possibilitar o uso de algoritmos e modelos estatísticos que resolvam os problemas desejados. Como mencionado na seção 2.1, os modelos mais importantes para este trabalho são os *word embeddings*.

2.3.1 *Embeddings*

Embeddings são técnicas mapeamento de unidades textuais (podendo ser palavras, sentenças etc.) em vetores de números reais, mantendo noções de similaridade. Isso significa que, no caso de um *word embedding*, palavras com significados semelhantes ocupam posições próximas no espaço vetorial do mapeamento. Atualmente, uma das técnicas mais populares de *embeddings* para palavras é o algoritmo *Word2Vec*, baseado em uma RNA de apenas uma camada oculta (MIKOLOV et al., 2013a; MIKOLOV et al., 2013b).

Após o treinamento do *Word2Vec* sobre um extenso *corpus*, o resultado é um espaço vetorial de alta dimensionalidade no qual cada palavra do vocabulário corresponde a apenas um vetor. A diferença desse algoritmo para qualquer outro modelo de RNA está no fato de que os próprios pesos sinápticos correspondem ao espaço vetorial, ou seja, o processo de treinamento não ocorre visando a classificação em si, mas a utilização da matriz de pesos como uma tabela de mapeamento. Ao se utilizar o modelo completo, é possível solucionar a tarefa de modelos de linguagens, ou seja, predição da próxima palavra dada uma sequência de palavras anteriores.

Isso significa que esse algoritmo é muito poderoso para modelar construções textuais em domínios específicos, dado que o contexto das palavras dentro do *corpus* desse domínio será preservado posteriormente no espaço vetorial. A Figura 7 demonstra essa manutenção de similaridade com uma projeção de vetores, gerados pelo algoritmo *Word2Vec*, utilizando Análise de Componentes Principais. A distância entre os vetores que representam os países é os vetores que representam suas capitais é similar para todo o *corpus*.

2.4 Trabalhos Relacionados

Considerando que a tarefa de sugestão de autocomplementos para consulta pode ser generalizada para uma tarefa comum de modelos de linguagem (*language modeling*), existem inúmeros trabalhos sobre diversas bases de dados que visam construir modelos abrangentes de geração de texto. Sendo assim, é mais interessante para o trabalho mencionar outros estudos que tratem de problemas de PNL relacionados a bases de código-fonte, ou que utilizem técnicas de RNAs para construção, extensão e/ou otimização de modelos de linguagem para outros domínios.

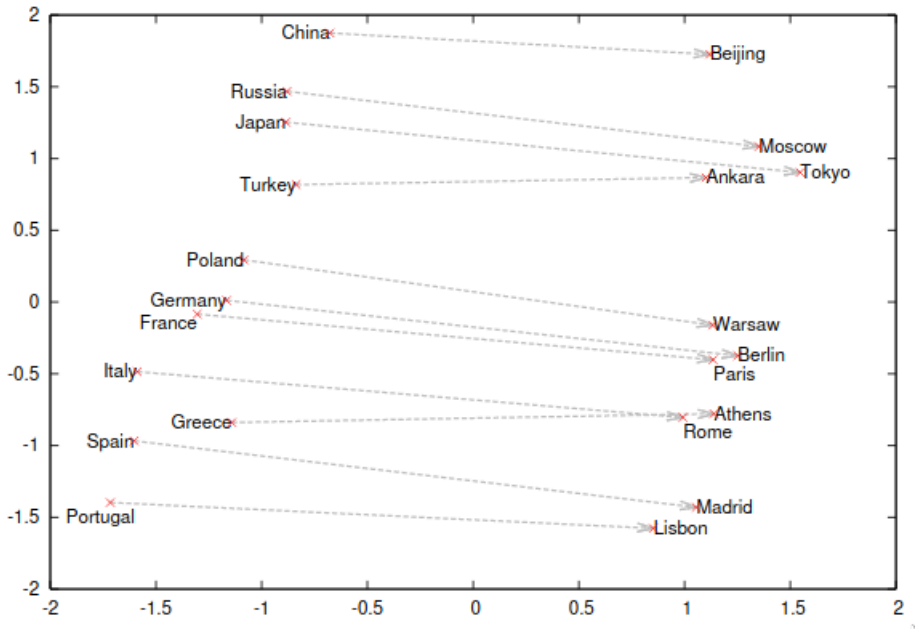


Figura 7 – Projeção de Vetores Gerados pelo *Word2Vec* - Adaptada de (MIKOLOV et al., 2013b)

