UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Felipe Freitas de Carvalho

Transferência de Aprendizado com Embeddings Contextuais para Classificação de Texto em Cenários de Baixo Volume de Dados

Belo Horizonte

2020

Felipe Freitas de Carvalho

Transferência de Aprendizado com Embeddings Contextuais para Classificação de Texto em Cenários de Baixo Volume de Dados

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Cristiano Leite de Castro

Belo Horizonte
2020

Carvalho, Felipe Freitas de.
C331t Transferência de aprendizado com embeddings contextuais para classificação de texto em cenários de baixo volume de dados [recurso eletrônico] / Felipe Freitas de Carvalho. - 2020.
1 recurso online (92 f. : il., color.) : pdf.

Orientador: Cristiano Leite de Castro.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Apêndices: 86-89.

Bibliografia: f. 90-92.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica - Teses. 2. Processamento da linguagem natural (Computação) - Teses. 3. Aprendizado profundo - Teses. I. Castro, Cristiano Leite de. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pela Bibliotecária Roseli Alves de Oliveira CRB/6 2121
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

"Transferencia de Aprendizado com Embeddings Contextuais para Classificação de Texto em Cenários de Baixo Volume de Dados"

Felipe Freitas de Carvalho

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 20 de fevereiro de 2020.

Por:

Prof. Dr. Cristiano Leite de Castro
(UFMG) - Orientador

Prof. Dr. Frederico Gadelha Guimarães
DEE (UFMG)

Prof. Dr. André Paim Lemos
DELT (UFMG)

AGRADECIMENTOS

Dedico este trabalho a todos que estiveram comigo ao longo dessa jornada e que me deram forças e me motivaram.

Agradeço a meus pais, Leonel e Jeanny pelo apoio incondicional e por sempre acreditarem em mim.

Agradeço a meu orientador Cristiano, que me incentivou e apoiou durante a jornada. Além de um grande professor e mentor, um excelente amigo e inspiração.

Agradeço a todas as pessoas com quem trabalhei. Fico muito feliz de ter tido a oportunidade de aprender, com cada um, algo novo diariamente.

E por fim agradeço a Deus, por me dar forças para seguir em frente diante de todas as adversidades e por iluminar meu caminho.

"Above all, don't fear difficult moments. The best comes from them."
(LEVI-MONTALCINISINEK, 2020)

RESUMO

Desenvolvimentos recentes no campo de processamento de linguagem natural demonstraram que redes profundas, baseadas na arquitetura *transformer*, treinadas como um modelo de linguagem em um corpus extenso de texto, de maneira não supervisionada, são capazes de transferir o aprendizado obtido através deste treinamento para tarefas relacionadas através do procedimento de ajuste fino do modelo. Em particular, as arquiteturas BERT e XLNet demonstraram resultados expressivos, atingindo performances que superam o estado da arte em diversas tarefas. Parte do mérito se deve ao fato destes modelos serem capazes de gerar melhores representações de texto, na forma de *embeddings* contextuais. No entanto, não muito foi explorado na literatura a capacidade de transferência de aprendizado destes modelos em cenários de baixa volumetria de dados e também como estes se comportam na tarefa de ajuste fino com diferentes quantidades de dados disponíveis para tal. Além disso, não existem muitos estudos sobre a diferença de ganhos de performance advinda da representação na forma de *embeddings contextuais* em comparação outros tipos de representação de texto, como por exemplo *embeddings* tradicionais, dado um cenário reduzido de dados. Este trabalho endereça estas questões através de uma série de experimentos que buscam, de forma empírica, avaliar o comportamento desses modelos quando sujeitos ao procedimento de ajuste fino em subconjuntos de bases de dados diversas e com volumetria progressivamente menor. Também são avaliados os ganhos de performance advindos desta nova forma de representação de dados, utilizando os modelos como extratores de características na tarefa de transferência de aprendizado. Mediante os resultados dos experimentos é possível observar que os modelos BERT e XLNet são capazes de terem performances boas, com baixa volumetria de dados disponíveis, na maioria dos casos. Além disso, é possível ver que embora exista um efeito positivo em utilizar um volume maior de dados, um volume pequeno já se mostra suficiente para obter uma performance melhor ou comparável a outros tipos de modelos treinados com mais dados. Também é possível observar que parte do poder destes métodos de fato deriva de representações mais robustas, dado que estas geram resultados melhores que *embeddings* tradicionais ao serem usadas como entrada para outros modelos, na maior parte dos casos. No entanto, é possível perceber que a arquitetura *transformer* como um todo, após o procedimento de ajuste fino, tem uma performance substancialmente melhor em baixas volumetrias de dados quando comparada a utilização destes modelos com extratores de características.

Palavras-chave: baixo volume de dados; classificação de texto; processamento de linguagem natural; representações contextuais; aprendizado de representação; aprendizado profundo.

ABSTRACT

Recent developments in the NLP (Natural Language Processing) field have shown that deep transformer based language model architectures trained on a large corpus of unlabeled data are able to transfer knowledge to downstream tasks efficiently through fine-tuning. In particular, BERT and XLNet have shown impressive results, achieving state of the art performance in many tasks through this process. This is partially due to the ability these models have to create better representations of text in the form of contextual embeddings. However not much has been explored in the literature about the robustness of the transfer learning process of these models on a small data scenario. Also not a lot of effort has been put on analyzing the behavior of the two models fine-tuning process with different amounts of training data available. Besides that, that are no studies about the difference, in terms of performance, that come from the contextual embedding representation versus traditional embedding representations, in a small data scenario. This paper addresses these questions through an empirical evaluation of these models on some datasets when fine-tuned on progressively smaller fractions of training data, for the task of text classification. The performance gains from the new way of text representation are also evaluated by using the models as feature extractors for transfer learning. It is shown that BERT and XLNet perform well with small data and can achieve good performance with very few labels available, in most cases. Results yielded with varying fractions of training data indicate that few examples are necessary in order to fine-tune the models and, although there is a positive effect in training with more labeled data, using only a subset of data is already enough to achieve a comparable performance in comparison to other models trained with substantially more data. It is also possible to observe that part of these models power is in fact due to more robust representations, given they yield better results than traditional embedding representations when used as features for other models, in most cases. However, it is noticeable how the transformer architecture as a whole is able to, after the fine tuning process, yield substatially better results in comparison to using the model as a feature extractor.

Keywords: small data, text classification; natural languange processing; contextual embeddings; deep learning;.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AR	*Auto-regressive*
CNN	*Convolutional Neural Networks*
GPU	*Graphics Processing Unit*
HSD	*Honest Significant Difference*
IDF	*Inverse Document Frequency*
LM	*Language Model*
LSTM	*Long Short-Term Memory network*
NER	*Named-Entity Recognition*
NLI	*Natural Language Inference*
NLP	*Natural Language Processing*
NLU	*Natural Language Understanding*
QA	*Question Awnsering*
RNN	*Recurrent Neural Network*
SVM	*Support Vector Machines*
TF	*Term Frequency*
TPU	*Tensor Processing Unit*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos houve um desenvolvimento de modelos baseados em arquiteturas de redes neurais profundas (*deep learning*) para realização de tarefas complexas de classificação como: detecção de objetos, reconhecimento de voz, classificação de documentos, filtragem de conteúdo de redes sociais, dentre outras. Os ganhos advindos destes modelos derivam-se principalmente de sua capacidade de gerar representações poderosas a partir dos dados, que na maioria dos casos são não estruturados (GOODFELLOW; BENGIO; COURVILLE, 2016).

Existe um viés na literatura sobre o uso de grandes volumetrias de dados como condição necessária para extração de boas performances de modelos de *deep learning*. Esta premissa é razoável, dado que o treinamento de modelos complexos e com uma quantidade grande de parâmetros, em bases de dados pequenas, é propenso a gerar o sobreajuste aos dados (HASTIE; TIBSHIRANI; FRIEDMAN, 2001). No entanto, o paradigma da utilização de modelos profundos em conjuntos de dados menores mudou com o surgimento e evolução das técnicas de transferência de aprendizado (TORREY; SHAVLIK, 2010). Esta, por sua vez, mostrou a possibilidade de se treinar modelos profundos em um grande volume de dados e então transferir parte do aprendizado obtido para resolução de tarefas relacionadas, com menor volume de dados que o necessário para treinar o modelo do zero, e utilizando parte do que foi aprendido sobre a representação dos dados proveniente do treinamento anterior.

Neste trabalho, modelos de *deep learning* com alta capacidade e muitos parâmetros, baseados na arquitetura *transformer* (VASWANI et al., 2017), serão utilizadas na tarefa de classificação de texto utilizando o paradigma de transferência de aprendizado, em um cenário de dados bastante reduzido.

A prática da transferência de aprendizado pode ocorrer de duas formas distintas. Na primeira o modelo e a representação utilizada são ajustados durante o treinamento para a tarefa subsequente. Na segunda, a representação obtida dos dados através do treinamento anterior é fixa e apenas o modelo é ajustado para a tarefa subsequente. A primeira prática ganha o nome de ajuste fino, enquanto que na segunda é dito que o modelo esta sendo usado como um extrator de características. Ambos os procedimentos são diferentes de como é realizado a tarefa de classificação de texto durante um processo de modelagem tradicional. O método clássico envolve o treinamento do modelo do zero e as representações utilizadas são baseadas em técnicas baseadas em contagem de palavras, e que não envolvem o treinamento de modelos para derivação de *embeddings*, como quando se realiza a transferência de aprendizado. Por isso, a prática da transferência de aprendizado representa uma quebra de paradigma em relação ao aprendizado tradicional

(GOODFELLOW; BENGIO; COURVILLE, 2016).

A escolha de tarefas de classificação de texto para este trabalho se deu pois esta representa um arcabouço genérico na resolução de uma grande quantidade de problemas pertencentes ao universo de processamento de linguagem natural (NLP- *natural language processing*). Tarefas diversas como analise de sentimentos, detecção de *spam*, detecção de intenção, categorização de tópicos, dentre outras, se enquadram como tarefas de classificação de texto.

1.1 Motivação

Avanços recentes no campo de processamento de linguagem natural demostraram que redes profundas, baseadas na arquitetura *transformer* (VASWANI et al., 2017), treinadas como um modelo de linguagem em um corpus extenso de texto de maneira não supervisionada, são capazes de realizar uma transferência de aprendizado eficiente para tarefas relacionadas subsequentes (DEVLIN et al., 2018; YANG et al., 2019; DAI et al., 2019; LIU et al., 2019; RADFORD et al., 2018). Estes modelos atingiram resultados estado da arte em diversas tarefas de NLP através do processo de pré-treino, (que é o treinamento do modelo de maneira não supervisionada em um corpus extenso de texto) seguido do processo de ajuste fino do modelo para uma tarefa específica. Além disso, estes modelos têm em comum o fato de possuir grande capacidade, contando com milhões de parâmetros, além de usarem o mecanismo de atenção (LUONG; PHAM; MANNING, 2015). Estes, por sua vez, criam um novo tipo de representação de texto, que é mais profunda e consegue carregar mais informação que as representações predecessoras, como o word2vec (MIKOLOV et al., 2013) e GloVe (PENNINGTON; SOCHER; MANNING, 2014), que são os *embeddings* contextuais (*contextual embeddings*). Esse novo tipo de *embedding* é capaz de capturar o contexto de uma sentença e usá-lo de maneira que a representação da palavra desejada encapsula características do contexto em que está inserida em uma frase ou documento. Isso, por sua vez, permite que aspectos importantes da linguagem sejam embutidos na representação das palavras, tais como polissemia, sarcasmo, negação e qualquer característica que seja dependente de contexto.

No entanto, não é claro o quão robustos estes modelos são em um cenário onde se deseja transferir o aprendizado obtido do treinamento prévio para conjuntos de dados muito pequenos, principalmente mediante o processo de ajuste fino. Isso se deve ao fato de que realizar este procedimento em conjuntos de dados muito reduzidos é significativamente mais desafiador, dado que o risco de sobre ajuste do modelo aos dados (*overfitting*) aumenta devido à grande capacidade do modelo e o baixo volume de dados disponíveis para treinamento (Karpathy, Andrej, 2019).

Em Devlin et al. (DEVLIN et al., 2018) é discutido como o modelo BERT (um dos mais bem sucedidos modelos de linguagem baseados na arquitetura *transformer*) tem uma probabilidade maior de gerar resultados degenerados após o ajuste fino quando o conjunto de dados é pequeno. Isso é esperado de um modelo com 340M de parâmetros que é ajustado em uma volumetria reduzida de dados (Karpathy, Andrej, 2019). A capacidade destes modelos de realizarem a tarefa de transferência de aprendizado neste cenário é especialmente importante dado seu valor prático, visto que em muitos cenários reais é comum se ter uma baixa volumetria de dados rotulados para treinamento de modelos.

Algo que também não é muito explorado na literatura é como modelos como o

BERT se comportam com diferentes quantidades de dados disponíveis de uma mesma base de dados para realização do procedimento de ajuste fino, ou seja, sua curva de transferência de aprendizado neste cenário. Além disso, não é quantificada a diferença na utilização destes modelos como extratores de características *versus* quando usados com o ajuste fino do modelo, em um cenário de poucos dados. Não obstante, pensando na utilização dos modelos como extratores de características, é interessante quantificar o ganho dos *embeddings* contextuais em relação a formas mais estabelecidas de *embeddings*, como word2vec (MIKOLOV et al., 2013) e GloVe (PENNINGTON; SOCHER; MANNING, 2014), de forma a mensurar possíveis ganhos em termos de poder de representação, dado o cenário de dados reduzidos, derivados da característica contextual dos *embeddings* mais recentes. Por último, também não é explorada a utilização de funções de custo distintas durante o processo de ajuste fino, dado que este é um processo de otimização complexo, principalmente considerando o cenário reduzido de dados que é explorado neste trabalho.

1.2 Objetivos

Este trabalho tem como objetivo responder as questões levantadas na seção anterior, que estão em aberto, considerando que o comportamento destes modelos em cenários de baixa volumetria de dados é muito pouco explorado na literatura. Isso é feito através de análises empíricas em dois dos mais bem sucedidos modelos de linguagem baseados na arquitetura *transformer*, que atingiram resultados estado da arte em diversas tarefas no momento em que foram propostos, BERT (DEVLIN et al., 2018) e XLNet (YANG et al., 2019).

Para isso, os dois modelos, BERT e XLNet, são primeiramente testados a fim de verificar a robustez do seu processo de transferência de aprendizado via ajuste fino através da avaliação de sua performance em tarefas de classificação de texto usando conjuntos de dados com volumes progressivamente menores de dados.

Em uma segunda etapa, os mesmos modelos são comparados em um cenário único de baixa volumetria de dados, onde é contrastado a prática de transferência de aprendizado com ajuste fino *versus* o uso desses modelos como extratores de características. Essa análise, por sua vez, visa observar os benefícios relativos do processo de ajuste fino do modelo em relação ao poder representacional puro obtido através do treinamento prévio como modelo de linguagem destes modelos.

Em uma outra análise, as representações provenientes destes modelos na forma de *embeddings* contextuais são comparadas com as representações de *embeddings* tradicionais, mais especificamente com a representação GloVe (PENNINGTON; SOCHER; MANNING, 2014). O objetivo desta análise é quantificar a importância e vantagens de uma representação mais profunda, que encapsula contexto, quando utilizada como vetor de características de entrada para um modelo em comparação com uma representação de *embedding* tradicional.

Por fim, uma função de custo diferente da usualmente usada no processo de ajuste fino do modelo, função de perda de articulação (*hinge loss*), usada para classificação de margem máxima e empregada usualmente no classificador SVM (*support vector machines*), é testada como função de custo para a tarefa de ajuste fino do modelo em um cenário de baixa volumetria de dados. Este último teste tem como objetivo observar se o princípio de máxima margem traz benefícios em termos de acurácia do modelo garantindo uma melhor separação entre classes e capacidade de generalização, dado o cenário de volumetria reduzida de dados.

1.3 Trabalhos relacionados

Antes do advento de modelos como o BERT e XLNet, tentativas de realizar a tarefa de transferência de aprendizado via ajuste fino do modelo em conjunto de dados relativamente pequenos não foram bem sucedidas (MOU et al., 2016; DAI; LE, 2015). Isso é esperado pois, mesmo em um cenário de transferência de aprendizado, otimizar modelos com alta capacidade usando volumetria de dados reduzida é uma tarefa intrinsecamente desafiadora (HASTIE; TIBSHIRANI; FRIEDMAN, 2001; Karpathy, Andrej, 2019). No entanto, autores como Howard et al. (HOWARD; RUDER, 2018) e Cer et al. (CER et al., 2018) apresentaram posteriormente, cada um utilizando arquiteturas de modelo distintas, maneiras de realizar a tarefa de ajuste fino em conjuntos de dados pequenos com um grau razoável de sucesso.

Em Howard et al (HOWARD; RUDER, 2018), é apresentado o modelo ULMFit, que também é um modelo de linguagem pré treinado em um corpus extenso de texto, porém baseado em uma arquitetura bi-LSTM, otimizada com uma heurística customizada para a taxa de aprendizado. Em seu trabalho, Howard et al mostra que, após o ajuste fino, é possível obter performance superior ao cenário onde o modelo é treinado do zero, para tarefas específicas, utilizando apenas o conjunto de dados da tarefa subsequente (HOWARD; RUDER, 2018). O autor demonstra isto através da utilização de algumas bases de dados com diferentes frações de dados disponíveis para treinamento. Embora esta comparação seja importante e capaz de mostrar a habilidade de transferência de aprendizado do modelo, uma melhor performance neste cenário em comparação ao treinamento do modelo do zero, apenas com os dados da tarefa a ser resolvida, é de certa forma esperada, visto que é bem conhecido na literatura que modelos com alta capacidade quando treinados em baixas volumetrias de dados tendem a se sobre-ajustarem aos dados (HASTIE; TIBSHIRANI; FRIEDMAN, 2001). Outras diferenças entre o estudo de Howard et al e as análises propostas neste trabalho são que a arquitetura bi-LSTM usada pelo autor é drasticamente diferente da arquitetura analisada neste trabalho, baseadas no modelo de *transformer* e, além disso, tem ordens de magnitude diferentes no que tange o número de parâmetros do modelo. Olhando os resultados obtidos em Horward et al, expostos na Figura 1, é possível perceber que a curva de transferência de aprendizado de seu modelo não é muito ingrime, e demora para saturar, o que indica menor robustez e capacidade de generalização, algo que também é evidenciado dado sua performance inferior ao ser comparado com BERT e XLNet, na maioria dos casos.

Em Cer et al (CER et al., 2018), o modelo USE é apresentado. Neste trabalho é mostrado que este apresenta uma característica de transferência de aprendizado boa e, neste caso o mesmo é comparado com outros modelos pre-treinados, que utilizam como entrada *embeddings* word2vec (MIKOLOV et al., 2013). As curvas de transferência de aprendizado apresentadas são ingrimes porém os resultados expostos não são tão bons

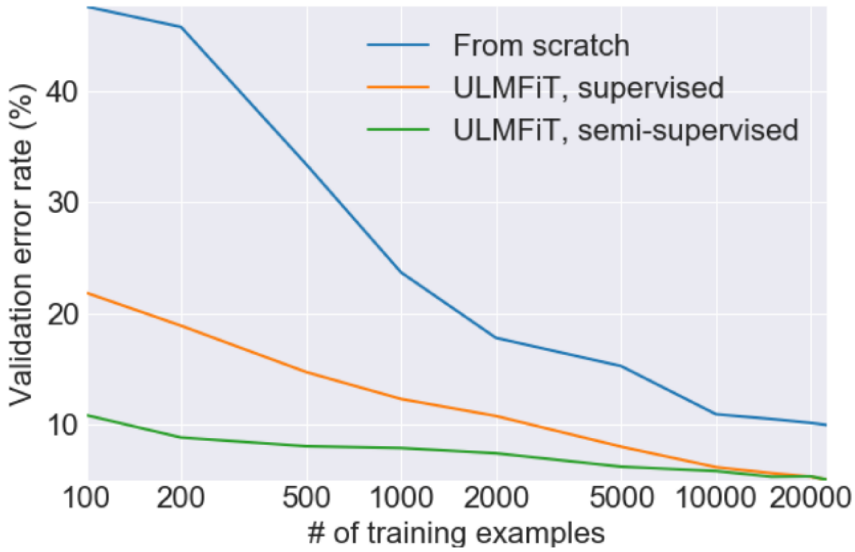


Figura 1 – Curva de transferência de aprendizado do modelo ULMFit para o conjunto de dados IMDb (HOWARD; RUDER, 2018)

como os apresentados nos modelos BERT e XLNet e requerem quantidade maior de dados para serem atingidos.

Independentemente dos resultados obtidos por estes trabalhos anteriores, estes motivaram os estudos empíricos conduzidos neste trabalho a fim de observar qual é o comportamento dos modelos BERT e XLNet em um cenário de volumetria reduzida de dados. Essa comparação também visa observar a capacidade de transferência de aprendizado desses métodos, tomando como inspiração os trabalhos previamente mencionados. A diferença fundamental do proposto neste trabalho é que os dois modelos estudados, ao conhecimento dos autores deste trabalho, não tiveram nenhum estudo feito em relação a sua performance e capacidade de transferência em cenários com poucos dados. Outra diferença é que tanto o BERT (DEVLIN et al., 2018) e XLNet (YANG et al., 2019) tem arquiteturas diferentes de seus predecessores e, além disso, tem capacidade superior, contando com um número de parâmetros que ultrapassa em muito os demais modelos. Isso contribui para que o processo de ajuste fino desses modelos seja anda mais desafiador e, como agravante, neste trabalho a quantidade de dados disponíveis para esse procedimento é ainda menor que o usado nos trabalhos anteriores mencionados (HOWARD; RUDER, 2018; CER et al., 2018).

Sendo modelos estado da arte, as performances dos modelos BERT e XLNet são bem estabelecidas e obviamente superiores à performance de demais modelos, no entanto suas habilidade de transferência de aprendizado e operação em um cenário de dados reduzido é ainda desconhecida. Isso é interessante pois, discutivelmente, este talvez seja um dos maiores benefícios de sua utilização, dado que estes contribuem principalmente por criarem novas representações de texto que são mais robustas e contextuais (DEVLIN

et al., 2018).

1.4 Organização do trabalho

O trabalho se encontra dividido da seguinte maneira: o Capítulo 2 apresenta uma fundamentação teórica dos principais conceitos e técnicas que fundamentam os experimentos apresentados neste trabalho. No Capítulo 3 as bases de dados utilizadas e o *setup* experimental é apresentado e no Capítulo 4 os resultados dos experimentos são apresentados e discutidos. Por fim, no Capítulo 5 a conclusão deste trabalho e de seus experimentos é apresentada.