Em (MERITY; KESKAR; SOCHER, 2017), são investigadas maneiras de se regularizar e otimizar o treinamento de Redes Neurais de Recorrências baseadas em células LSTM. A técnica DropConnect (WAN et al., 2013) - semelhante ao *dropout* empregado em Redes Neurais Convencionais - é utilizada para regularização de pesos entre camadas ocultas, e um novo método de otimização é proposto, chamado NT-ASGD. O método é uma variação do método do gradiente estocástico médio, na qual o cálculo da média ocorre segundo um critério escolhido automaticamente, e não pelo usuário.

Em (Wong; Jinqiu Yang; Lin Tan, 2013), um conjunto de dados de um fórum de perguntas e respostas foi minerado para alimentar um modelo de geração automática de comentários explicativos para códigos na linguagem Java e para programação Android. Em (VAN; KAUCHAK; LEROY, 2020), é apresentado um modelo de autocomplementos para simplificação de textos médicos baseados em redes neurais pré-treinadas, como BERT (DEVLIN et al., 2018) e GPT-2 (RADFORD et al., 2018).

Em (HUSAIN et al., 2019), é apresentado um conjunto de dados de consultas em linguagem natural e mais de seis milhões de soluções candidatas em código-fonte, cujo objetivo é promover a criação de modelos melhores para a tarefa de recuperação de código dada uma consulta em linguagem natural. O contrário ocorre em (ALON et al., 2018), no qual é proposto um modelo para gerar construções em linguagem natural a partir de trechos de códigos-fonte, o que pode contribuir imensamente na geração de documentação, sumarização e recuperação posterior dos próprios trechos de código.

3 Metodologia

Este capítulo apresenta as etapas necessárias para a implementação do modelo computacional escolhido, descreve as etapas de extração e tratamento dos dados utilizados, e demarca os passos para integração do módulo ao CROKAGE.

3.1 Base de Dados

O *corpus* a ser utilizado no trabalho foi obtido a partir da API da ferramenta CROKAGE que, por sua vez, extraiu os dados de perguntas e respostas da plataforma *StackOverflow* (SILVA et al., 2019). Pela API, foi possível obter um conjunto de 3572 amostras com um vocabulário de 23287 palavras distintas - consistindo em título, corpo da pergunta e corpo das respostas - relacionadas à linguagem Java. Vale notar que, antes de serem incluídos no banco de dados do CROKAGE, os registros passam por etapas de pré-processamento, como conversão para letras minúsculas e remoção de *stopwords*. Os dados brutos também incluem alguns valores estatísticos que não serão utilizados neste trabalho, restando apenas dados textuais, como mostra a figura 8.

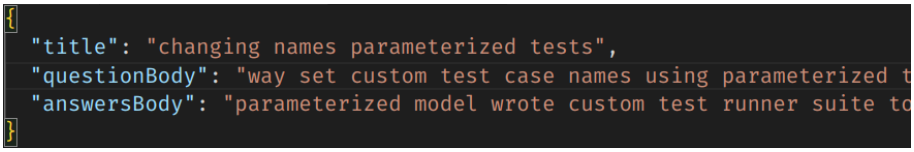


Figura 8 – Exemplo de registro do *corpus* escolhido, no formato JSON.

3.2 Pré-processamento

O pré-processamento da base de dados envolve transformar palavras em valores numéricos. Para isso, uma versão do algoritmo de *embedding* Word2Vec implementada pela biblioteca *gensim* foi treinada sobre a base de dados, permitindo a representação do vocabulário de mais de vinte mil palavras em vetores de apenas 100 dimensões. O algoritmo permite explorar os dados matematicamente, preservando a similaridade semântica entre palavras dentro do espaço vetorial. Nas Figuras 9 e 10, é possível visualizar, primeiramente, uma lista de palavras consideradas pelo modelo como mais similares à palavra "Java", bem como uma representação bidimensional do espaço vetorial populado pelo *corpus*.

```
43  print(w2v_model.wv.most_similar('java'))
44
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

❯ python word2vec.py
[('xml', 0.8503807783126831), ('list', 0.836831271648407), ('use', 0.8240327835083
008), ('example', 0.8129631280899048), ('file', 0.811137855052948), ('project', 0.
8029367923736572), ('using', 0.7978383302688599), ('methods', 0.7958458065986633),
 ('variable', 0.7895363569259644), ('also', 0.7826399803161621)]
```

Figura 9 – Palavras mais similares à palavra "Java".