2 Fundamentação Teórica

Neste capítulo serão discutidos os principais conceitos, técnicas e modelos que serão usados neste trabalho. Estes serão expostos em uma ordem que vai das técnicas mais clássicas de processamento de linguagem natural até os mais recentes modelos, foco deste trabalho, BERT e XLNet.

2.1 Modelos de representação de texto baseados em contagem de palavras

Dados não estruturados precisam ser representados de alguma maneira a fim de serem utilizados em um problema de aprendizado de máquina. Técnicas clássicas para representação de dados textuais utilizam técnicas baseadas na contagem de palavras. Dentre as mais utilizadas encontram-se o modelo *bag of words (BOW)* e o modelo TF-IDF.

2.1.1 Bag of words

A técnica mais simples de representação do texto é através do modelo de *bag of words* (ZHANG; JIN; ZHOU, 2010). Nesta técnica, cada palavra de um documento é representada como uma característica cujo valor é igual à sua frequência no documento, ou seja, quantas vezes ela aparece. Em NLP o termo documento tem um significado diferente dependendo do caso de uso, podendo significar uma frase, um parágrafo, uma página ou até mesmo um livro mas, no geral, um documento gera um exemplo de treinamento para o modelo, embora sua definição varie de acordo com a tarefa desejada. A figura 2 exemplifica de maneira bem clara o funcionamento do técnica *bag of words*. O resultado é um vetor cujo tamanho V é igual ao tamanho do vocabulário do corpus de documentos e onde cada coluna é uma palavra, cujos valores são as contagens dessas palavras no documento. Este vetor é bastante esparso e tem alta dimensão, dado que, na maioria das vezes, o vocabulário de um corpus de texto é extenso.

Ao se utilizar o método *bag of words* para representação de texto perde-se a noção da ordem das palavras no documento. Isso é bastante crítico, dado que a ordem das palavras pode ser algo muito importante em determinadas aplicações. A representação de texto gerada pelo método *bag of words* é bastante rudimentar porém esta é uma técnica clássica que funciona, na maioria dos casos, como um *baseline* para tarefas de NLP e, surpreendentemente, pode gerar resultados muito bons em várias aplicações, independente de sua simplicidade. Uma técnica muito similar, porém que consegue capturar um pouco mais de informação e gera uma representação do texto ligeiramente mais robusta, é o modelo TF-IDF.

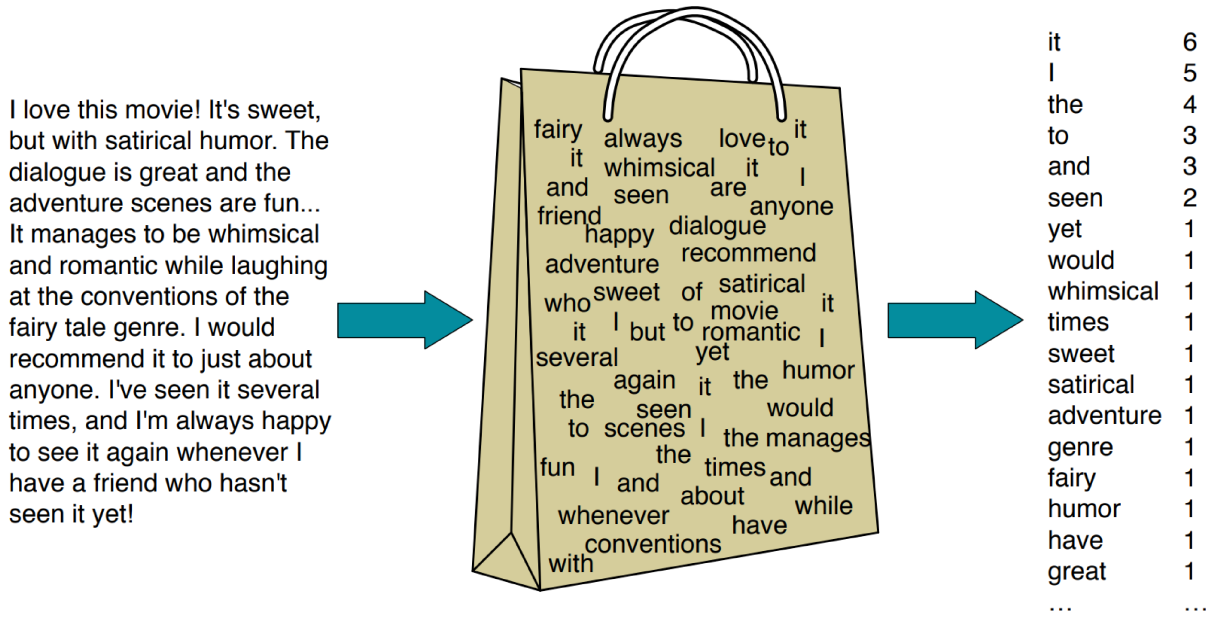


Figura 2 – O modelo *bag of words* (JURAFSKY; MARTIN, 2019)

2.1.2 TF-IDF

O método TF-IDF (ROBERTSON, 2004) representa as palavras realizando dois tipos de contagem. A primeira é através do TF ou *term frequency*. Este procedimento é idêntico ao feito no modelo *bag of words*, onde cada palavra ou termo é representado como uma característica cujo valor é igual à sua frequência no documento, ou seja, quantas vezes ele aparece. Em uma segunda etapa, é feita a contagem de quantos documentos cada palavra aparece em relação ao número de documentos presentes no corpus de texto analisado. O inverso dessa razão, dado por N/df_t, onde N é o número de documentos total no corpus e df_t é o número de documentos onde o termo t aparece, dá origem ao termo IDF, ou *inverse document frequency*. Unindo estas duas contagens, o método TF-IDF atribui a cada termo (palavra) um valor que é dado pela multiplicação do número de vezes em que ele aparece em um documento (TF) ponderado pela razão de em quantos documentos este elemento aparece dentro do corpus(IDF). Isso implica que palavras que raramente aparecem em documentos são consideradas mais importantes pois tem um IDF alto, enquanto palavras comuns a todos os documentos de um corpus são consideradas menos importantes por serem muito comuns e potencialmente não trazerem informações que discriminam tanto o documento analisado de um outro qualquer.

Este método é ligeiramente mais robusto que o método *bag of words* pois incorpora o termo IDF que trás informação adicional sobre a importância relativa da palavra no corpus de documentos. Assim como no método *bag of words*, o método TF-IDF também gera vetores igualmente esparsos e de alta dimensionalidade, cujo tamanho V é igual ao tamanho do vocabulário. Não obstante, a noção de ordem das palavras em um documento também não consegue ser capturada pelo método TF-IDF.

2.2 Tokenização de texto

Um primeiro passo no pré-processamento de um texto é a separação de cada termo (ou *token*) nesse texto, um procedimento chamado de tokenização (JURAFSKY; MARTIN, 2019). Este é usualmente usado para separar cada palavra do texto, embora possa também separar o texto em outras granularidades. Não existe maneira padrão de realizar este procedimento, embora seja comum a separação de palavras por espaços em branco.

No processo de tokenização, uma mesma palavra pode gerar mais de um *token*, ou seja, ter mais de uma representação, caso ela esteja no início de uma frase com letra maiúscula e depois no meio de uma frase com letra minúscula, por exemplo. Logo um procedimento padrão é remoção de acentos, pontuação e transformação do texto para letra minúscula. Esse procedimento pode ajudar a reduzir o tamanho do vocabulário gerado. No entanto, existe sempre o risco de perda de informação com pré-processamentos aplicados, portanto qualquer tipo de pré-processamento deve ser executado de forma cautelosa.

É possível separar *tokens* a nível sub-palavra, onde cada *token* seria um conjunto de caracteres. Esse procedimento pode ser vantajoso em determinados cenários, onde o texto analisado conta com muitos erros ortográficos, muito comum em textos provenientes de redes sociais, por exemplo.

Tokens podem ter tamanhos variados dependendo da técnica. Alguns modelos contam com *tokenizers* próprios, que dividem as palavras de acordo com heurísticas predeterminadas, como é o caso do BERT (DEVLIN et al., 2018) e XLNet (YANG et al., 2019), que serão discutidos posteriormente neste trabalho.

2.3 N-gramas

Um N-grama, ou em inglês *N-gram*, é uma sequência de N elementos, que podem ser caracteres, palavras, sílabas, fonemas, etc. Um bi-grama é uma sequência de 2 elementos e um tri-grama, de 3.

N-gramas são uma forma alternativa de representação de *tokens* em um texto. Estes podem funcionar em conjunção com os modelos *bag of words* ou TF-IDF, onde grupos de palavras representariam cada termo, ou mesmo grupos de caracteres dariam origem a cada característica do modelo. Esta ultima prática é mais popular pois grupos de palavras podem ser muito infrequentes e, portanto, causarem problemas como o de não aparecerem simultaneamente no conjunto de treino e teste. Enquanto isso, grupo de caracteres conseguem capturar partes iguais de palavras que apresentam sufixos ou prefixos específicos, além de serem adequados para linguagens com morfologia rica (JURAFSKY; MARTIN, 2019).

A pratica de *tokenização* de texto a nível caractere e representação dos n-gramas resultantes através de um modelo BOW ou TF-IDF é bastante comum para diversas tarefas de processamento de linguagem natural. Além disso, modelos de n-gramas, especialmente o de palavras, podem também ser utilizados como modelo de linguagem, ou *language models* (LMs) em inglês.

2.4 Modelos de linguagem

Modelos de linguagem são modelos que atribuem probabilidade para uma sequência de palavras, ou *tokens* (JURAFSKY; MARTIN, 2019). Os modelos de linguagem são capazes, e muitas vezes tem o objetivo, de prever a próxima palavra em uma sequência de palavras.

Dado uma sequencia de palavras $w_1, w_2, ...w_n$, podemos computar a probabilidade da sequência, através da aplicação da regra da cadeia da probabilidade, como:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \tag{2.1}$$

Assumindo uma premissa Markoviana de que a palavra w_i pode ser aproximada apenas pelas n-1 palavras que a precedem ao invés do histórico completo de i-1 palavras temos que:

$$\prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) \tag{2.2}$$

O primeiro termo da equação 2.2 pode ser calculado através do modelo *N-gram*. A probabilidade condicional P(w_5 — ($w_4 \ldots w_1$)) pode ser visto como um N grama com n=5. Através da contagem do numero de vezes que um n-grama especifico de tamanho N aparece, condicionado à frequência de que os N-1-gramas aparecem, podemos estimar as probabilidades condicionais associadas e assim derivar o modelo de linguagem através da Equação 2.3. Esta mostra como um modelo de N-grama é utilizado para derivar um modelo de linguagem.

$$P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})} \tag{2.3}$$

Os modelos de linguagem são de grande importância para a área de NLP. Além do modelo de N-grama, existem outras formas de se derivar um modelo de linguagem. Para isto, é somente necessário um modelo que consegue estimar a distribuição de probabilidade de um sequência, que é algo que pode ser realizado de diversas maneiras. Como será visto nas seções subsequentes, estes modelos estão sendo usados como tarefa base (pré-treino)

(DEVLIN et al., 2018; YANG et al., 2019; CER et al., 2018; HOWARD; RUDER, 2018) para posterior aplicação de técnicas de transferência de aprendizado, como acontece no caso dos modelos avaliados neste trabalho, BERT (DEVLIN et al., 2018) e XLNet (YANG et al., 2019). Estes modelos cumprem, muitas vezes, a função de realizar um entendimento da linguagem, tarefa conhecida como *natural language understanding* (NLU) (DEVLIN et al., 2018).

2.5 SVM para classificação de texto e função de perda de articulação

As representações de texto descritas na seções anteriores, *BOW (bag of words)* e TF-IDF, em conjunção com a utilização de n-gramas de tamanhos variados, e muitas vezes a um nível de caracteres, são maneiras clássicas de representação de texto utilizadas para diversas tarefas de NLP e constituem um *baseline* robusto.

Como discutido previamente, um dos problemas deste tipo de representação é a esparsidade dos vetores resultantes. Dado este cenário, para resolver problemas de classificação usando como entrada BOW ou TF-IDF são normalmente usados classificadores lineares através de técnicas como regressão logística e SVM (JURAFSKY; MARTIN, 2019).

Em particular, a representação de texto através de TF-IDF em conjunção com a utilização do SVM linear como classificador é um *baseline* ótimo e seus benefícios são discutidos por Joachims et al (JOACHIMS, 1998). Dentre os motivos que corroboram a utilização do SVM como classificador para este tipo particular de representação se destaca o fato de que SVMs tem uma propriedade de aprender independente da alta dimensionalidade do espaço de características do problema (JOACHIMS, 1998). Isto se deve ao fato de que sua formulação de aprendizado mede a complexidade da hipótese baseado na margem de separação e não no número de características, usando o principio da máxima margem de separação. Além disso, também é discutido como problemas de classificação de texto que usam a representação TF-IDF tendem a ser linearmente separáveis, o que justifica a utilização do SVM linear sem *kernel trick*.

O princípio da máxima margem de separação de SVMs, por sua vez, é obtido através de uma função de custo chamada função de perda de articulação (*hinge loss*). A Equação 2.4 descreve a função de custo de perda de articulação, onde $\vec{w} \cdot \vec{x}_i - b$ representa a saída do modelo, para cada entrada x_i e y_i é o rotulo i. O resultado da aplicação dessa função de custo pode ser visto através da Figura 3. Se o ponto x_i está no lado certo da margem de separação $\vec{w} \cdot \vec{x}_i - b$ seu custo é 0 para o problema de otimização, caso contrario, é proporcional à distância a essa margem. O parâmetro λ determina o *trade-off* entre aumentar o tamanho da margem de separação e garantir que x_i esteja sempre do lado correto da margem.

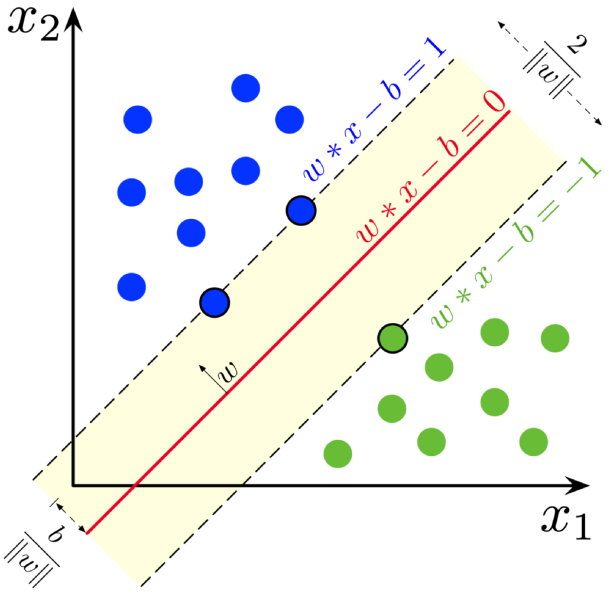


Figura 3 – SVMs e o princípio da máxima margem (Wikipedia contributors, 2004)

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(w \cdot x_i - b)\right)\right] + \lambda\|w\|^2 \tag{2.4}$$

Esta função de custo para o SVM é equivalente à utilização da função de perda logarítmica para a regressão logística, por exemplo, e pode ser considerada um caso especial de regularização Tikhonov (POGGIO; TORRE, 1984). Uma propriedade importante desta função é tentar ao mesmo tempo minimizar o risco empírico e maximizar uma margem geométrica (VAPNIK, 1999) como pode ser visto na figura 3. Por esta razão, também é uma premissa razoável a utilização desta função em um cenário de escassez de dados, devido ao seu potencial poder de garantir uma maior generalização devido à aplicação do princípio de máxima margem. Isso será explorado no processo de ajuste fino dos modelos BERT e XLNet, como experimento, a fim de observar, de maneira empírica, se existe algum efeito positivo na utilização desta função de custo durante este processo, em comparação com a utilização da função de custo padrão logarítmica durante o ajuste fino.

2.6 Aprendizado da representação

Os métodos de representação de texto baseados em modelos de contagens de palavras geram representações com alta dimensionalidade e bastante esparsas. No geral, algoritmos de aprendizado de máquina sofrem com altas dimensões devido a um efeito conhecido como maldição de dimensionalidade (*curse of dimensionality*) (GOODFELLOW; BENGIO; COURVILLE, 2016). Este efeito acontece pois, em altas dimensões, o número de combinações distintas do conjunto de variáveis cresce de forma exponencial devido ao aumento do número de características. Com isso um desafio estatístico surge pois o número

de combinações possíveis de características em altas dimensões é usualmente muitas vezes maior que o número de exemplos de treino e, desta forma, os modelos muitas vezes perdem poder de generalização, comprometendo sua capacidade de inferência em cima de um conjunto de dados que nunca foi antes visto (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figura 4 – Maldição de dimensionalidade, conforme o número de dimensões cresce, são necessário mais exemplos de treinamento para mapear o espaço (GOODFELLOW; BENGIO; COURVILLE, 2016)

A fim de se distinguir O(k) regiões no espaço, modelos baseados em aprendizado estatístico, principalmente os baseados em *kernels*, requerem O(k) exemplos de treino (GOODFELLOW; BENGIO; COURVILLE, 2016; HASTIE; TIBSHIRANI; FRIEDMAN, 2001), o que usualmente não acontece para casos em que se tem altas dimensões. Estes modelos usualmente assumem um conjunto de premissas sobre a função que eles desejam aprender, que muitas vezes não é suficiente para atingir uma boa generalização (GOODFELLOW; BENGIO; COURVILLE, 2016). Dentre estas premissas destaca-se a de que a função a ser descoberta é localmente suave e que pequenos deltas na entrada não impactam severamente a saída. Isso reforça a grande utilização de técnicas baseadas em *kernels* ou *template matching* como formas clássicas de lidar com problemas de classificação e regressão (HASTIE; TIBSHIRANI; FRIEDMAN, 2001).

Com o advento de técnicas baseadas em *deep learning*, os problemas anteriores foram atacados. A principal maneira para tal foi através de modelos que aprendessem representações dos dados mais robustas a partir de seu treinamento, extraindo características de maneira automática (GOODFELLOW; BENGIO; COURVILLE, 2016). Tal capacidade é conhecida como aprendizado da representação (*representation learning*). Para se gerar esse tipo de representação, primeiramente, são assumidas premissas sobre a distribuição dos dados de tal maneira que estes são representados na forma de uma composição de fatores, de forma a gerar uma hierarquia onde representações mais simples geram representações cada vez mais complexas, conforme se cresce em profundidade nas redes utilizadas pelos métodos profundos. Assim, um número de regiões $O(2^k)$ pode ser

definido com O(k) exemplos em situações em que estas premissas são verdadeiras, como em casos onde há simetrias, por exemplo.

Além disso, os métodos baseados em *deep learning*, da forma como são estruturados, e de suas arquiteturas, aproveitam de um tipo de aprendizado baseado em *manifold learning* (GOODFELLOW; BENGIO; COURVILLE, 2016). Métodos baseados em *Manifold Learning* partem do principio de que, dentre todas as regiões de $\mathbb{R}^n$, onde n é o número de características, grande parte delas contém pouca informação relevante e que regiões interessantes ocorrem em superfícies (ou *manifolds*) que contém conjuntos de dados específicos. Nestas superfícies, variações importantes ocorrem em algumas direções específicas. Isso quer dizer que algoritmos de *deep learning*, através de suas técnicas de modelagem de dados, são capazes de aprender *manifolds* onde a distribuição de probabilidade dos dados está altamente concentrada, descartando em grande parte regiões implausíveis de combinações das variáveis. É fácil ver a teoria de *manifold learning* aplicada em dados não estruturados, basta notar que a geração aleatória de palavras em um texto ou de pixeis em imagens, por exemplo, geram, na maioria dos casos, nada humanamente compreensível. Textos que fazem sentido, ou imagens validas, estão distribuídos com probabilidade concentrada em regiões específicas do espaço.

A importância de representações de texto mais robustas, que não sofram tanto de fenômenos como alta dimensionalidade, esparsidade e, consequentemente, de maldição de dimensionalidade, aliados com os desenvolvimentos advindos da pesquisa de técnicas de *deep learning*, culminaram no surgimento de novas formas de representação de texto que visavam resolver os problemas anteriores, através dos *embeddings*.

2.7 Embeddings

Embeddings são uma forma de representação que consegue garantir significado semântico ao texto e o representa através de vetores densos de tamanho fixo, com dimensão menor quando comparado aos vetores gerados com TF-IDF ou *bag of words*, por exemplo. Esta técnica surgiu primeiramente com o algoritmo word2vec (MIKOLOV et al., 2013) que gera os *embeddings* através do treinamento de um modelo neural de forma não supervisionada em um corpus extenso de texto. *Embeddings* capturam o significado das palavras através do contexto em que estas estão inseridas e tangibilizam, no universo de dados de texto, o conceito de aprendizado da representação, advindo com o avanço da teoria de *deep learning*.

A ideia de que palavras que acontecem em contextos similares tem significados similares é conhecida como hipótese distribucional (*distributional hypothesis*) e foi formulada na década de 1950 por linguistas (JURAFSKY; MARTIN, 2019). Partindo desta ideia, baseado na distribuição das palavras em um texto, modelos de representação semânticos

baseados em vetores densos surgiram na forma de *embeddings*. Representar uma palavra como um ponto em um espaço multidimensional de forma que palavras similares estejam próximas caracterizam o que se chama de *semantic vectors*, ou vetores semânticos, que são os *embeddings*, que ganham este nome pois eles embutem (*embedd*) palavras neste espaço. Cada dimensão, ou componente, de um vetor de *embeddings* representa uma característica sobre esta palavra, como por exemplo o quão pertencente ao gênero feminino uma palavra é por exemplo, embora isso não seja explicitamente visível (MIKOLOV et al., 2013). A Figura 5 ilustra este comportamento.

As técnicas baseadas em *embeddings* atacam vários problemas e limitações que as técnicas clássicas de representação de texto tinham. Dentre os problemas, os *embeddings* eliminam a esparsidade da representação gerada, através da utilização de vetores densos e, além disso, conseguem extrair significado semântico das palavras. *Embeddings* são usualmente gerados de maneira não supervisionada, através da utilização de um corpus extenso de texto, o que faz com que estes tenham um poder de generalização razoável advindo da grande quantidade de dados de treinamento utilizados. Este processo é conhecido como pré treino e, embora seu custo computacional seja elevado, é usualmente realizado apenas uma vez. As representações de texto obtidas podem ser utilizadas para tarefas posteriores, através da prática de transferência de aprendizado (TORREY; SHAVLIK, 2010).