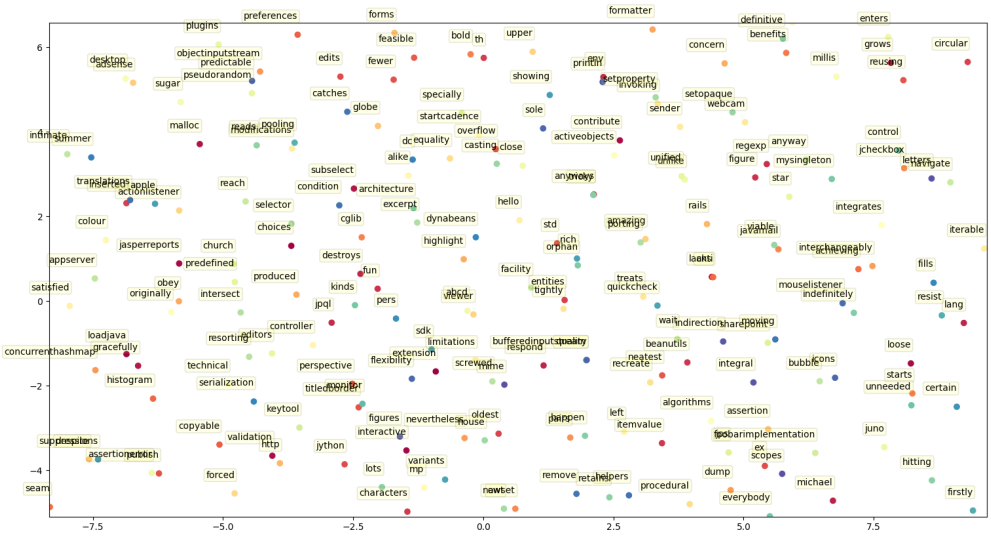


Figura 10 – Representação bidimensional de uma amostra do *corpus*.

3.3 Rede Neural

Para alimentar a rede neural, as amostras do corpus são separadas em sentenças de comprimento fixo (em número de palavras), que serão usadas como dados de entrada no treinamento. A Figura 11 ilustra a rede neural implementada do tipo LSTM, que consiste em apenas três camadas:

1. Uma camada de **embeddings**, onde serão utilizados os pesos do algoritmo Word2Vec pré-treinado

2. Uma camada de células do tipo LSTM do mesmo tamanho do espaço vetorial gerado pela camada anterior

3. Uma camada de saída do tamanho de todo o vocabulário que, ativada pela função *softmax* (BOLTZMANN; HASENÖHRL, 2012), que converte os resultados em uma distribuição probabilística escolhendo, efetivamente, a próxima palavra da sequência.