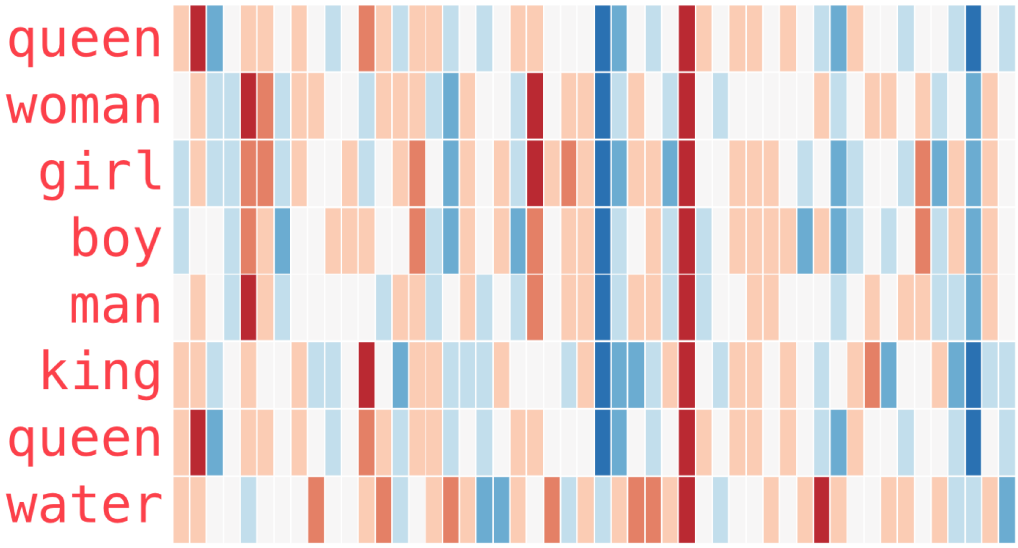


Figura 5 – Visualização de um vetor de *Embeddings*, é possível notar que algumas dimensões são correlacionadas entre alguns vetores, o que pode indicar características em comum como sexo, ou nível de realeza por exemplo. (Alammar, Jay , 2019)

Os *embeddings*, se tratando de vetores em um espaço N dimensional, apresentam propriedades interessantes. É possível, por exemplo, derivar analogias entre palavras como: rei está para rainha assim como homem está para mulher. Essas propriedades podem ser obtidas através do calculo de distancias ou ângulos entre estes vetores. Uma das formas de

se fazer isso é através da similaridade coseno, que pode ser obtida a partir do cálculo do coseno do angulo entre dois vetores A e B, conforme Equação 2.5. Vetores cuja diferença de angulo é pequena são correlacionados, enquanto vetores que são ortogonais são não correlacionados.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{2.5}$$

Portanto a diferença entre rei e rainha, que é um vetor, deve ter uma similaridade coseno alta com a diferença entre homem e mulher por exemplo. Isso significa que a diferença de angulo entre as diferenças destes *embeddings* é pequena.

Ao se gerar a representação na forma de *embeddings* de palavras em um texto, cada palavra fica associada a um vetor denso N dimensional. Na pratica, ao se treinar modelos que geram *embeddings* se fixa um tamanho de vocabulário e é construída uma tabela grande que associa cada palavra a seu vetor correspondente. Palavras que não pertencem ao vocabulário, assim como outros tipos de *tokens* específicos, são representados através de *tokens* especiais (MIKOLOV et al., 2013; PENNINGTON; SOCHER; MANNING, 2014). Um dos métodos pioneiros e mais utilizados para geração de *embeddings* é o algoritmo word2vec (MIKOLOV et al., 2013).

2.7.1 word2vec

Para gerar *embeddings*, o *word2vec*, partindo da hipótese distribucional, tenta prever qual a probabilidade de uma palavra específica aparecer perto de outras palavras em um texto. Estas outras palavras, dentro do modelo, podem ser escolhidas através de uma técnica conhecida como *skip-gram with negative sampling* (MIKOLOV et al., 2013).

O modelo de *skip-gram with negative sampling* conta com dois elementos importantes. O *skip-gram model* trata uma palavra e uma janela de palavras em volta dela como exemplos positivos para um classificador e usa as demais palavras do corpus como exemplos negativos. A partir deste processo, é treinado um modelo logístico e os pesos obtidos através deste modelo são usados como *embeddings*, que sumarizam o aprendizado da rede a respeito das palavras usadas no processo de treinamento do modelo.

Um problema com o método *skip-gram* puro é que, usar as demais palavras do corpus como exemplos negativos implicaria em um altíssimo custo computacional e daria origem a um problema altamente desbalanceado. Para tratar isso, a técnica de *negative sampling*, faz com que, em vez de serem escolhidas todas as demais palavras do corpus de palavras como exemplos negativos, k palavras sejam escolhidas de maneira aleatória para cada palavra a ser treinada (que gera um exemplo positivo), como pode ser visto na

Figura 6. Isso permite com que o algoritmo tenha um custo computacional bem menor e ainda gere uma boa representação das palavras.

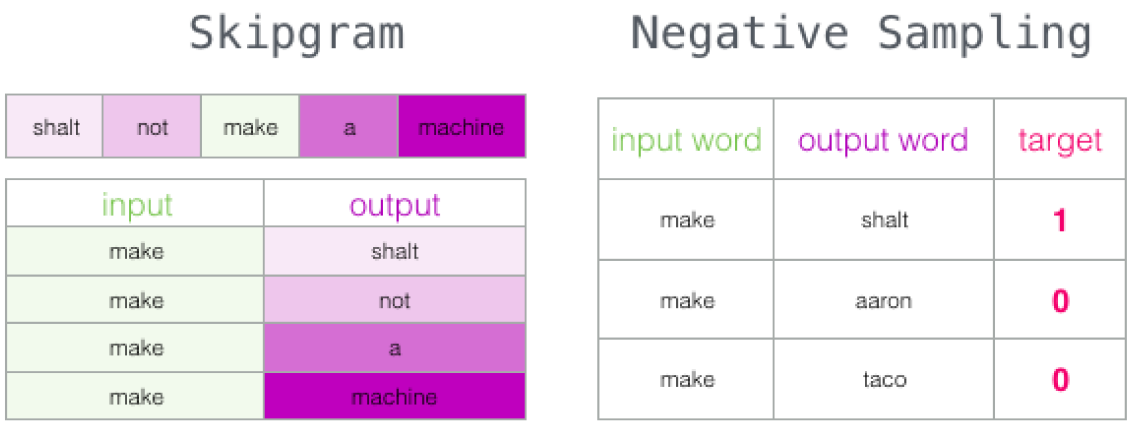


Figura 6 – Modelo *Skip-gram with negative sampling* (Alammar, Jay , 2019)

A Figura 7 mostra o processo de treinamento para obtenção dos *embeddings*, através da técnica de *skip-gram with negative sampling*. Esta consiste na multiplicação de uma matriz de contexto C e outra de *embeddings* W, onde W contém a representação vetorial da palavra central e C as palavras que se encontram próximas, em uma janela de tamanho k desta. Nessa formulação, é maximizada a similaridade entre as palavras analisadas, que se encontram na mesma janela de contexto k e, ao mesmo tempo, exemplos negativos (*negative samples*), tem sua similaridade com a palavra analisada minimizada.

A Equação 2.6 representa a função objetivo utilizada na técnica *skip-gram with negative sampling* para uma palavra w_j e uma palavra de contexto c_{j+1}. A primeira parte da equação minimiza produto da palavra w_j e a palavra de contexto c_{j+1} enquanto a segunda parte da equação busca maximizar a distancia entre w_j e exemplos negativos de 1 a k, representados por c_n. Uma vez otimizado este objetivo, que pode ser visto como o treinamento de um modelo logístico, a matriz W é utilizada como *embeddings* obtidos e a matriz C é descartada.

$$L(\theta) = log\sigma(w_j \cdot c_{j+1}) + \sum_{i=1}^{k} log\sigma[1 - (w_j \cdot c_n)] \tag{2.6}$$

O tamanho da janela utilizada, tanto para a seleção de elementos positivos e negativos para o modelo, influi nas características dos *embeddings* encontrados. Janelas menores de contexto levam a representações mais sintáticas, devido ao fato da informação vir de palavras imediatamente próximas, enquanto que janelas maiores levam a representações que são mais semânticas (JURAFSKY; MARTIN, 2019).

A utilização da técnica de *skip-gram with negative sampling* é categorizada como uma técnica neural para obtenção de *embeddings*, embora, na pratica, é visível que se trata

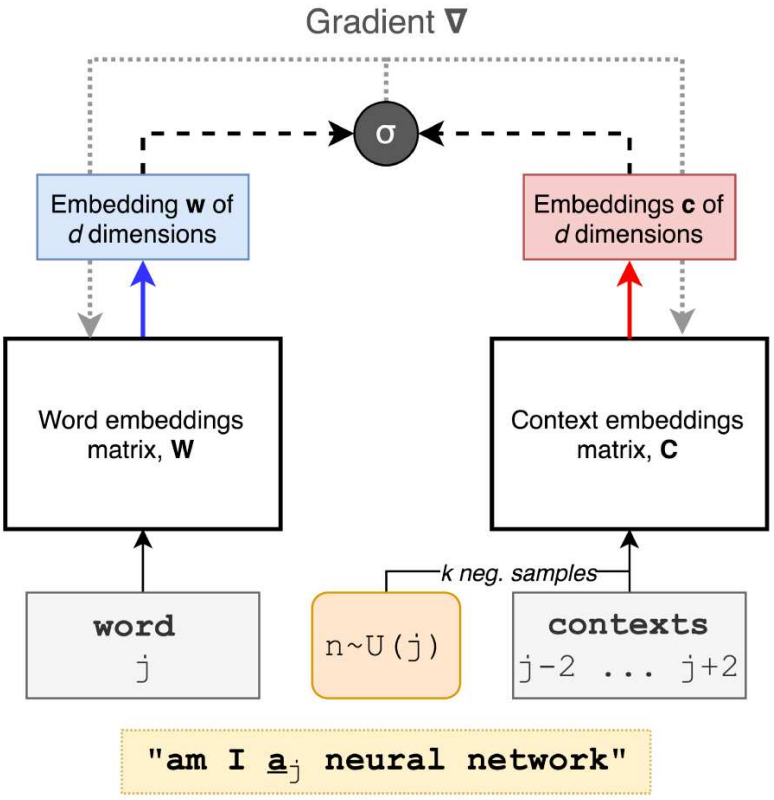


Figura 7 – Fluxograma de treinamento do modelo *Skip-gram with negative sampling*

da utilização de uma função que se assemelha bem mais a uma regressão logística.

Além da técnica de *skip gram with negative sampling*, utilizada pelo word2vec (MIKOLOV et al., 2013), existem outras técnicas para geração de *embeddings*, baseadas em outras metodologias. Dentre elas, se destaca a técnica *GloVe: Global Vectors for Word Representation* de (PENNINGTON; SOCHER; MANNING, 2014), que também gera *embeddings*, gerando representações na forma de vetores densos, que também incorporam características semânticas da linguagem, para cada palavra representada.

2.7.2 GloVe

GloVe (PENNINGTON; SOCHER; MANNING, 2014), ou *Global vectors for word representation* é uma técnica para representação de texto através de *embeddings*. Assim como o word2vec, o modelo GloVe tem como objetivo construir representações de texto na forma de vetores densos capazes de capturar significado semântico das palavras. A diferença entre as duas técnicas consiste no processo utilizado para geração dos *embeddings*.