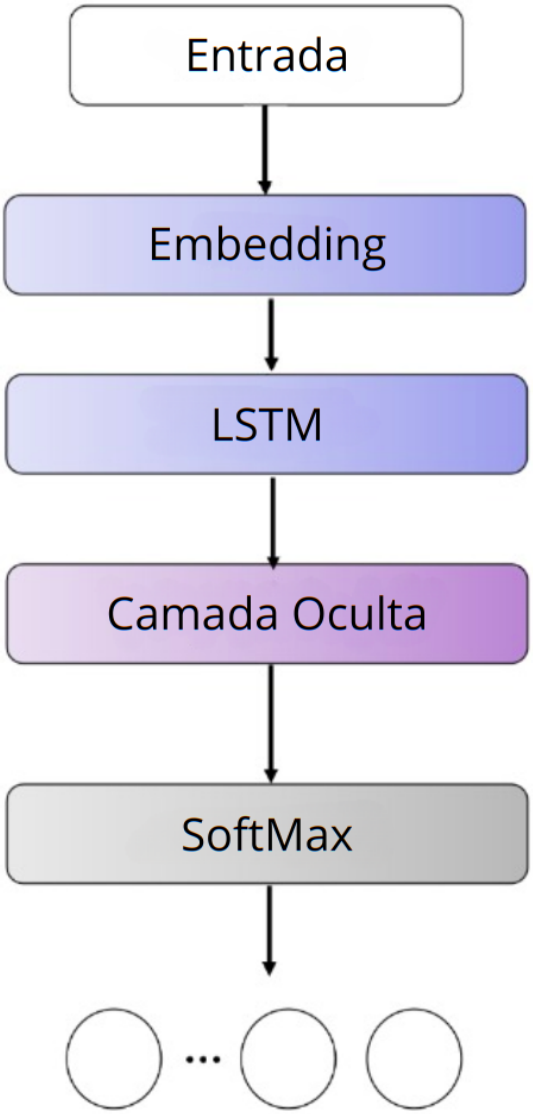


Figura 11 – Arquitetura da rede LSTM.

O modelo foi treinado por 100 épocas sobre toda a base de dados. O otimizador Adam (KINGMA; BA, 2014) foi escolhido para minimizar a função de erro entropia cruzada, que compara os resultados do modelo com o texto real do *corpus* em termos de distribuições de probabilidade.

4 Resultados

Neste capítulo serão apresentados os resultados do treinamento do modelo, bem como uma análise qualitativa das frases efetivamente geradas por este, para efeitos de complementar a consulta de um usuário.

Para testar a geração de palavras do modelo, foram construídas dez consultas relativas a diferentes tarefas de programação, seguindo uma mesma estrutura: o nome de uma API popular no domínio da tarefa apresentada, seguida das palavras base *how to*, terminando em um verbo também relacionado à tarefa. Tais consultas foram alimentadas ao modelo e também ao mecanismo de busca *Google* para efeitos de comparação e, para ambos, foi escolhido apenas o complemento mais provável, ou seja, o primeiro da lista. Foram tomadas como parâmetro as consultas presentes em (ROCHA; MAIA, 2022), adaptadas para a estrutura descrita.

É importante notar primeiramente que a palavra *Java* também foi incluída nas consultas ao *Google*, não sendo necessária para a rede LSTM visto que foi treinada em uma base de dados específica da linguagem. Além disso, para o modelo, as palavras "*how to*"são efetivamente ignoradas pois a base em questão foi pré-processada com a remoção de *stopwords*. A Tabela 1 apresenta as consultas, bem como as sugestões de complementos do modelo LSTM e do *Google*, destacadas em negrito.

Tabela 1 – Comparativo de Consultas

Consulta Original	**Modelo LSTM (e=0.0160)**	**Google**
java spring how to upload image	spring how to upload **service security**	java spring how to upload **large files**
java jpa how to implement crud operations	jpa how to implement **hibernate table**	java jpa how to implement **it**
java selenium how to search a product item on an e-commerce web application	selenium how to search **labels form**	java selenium how to search **for text**
java tensorflow how to build a neural network	tensorflow how to build **(sem sugestão)**	java tensorflow how to build **a table**
java junit how to implement matchers test	junit how to implement **also tests**	java junit how to implement **test cases**
java javafx how to implement menu	javafx how to implement **process process**	java javafx how to implement **it**
java opencv how to classify image	opencv how to classify **(sem sugestão)**	java opencv how to classify **images**
java jackson how to process JSON data	jackson how to process **copy running**	java jackson how to process **json**
java libgdx how to implement animation	libgdx how to implement **(sem sugestão)**	java libgdx how to implement **it**
java opengl how to draw 2D objects	opengl how to draw **skip colors**	java opengl how to draw **line**

Nas consultas relativas a *Selenium* e *JUnit*, o modelo foi capaz de gerar complementos coerentes e pertinentes ao domínio (apesar de, no caso da segunda, o complemento *"also"* não contribuir para a estrutura da frase. Em outras, como *Spring*, *Jackson* e *OpenGL*, os complementos estão dentro do domínio explorado, porém não formam uma consulta dotada de sentido completo, tal que seria necessário a geração de mais palavras para dar contexto adicional. Já nos casos de *JPA*, *TensorFlow*, *JavaFX*, *OpenCV* e *libGDX*, o modelo não foi capaz de gerar complementos adequados, gerando ou repetindo palavras desconexas, ou nem gerando sugestões devido à falta de vocabulário relativo ao domínio proposto.