Em GloVe (PENNINGTON; SOCHER; MANNING, 2014) os *embeddings* são gerados a partir de uma matriz de co-ocorrência de palavras, o que contrasta com o método *skip-gram with negative sampling* de word2vec, onde os *embeddings* são gerados através de um processo de janelamento, descrito na seção anterior. Uma matriz de co-ocorrência

de palavras X, pode ser vista tal como uma matriz onde cada entrada X_ij representa o número de vezes que uma palavra j ocorre em um mesmo contexto de uma palavra i (PENNINGTON; SOCHER; MANNING, 2014). Ao utilizar a matriz de co-ocorrência, o método GloVe busca explorar estatísticas a respeito da contagem de palavras em um corpus de texto, e ao mesmo tempo, extrair significado semântico do texto.

Para isso o modelo GloVe usa uma formulação matemática baseada no método de quadrados mínimos regularizado, que opera em cima da matriz de co-ocorrência de palavras, como pode ser visto através da equação 2.7, onde X_{ij} representam os termos da matriz de co-ocorrência, w_i o *embedding* gerado, $\tilde{w_j}$ o vetor de contexto (similar ao word2vec) e V o tamanho do vocabulário.

$$\sum_{ij}^{V} \text{weight}(X_{ij})(\text{dot}(w_i, \tilde{w_j}) + b_i + \tilde{b_j} - \log(X_{ij}))^2 \tag{2.7}$$

Em Pennington et al. (PENNINGTON; SOCHER; MANNING, 2014) o autor discute as vantagens do Glove em relação ao word2vec, que desconsidera estatísticas sobre as contagens de palavras, e sobre métodos clássicos de decomposição de matrizes para geração de *embeddings*, que não são capazes de capturar semântica nas palavras (PENNINGTON; SOCHER; MANNING, 2014).

Em geral, os *embeddings* gerados tanto pela técnica *skip gram with negative sampling*, presente em word2vec (MIKOLOV et al., 2013) e pelo modelo GloVe (PENNINGTON; SOCHER; MANNING, 2014), geram resultados bastante similares, na prática. No entanto, no trabalho onde é apresentado, através de uma argumentação teórica e embasada em testes empíricos, o modelo Glove é considerado mais robusto quando comparado ao word2vec (PENNINGTON; SOCHER; MANNING, 2014) principalmente devido ao fato de usar estatísticas a respeito da contagem de palavras, derivadas da matriz de co-ocorrência.

2.7.3 LSTMs e CNNs para texto

Embeddings não incorporam nativamente nenhuma informação pertinente à ordenação das palavras em um documento de texto. No entanto, como é claro pela estrutura da linguagem, existem dependências temporais entre as diversas palavras presentes em um documento (JURAFSKY; MARTIN, 2019). Redes recorrentes (RNNs - *Recurrent Neural Networks*) surgem para atacar o problema de dependência temporal em modelos de sequência (GOODFELLOW; BENGIO; COURVILLE, 2016). Redes recorrentes são compostas de células recorrentes conectadas entre si através de conexões temporais, de forma que compartilham estados escondidos entre cada atraso temporal (GOODFELLOW; BENGIO; COURVILLE, 2016). A entrada dessas células é usualmente cada elemento

da sequência, que geram o estado escondido h e uma saída y. Para o caso de modelos de classificação de texto, usualmente apenas a saída y equivalente ao último instante de tempo da sequência avaliada é utilizada para gerar o rótulo de saída.

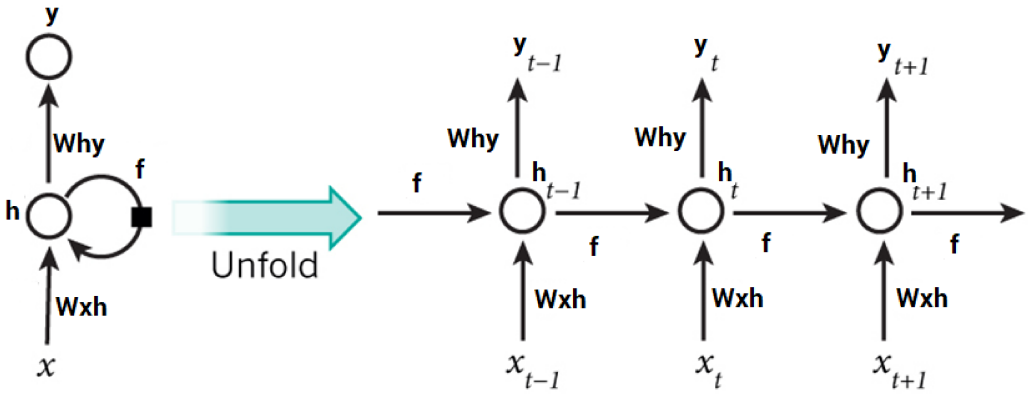


Figura 8 – RNNs (Gupta, Dishashree, 2019)

A figura 8 mostra a célula recorrente e sua versão desenrolada no tempo, onde são mostradas as conexões temporais entre cada elemento da sequência avaliada. Entre cada conexão, existem pesos que aplicam transformações às suas entradas. A rede é governada pelo seguinte conjunto de equações:

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{2.8a}$$

$$y_t = W_{hy}h_t \tag{2.8b}$$

A função f, na figura 8, representa a função para obtenção de h_t, vista na equação 2.8a. Os subscritos de cada matriz W indicam o sentido dos pesos dos neurônios da célula recorrente. É perceptível que é utilizada a mesma matriz de pesos para cada instante de tempo, o que significa que os pesos em uma rede recorrente são compartilhados entre os diferentes instantes de tempo. Isso é algo comum em técnicas de *deep learning* e é útil tanto para reduzir custo computacional como também ajuda na capacidade de aprendizado e generalização do modelo (GOODFELLOW; BENGIO; COURVILLE, 2016).

Um dos problemas com as redes recorrentes simples é que, ao serem otimizadas, o gradiente que é transmitido entre os pesos das camadas escondidas, através das conexões temporais, pode explodir ou decair a um nível em que os pesos das camadas iniciais não conseguem mais serem atualizados (GOODFELLOW; BENGIO; COURVILLE, 2016). Este ultimo problema é mais critico e ganha o nome de *vanishing gradients*, sendo muito comum a redes profundas, principalmente redes com arquitetura recorrente, visto que nestas a propagação da informação acontece no tempo, e pode ser durante um intervalo considerável de instantes de tempo (PASCANU; MIKOLOV; BENGIO, 2013).

Como uma forma de evitar o fenômeno de *vanishing gradients*, um tipo específico de célula recorrente é usualmente utilizado, chamada LSTM (HOCHREITER; SCHMIDHUBER, 1997). A forma como a informação se propaga através dos diferentes instantes de tempo em uma rede LSTM é diferente de como é propagada em uma rede recorrente convencional. Nesta são usados *gates*, ou portões, por onde a informação pode fluir livremente sem a multiplicação desta por pesos. Assim, o fenômeno de *vanishing gradients* é atenuado.

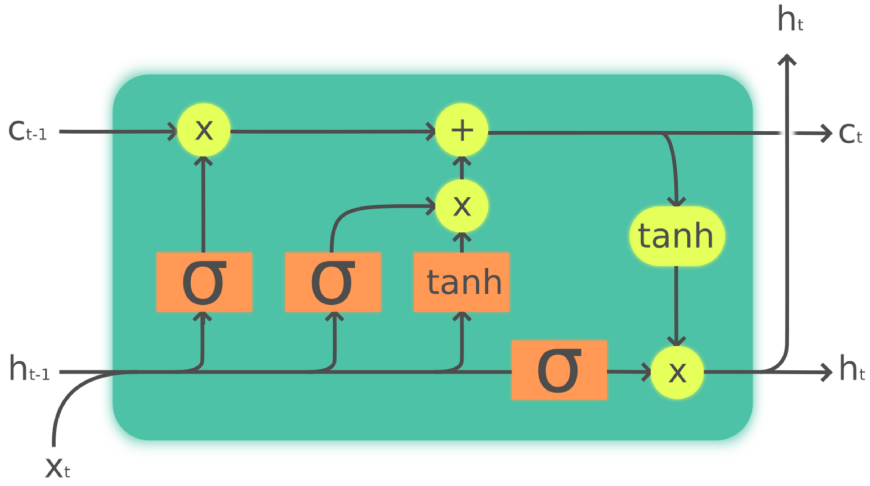


Figura 9 – LSTM cell (Wikipedia contributors, 2004)

Através das ativações sigmoidais, e como pode ser visto na figura 9, a célula da rede LSTM tem a capacidade de cortar ou deixar passar o sinal, devido à característica da função sigmoide logística, presente em cada *gate* da célula LSTM. Em vez de utilizar apenas um vetor h, que equivale ao estado escondido, a célula LSTM também utiliza um vetor de contexto c, que pondera o vetor h através de um portão, e cujo objetivo é determinar quanto do histórico importa em relação ao estado anterior. Equivalentemente, c também é alterado devido ao estado anterior. Esse processo de interação entre os *gates* e pesos é inteiramente aprendido com o processo de otimização e ajuste aos dados e pode ser visto através das equações que governam a célula LSTM:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
\tag{2.9}
$$

Onde:

$$x_t \in \mathbb{R}^d : \text{é o vetor de entrada em cada instante de tempo} \tag{2.10a}$$

$$f_t \in \mathbb{R}^h : \text{é o vetor de ativação do } \textit{forget gate} \tag{2.10b}$$

$$i_t \in \mathbb{R}^h : \text{é o vetor de ativação do } \textit{input gate} \tag{2.10c}$$

$$o_t \in \mathbb{R}^h : \text{é o vetor de ativação do } \textit{output gate} \tag{2.10d}$$

$$h_t \in \mathbb{R}^h : \text{é o vetor do estado escondido} \tag{2.10e}$$

$$c_t \in \mathbb{R}^h : \text{é o vetor de estado da célula, ou contexto} \tag{2.10f}$$

$$W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}, b \in \mathbb{R}^h : \text{são matrizes de pesos} \tag{2.10g}$$

Os gates comandam o processo de esquecimento e lembrança das informações que fluem no tempo na LSTM. Por sofrerem pouco com o problema de longas dependências no tempo e por serem de uma arquitetura recorrente ideal para lidar com problemas de sequências, as redes LSTM são bastante utilizadas para lidar com problemas em NLP cuja entrada são *embeddings*.

Células recorrentes, como a LSTM, no geral, recebem como entrada o vetor de entrada em seus diversos instantes de tempo e o vetor ou vetores relativos a representação h de sua camada escondida. Como saída, geralmente tem-se que algum tipo de transformação é aplicada a h para ser gerado o rótulo y e, além disso, o próprio h funciona como uma saída para as os instantes de tempo subsequentes.

No entanto, existem varias formas de se conectar diversas saídas e entradas dos diferentes instantes de tempo de uma rede recorrente, que dão origem a arquiteturas distintas, onde cada uma serve propósitos diversos.

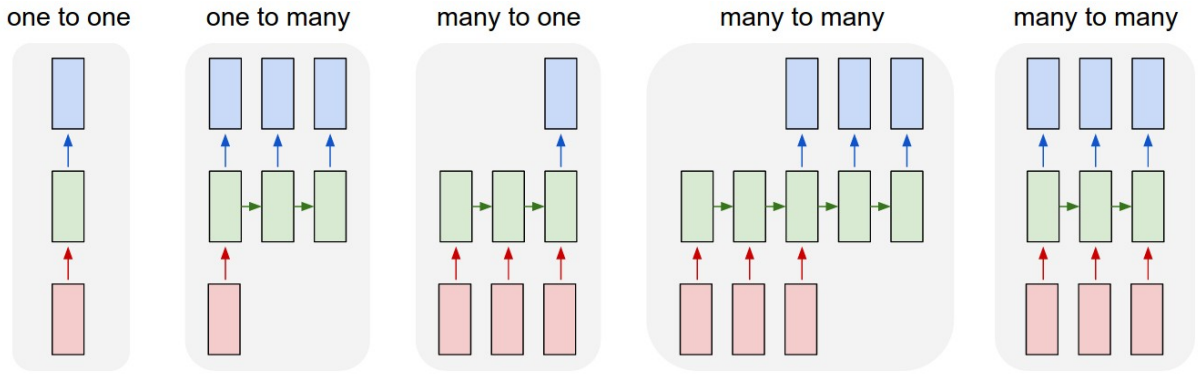


Figura 10 – Diferentes arquiteturas de redes recorrentes (Wikipedia contributors, 2004)

A figura 10 exemplifica alguns tipos de arquiteturas possíveis para redes recorrentes. A arquitetura *many to one* é geralmente utilizada na tarefa de classificação de texto. A arquitetura *many to many* é utilizada quando a saída da rede também é uma sequência, o que acontece em problemas de tradução de linguagem, por exemplo (*neural machine*

translation) (LUONG; PHAM; MANNING, 2015). Este tipo de problema também é conhecido como *seq2seq*, pois a entrada e a saída da rede são sequências.

Além destes diversos tipos de arquitetura, as redes recorrentes podem ser bi-direcionais, como mostrado na Figura 11. Redes bi-LSTM funcionam exatamente igual a redes LSTM normais porém uma segunda célula LSTM agora computa os estados escondidos, entradas e saídas, em ordem oposta, do ultimo elemento da sequencia para o primeiro. Uma vez computado, o resultado obtido é concatenado com o resultado de uma célula LSTM que opera na ordem convencional constituindo o resultado final da rede bi-LSTM.

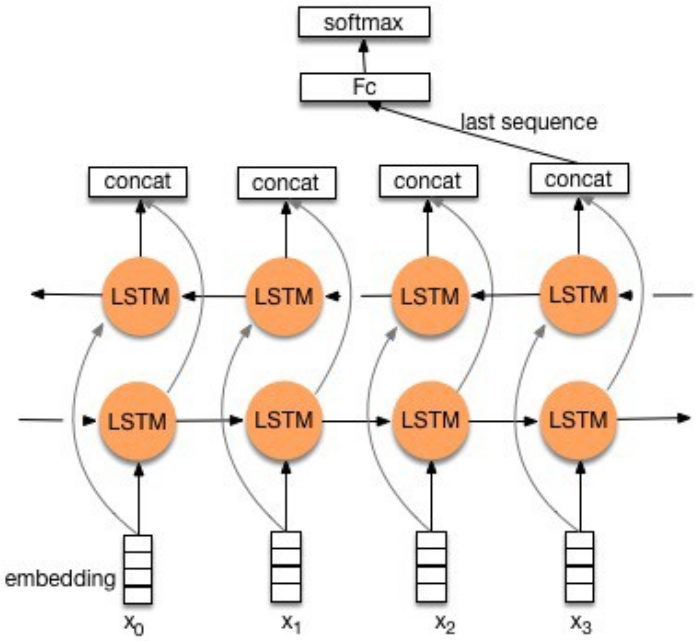


Figura 11 – Bi-LSTM (Lee, Ceshine, 2019)

A vantagem obvia das redes bi-LSTM é seu caráter não causal e o fato de conseguirem incorporar informações de instantes de tempo futuros. Isso é particularmente útil para tarefas de NLP e para tarefas de classificação de texto, pois existem elementos na linguagem tais como concordância verbal, negação, dentre outros, que acontecem em ordem invertida dentro de uma frase, de forma que uma rede bi direcional teria mais facilidade em capturar esse comportamento.

Além de arquiteturas recorrentes, também é possível utilizar redes convolucionais, CNNs (*Convolutional Neural Networks*), para tarefas de classificação de texto (KIM, 2014).

As redes convolucionais são redes usualmente utilizadas para lidar com imagens, como ilustrado na Figura 12. Foram criadas usando como inspiração o córtex frontal cerebral, que é a região responsável pelo processamento de imagens no cérebro humano (GOODFELLOW; BENGIO; COURVILLE, 2016). Através de operações de convolução

e *pooling*, estas redes extraem representações dos dados em uma hierarquia de conceitos (GOODFELLOW; BENGIO; COURVILLE, 2016).

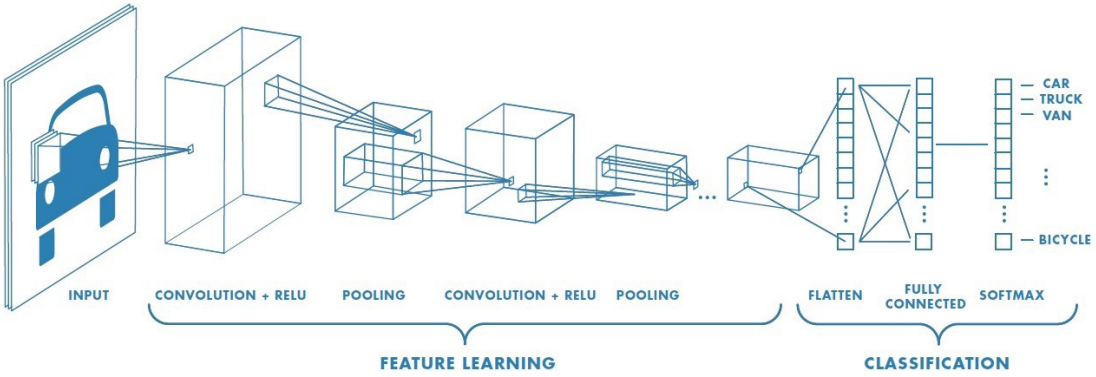


Figura 12 – Exemplo da arquitetura de uma rede convolucional, porém para lidar com imagens (Wikipedia contributors, 2004)

As camadas de convolução aplicam um operador de convolução nos dados, utilizando *kernels*, também chamados de filtros, que são aprendidos durante o treinamento da rede. Matematicamente, a operação de convolução é descrita através da equação 2.11, onde K representa a função de *kernel*, que opera em uma imagem I de tamanho m por n (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.11}$$

Usualmente, uma vez que as operações de convolução são aplicadas, etapas de *pooling* fazem a agregação da informação obtida, em uma espécie de sub amostragem, usando algum tipo operador. O operador de máximo (*max pooling*) é o mais comumente utilizado durante a etapa de *pooling*.

Redes convolucionais tem propriedades interessantes que garantem determinadas características especiais. Dentre elas destacam-se o compartilhamento dos parâmetros que existem na rede, a esparsidade de suas interações e invariância a pequenas translações (GOODFELLOW; BENGIO; COURVILLE, 2016). Estas propriedades, por sua vez, derivam da aplicação das operações de convolução e *pooling*. De maneira prática, a invariância a pequenas translações pode ser uma propriedade particularmente útil quando se mostra mais importante detectar se uma característica está presente do que onde exatamente ela se encontra em um texto ou imagem.

Para o contexto de classificação de texto, o operador de convolução envolve um filtro $W \in IR^{hk}$ aplicado a uma janela de h palavras representadas através de *embeddings* de tamanho k. Já a operação de *pooling*, aplicada sobre o resultado da convolução, é conhecida como *max-over-time pooling*, neste caso. Tanto a operação de convolução quanto de *pooling* operam no eixo temporal da sentença para tarefas envolvendo dados de texto, mantendo a ordem da frase.

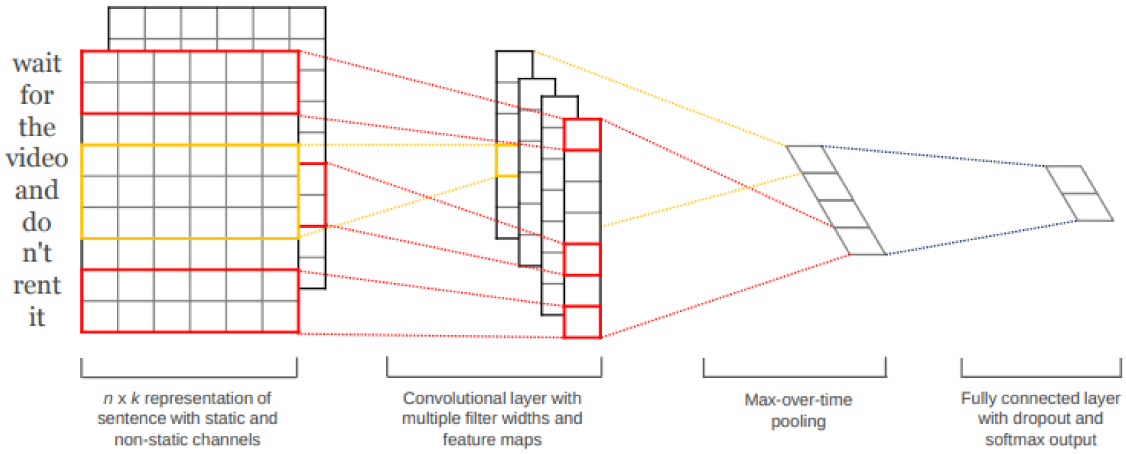


Figura 13 – CNN para classificação de texto (KIM, 2014)

As propriedades descritas anteriormente a respeito das redes convolucionais se aplicam igualmente para tarefas de classificação de texto. O fato da rede ser invariante a pequenas translações significa, neste contexto, que esta é adaptada para achar um conjunto de palavras descritivas onde quer que elas estejam na frase. A figura 13 ilustra a arquitetura de redes CNN para tarefas onde os dados de entrada são textuais.

2.8 Transferência de aprendizado

Transferência de aprendizado (ou *Transfer learning* em inglês) consiste na pratica de aplicar o aprendizado obtido através do treinamento de um modelo em uma tarefa, em outra, usualmente relacionada ao problema original (TORREY; SHAVLIK, 2010). Essa é uma pratica comum na utilização de modelos de *deep learning*, visto que estes quando treinados do zero em uma tarefa específica tendem ao sobreajuste. A pratica da transferência de aprendizado viabiliza a utilização de modelos grandes em bases de dados menores, através de um treinamento anterior que extrai características que podem funcionar como características genéricas, passiveis de serem aproveitadas para a tarefa subsequente. Assim, ao se usar técnicas de transferência de aprendizado, é visado manter a capacidade de generalização do modelo, obtida através do treinamento prévio, e ao mesmo tempo se especializar em padrões presentes no novo conjunto de dados.

Em um cenário indutivo, como o apresentado na Figura 14, a prática de transferência de aprendizado pode ser vista como a realização de uma busca direcionada em um espaço de busca pré-especificado, dado o treinamento anterior do modelo (também chamado de pré treino).