Já no *Google* nota-se que, para todos os domínios, um complemento sintaticamente coerente foi gerado ainda que, como no caso das consultas relativas a *JPA*, *JavaFX* e *libGDX*, o resultado tenha sido uma partícula genérica para completar a frase ("*it*"). Nas demais, os complementos foram pertinentes ao domínio, apesar de não capturarem aspectos mais complexos das consultas originais, resultando em sugestões simples.

É possível perceber que, apesar de a função de erro ter atingido um bom resultado, o modelo LSTM não é muito robusto, tendo dificuldades com partículas gramaticais e domínios desconhecidos. O *Google*, apesar de apresentar sugestões mais generalistas, se mostrou menos propenso a erros de gramática e domínio nas sugestões. Dito isso, existem alguns fatores que poderiam contribuir para um melhor desempenho na geração de complementos de consultas no modelo implementado. A base de dados precisa ser expandida para alimentar melhor o treinamento de modelos, o que permitiria também a construção de arquiteturas mais complexas; O texto presente na base de dados utilizada já passou por algumas etapas de pré-processamento como remoção de *stopwords*, o que faz sentido para algumas tarefas de processamento de linguagem natural, mas que torna o modelo de linguagem deficiente na abstração e geração de frases gramaticalmente corretas.

5 Conclusão

Redes neurais são uma tecnologia poderosa e seu uso vem sendo consolidado para a resolução de muitos problemas nas últimas décadas, devido ao volume de dados que passou a ser produzido no meio digital. Arquiteturas de redes baseadas em células LSTM são amplamente utilizadas para geração de texto e geram resultados que refletem o estado da arte.

É importante lembrar, no entanto, que o um grande volume de dados é necessário para treinar uma rede neural de forma eficiente. No caso do CROKAGE, as redes LSTM não obtiveram bons resultados na geração de autocomplementos de consultas. Algumas possíveis melhorias poderiam ser exploradas para além do escopo deste trabalho: A obtenção de um banco de dados mais extenso, contendo também dados de perguntas e respostas não relacionadas apenas à linguagem Java; o pré-processamento dos dados de forma que *stopwords* e outras palavras auxiliares não fossem descartados; finalmente, a expansão da base de dados permitiria o emprego de outras arquiteturas para a resolução do problema, como redes do tipo *Autoencoder* e *Autodecoder*, além de abordagens baseadas em mecanismos de atenção como *Transformers*.

Referências

ALON, U. et al. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400*, 2018. <https://doi.org/10.48550/arXiv.1808.01400>. Citado na página 17.

BAEZA-YATES, R.; RIBEIRO-NETO, B. et al. *Modern information retrieval*. [S.l.]: ACM press New York, 1999. v. 463. Citado na página 10.

BOLTZMANN, L.; HASENÖHRL, F. Studien über das gleichgewicht der lebendigen kraft zwischen bewegten materiellen punkten. In: . [S.l.: s.n.], 2012. <https://doi.org/10.1017/cbo9781139381420.006>. Citado na página 19.

COMMONS, W. *Artificial Neuron Model*. 2005. Disponível em: <https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png>. Acesso em: 11 nov. 2020. Citado na página 12.

COMMONS, W. *The Elman Network*. 2009. Disponível em: <https://en.wikipedia.org/wiki/Recurrent_neural_network#/media/File:Elman_srnn.png>. Acesso em: 11 nov. 2020. Citado na página 14.

COMMONS, W. *The LSTM Cell*. 2018. Disponível em: <https://commons.wikimedia.org/wiki/File:The_LSTM_cell.png>. Acesso em: 11 nov. 2020. Citado na página 15.

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. <https://doi.org/10.48550/arXiv.1810.04805>. Citado na página 17.

FANG, L.; YU, C.; MENG, W. Personalized web search for improving retrieval effectiveness. *IEEE Transactions on Knowledge and Data Engineering*, v. 16, n. 1, p. 28–40, 2004. <https://doi.org/10.1109/tkde.2004.1264820>. Citado na página 7.

HAN, J.; PEI, J.; KAMBER, M. *Data mining: concepts and techniques*. [S.l.]: Elsevier, 2011. Citado na página 10.

HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1026–1034. <https://doi.org/10.1109/iccv.2015.123>. Citado na página 11.

HE, K. et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778. <https://doi.org/10.1109/cvpr.2016.90>. Citado na página 12.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997. <https://doi.org/10.1162/neco.1997.9.8.1735>. Citado na página 15.

HUANG, G. et al. Densely connected convolutional networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 4700–4708. <https://doi.org/10.1109/cvpr.2017.243>. Citado na página 12.

HUSAIN, H. et al. Codesearchnet challenge: Evaluating the state of semantic code search. *ArXiv*, abs/1909.09436, 2019. <https://doi.org/10.48550/arXiv.1909.09436>. Citado na página 17.

KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2014. <https://doi.org/10.48550/arXiv.1412.6980>. Citado na página 20.

LEMARÉCHAL, C. Cauchy and the gradient method. *Doc Math Extra*, v. 251, p. 254, 2012. <https://doi.org/10.4171/dms/6/27>. Citado na página 13.

LOVINS, J. B. Development of a stemming algorithm. *Mech. Transl. Comput. Linguistics*, v. 11, n. 1-2, p. 22–31, 1968. Citado na página 10.

LUHN, H. P. Key word-in-context index for technical literature (kwic index). *American Documentation*, Wiley Online Library, v. 11, n. 4, p. 288–295, 1960. <https://doi.org/10.1002/asi.5090110403>. Citado na página 10.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943. <https://doi.org/10.7551/mitpress/12274.003.0011>. Citado na página 11.

MERITY, S.; KESKAR, N. S.; SOCHER, R. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017. <https://doi.org/10.48550/arXiv.1708.02182>. Citado na página 17.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. <https://doi.org/10.48550/arXiv.1301.3781>. Citado na página 16.

MIKOLOV, T. et al. Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2013. p. 3111–3119. Citado 2 vezes nas páginas 16 e 17.

MITRA, B.; CRASWELL, N. et al. *An introduction to neural information retrieval*. [S.l.]: Now Foundations and Trends, 2018. <https://doi.org/10.1561/9781680835335>. Citado na página 8.

MNIH, V. et al. Human-level control through deep reinforcement learning. *nature*, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015. <https://doi.org/10.1038/nature14236>. Citado na página 11.

MOHANTY, A. *Multi layer Perceptron (MLP) Models on Real World Banking Data*. 2019. Disponível em: <https://miro.medium.com/max/700/1*-IPQlOd46dlsutIbUq1Zcw.png>. Acesso em: 11 nov. 2020. Citado na página 13.

RADFORD, A. et al. *Improving language understanding by generative pre-training*. 2018. Citado na página 17.

ROBERTSON, S. E.; WALKER, S. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In: SPRINGER. *SIGIR'94*. [S.l.], 1994. p. 232–241. <https://doi.org/10.1007/978-1-4471-2099-5_24>. Citado na página 11.

ROCHA, A. M.; MAIA, M. de A. [Research Data] Mining Relevant Solutions for Programming Tasks from Search Engine Results. Zenodo, abr. 2022. Disponível em: <https://doi.org/10.5281/zenodo.6467629>. Citado na página 21.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. <https://doi.org/10.1037/h0042519>. Citado na página 12.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986. <https://doi.org/10.1038/323533a0>. Citado na página 14.

SILVA, R. F. G. da et al. Recommending comprehensive solutions for programming tasks by mining crowd knowledge. *CoRR*, abs/1903.07662, 2019. <https://doi.org/10.48550/arXiv.1903.07662>. Citado 3 vezes nas páginas 8, 11 e 18.

VAN, H.; KAUCHAK, D.; LEROY, G. Automets: The autocomplete for medical text simplification. *arXiv preprint arXiv:2010.10573*, 2020. <https://doi.org/10.48550/arXiv.2010.10573>. Citado na página 17.

VENKATACHALAM, M. *Recurrent Neural Networks. Remembering what is important.* 2019. Disponível em: <https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce>. Acesso em: 11 nov. 2020. Citado na página 14.

WAN, L. et al. Regularization of neural networks using dropconnect. In: *International conference on machine learning*. [S.l.: s.n.], 2013. p. 1058–1066. Citado na página 17.

WEININGER, N. B. et al. *Presenting autocomplete suggestions.* [S.l.]: Google Patents, 2013. US Patent 8,601,019. Citado na página 7.

Wong, E.; Jinqiu Yang; Lin Tan. Autocomment: Mining question and answer sites for automatic comment generation. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. [S.l.: s.n.], 2013. p. 562–567. <https://doi.org/10.1109/ase.2013.6693113>. Citado na página 17.