Existem duas formas de se realizar o procedimento de transferência de aprendizado (Karpathy, Andrej, 2019). A primeira consiste em usar o modelo pre treinado como um extrator de características. Isso usualmente envolve extrair a representação advinda de

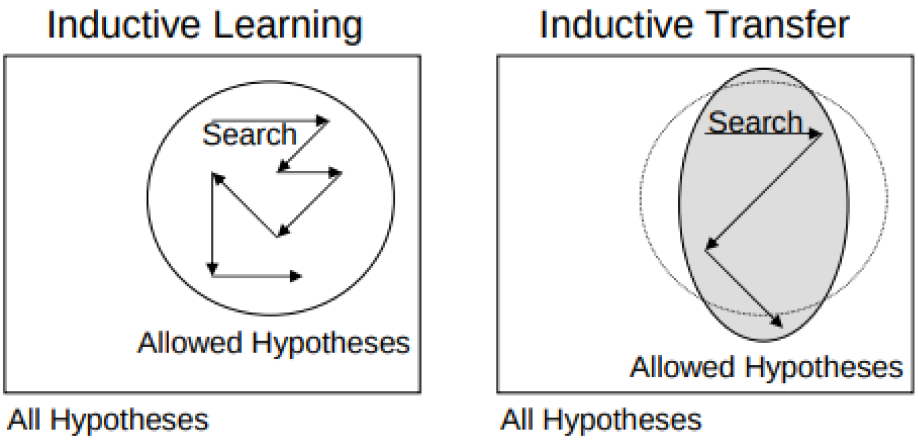


Figura 14 – Transferência de aprendizado (TORREY; SHAVLIK, 2010)

camadas ou estados escondidos da rede utilizada(JURAFSKY; MARTIN, 2019). Após esta etapa, um segundo modelo é treinado em cima da representação obtida através da fase anterior. A segunda maneira de se realizar o procedimento consiste em usar toda a rede para a tarefa subsequente e, durante o treinamento com os novos dados, ajustar toda a estrutura de pesos do modelo original. Esta prática ganha o nome específico de ajuste fino (*fine-tuning*).

Existem alguns cenários onde é apropriado a utilização do modelo como extrator de características e outros onde é recomendado a utilização do procedimento de ajuste fino, como pontuado em Karphathy et al. (Karpathy, Andrej, 2019). Usualmente, quando o conjunto de dados da tarefa subsequente é pequeno, não é recomendado realizar o ajuste fino devido a uma alta probabilidade de sobreajuste. Neste caso é recomendado a utilização do modelo como extrator de características. Em contrapartida, o oposto é verdade quando o conjunto de dados da tarefa subsequente é grande. Quando o novo conjunto de dados tem uma distribuição de dados muito diferente da utilizada no pré-treino da rede, também é recomendada a utilização do modelo como extrator de características. No entanto, quando esse mesmo conjunto de dados é muito grande, a utilização da técnica de ajuste fino pode ser uma alternativa viável (Karpathy, Andrej, 2019).

Neste trabalho, o ajuste fino é utilizado com uma base de dados bastante reduzida nos métodos BERT e XLNet. Estes desafiam este guia para a prática de transferência de aprendizado apresentado. Isso foi feito devido às características da arquitetura destes modelos, que levam a crer que estes funcionam bem neste cenário, e que serão elucidadas nas seções subsequentes.

2.9 Mecanismo de atenção

O mecanismo de atenção, ou *Attention mechanism*, é uma das ideias recentes no ramo de pesquisa em *deep learning* mais influentes atualmente (VASWANI et al., 2017). Esta técnica foi inicialmente empregada em tarefas de *machine translation*, onde um modelo *seq2seq* é usado e tem como entrada uma sequência de texto em uma língua e como saída a mesma sequência em outra língua.

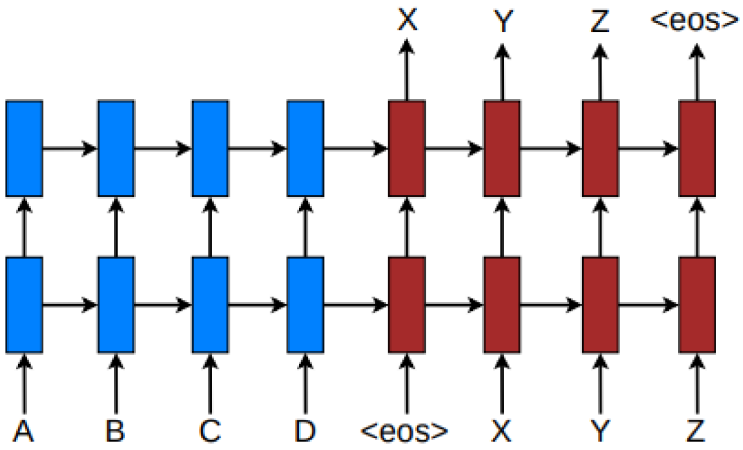


Figura 15 – Arquitetura para tarefas de *Neural machine translation* (LUONG; PHAM; MANNING, 2015)

Na figura 15 é possível ver uma arquitetura padrão para um modelo de *neural machine translation*. Neste modelo, a parte azul, chamada de *encoder*, é onde entra o texto original, não traduzido. A parte vermelha é o *decoder*, responsável por gerar as palavras traduzidas no modelo. Usualmente, modelos de *machine translation* utilizam arquiteturas recorrentes, logo, cada célula, representada através dos quadrados na figura, pode ser vista como uma célula de alguma arquitetura recorrente, como uma célula LSTM, por exemplo. Na figura vemos uma rede com duas camadas de profundidade (LUONG; PHAM; MANNING, 2015).

A ponte entre o *encoder* e o *decoder* é feita através da propagação do estado escondido relativo ao ultimo instante de tempo do *encoder*, chamado usualmente de vetor de contexto. A utilização do vetor de contexto como única entrada do *encoder* para o *decoder* se mostrou um gargalo para este tipo de modelo, especialmente em sentenças longas. A solução proposta veio através da utilização do mecanismo de atenção, proposto em Luong et al (LUONG; PHAM; MANNING, 2015).

O mecanismo de atenção funciona de forma que, a cada instante de tempo na fase de *decoding* do modelo, o *decoder* leve como entrada uma concatenação de seu estado escondido atual h_t junto com um vetor de contexto c_t, de mesma dimensão de h_t, derivado do mecanismo de atenção. Este vetor é então utilizado para produzir uma saída y_t, que é

mapeada para uma palavra traduzida. Este processo pode ser visto através das equações 2.12 e 2.13 e da figura 16.

$$\tilde{h}_t = tanh(W_c[c_t; h_t]) \tag{2.12}$$

$$p(y_t|y_{<t}, x) = softmax(W_s\tilde{h}_t) \tag{2.13}$$

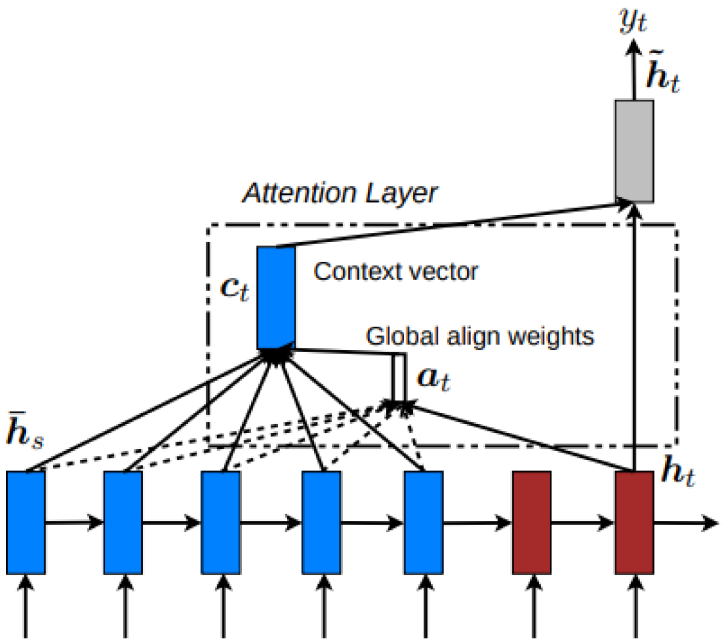


Figura 16 – Mecanismo de atenção (LUONG; PHAM; MANNING, 2015)

A geração do vetor de contexto c_t depende do vetor a_t, chamado vetor de alinhamento, cuja dimensão é igual ao número de instantes de tempo. Este é calculado através das seguintes equações:

$$a_t(s) = align(h_t\tilde{h}_s) = \frac{exp(score(h_t\tilde{h}_s))}{\sum_{s'} exp(score(h_t\tilde{h}_s))} \tag{2.14a}$$

$$score(h_t\tilde{h}_s) = h_t^T\tilde{h}_s \quad (dot) \quad or \tag{2.14b}$$

$$score(h_t\tilde{h}_s) = h_t^T W_a\tilde{h}_s \quad (general) \quad or \tag{2.14c}$$

$$score(h_t\tilde{h}_s) = v_a^T tanh(W_a[h_t; \tilde{h}_s]) \quad (concat) \tag{2.14d}$$

$$\tag{2.14e}$$

O vetor de alinhamento usa os estados escondidos $\tilde{h}_s$ do *encoder* em conjunção com o estado atual a ser previsto no *decoder*, h_t, através das equações acima, para criar pesos de alinhamento, que são multiplicados novamente pelos estados escondidos do *encoder*, $\tilde{h}_s$,

gerando uma média ponderada que da origem ao vetor c_t (LUONG; PHAM; MANNING, 2015). Esse processo pode ser visto através da figura 16. Por sua vez, conforme descrito anteriormente, o vetor c_t é usado em conjunção com h_t (equação 2.12), gerando $\tilde{h}_t$ e por fim a saída y_t (equação 2.13).

Em alto nível, o mecanismo de atenção garante ao modelo a capacidade de determinar quais partes da entrada são relevantes, através da utilização e ponderação dos estados escondidos do *encoder*, e fornece essa informação para auxiliar o *decoder* no seu processo de geração do texto traduzido. Esse mecanismo foi de grande valor para auxiliar em tarefas de tradução de texto (LUONG; PHAM; MANNING, 2015).

O mecanismo de atenção descrito nesta seção tem o nome de *global attention* (LUONG; PHAM; MANNING, 2015), e é a forma tradicional do mecanismo de atenção. No entanto, existem outras formas de *attention*, dentre elas, o mecanismo de *self attention*, utilizado em arquiteturas *transformer*, que serão discutidas na próxima seção.

2.10 O Transformer

A eficácia do mecanismo de atenção, junto com o fato de que este é altamente paralelizável, motivaram a criação de um modelo baseado somente neste para solução de problemas de *machine translation*, e cuja arquitetura ganhou o nome de *transformer* (VASWANI et al., 2017). O *transformer* dispensa a utilização de modelos recorrentes, tipicamente usados para este tipo de tarefa (LUONG; PHAM; MANNING, 2015). O título do artigo de Vaswani et al., onde este modelo é apresentado, faz uma alusão ao fato da arquitetura *transformer* usar somente o mecanismo de atenção e ainda sim atingir resultados estado da arte no campo de *machine translation*, com o nome "*Attention is all you need*".

Sendo um modelo para realização de tarefas de *machine translation*, o *transformer* segue uma arquitetura *encoder-decoder*, mesmo que baseada somente em mecanismos de atenção. A arquitetura da rede pode ser vista na figura 17.

De maneira simplificada, removendo as camadas de *add & norm*, as camadas de *encoder* e *decoder* podem tem suas operações principais representadas na figura 18.

Na parte do *encoder*, é possível ver através da figura 18, que primeiramente a entrada passa por uma camada de *self-attention*, depois por uma rede *feed-foward* convencional. Isso é feito um numero N de vezes, pois existem N blocos que realizam essa operação. Após o primeiro bloco, os demais usam as saídas dos anteriores como entrada. No primeiro bloco, a palavra é primeiro transformada em um *embedding*, que é atualizado com o treinamento da rede. Assim como em *embeddings* tradicionais, também é usado um tamanho de vocabulário fixo para as palavras presentes no corpus de texto e *tokens*

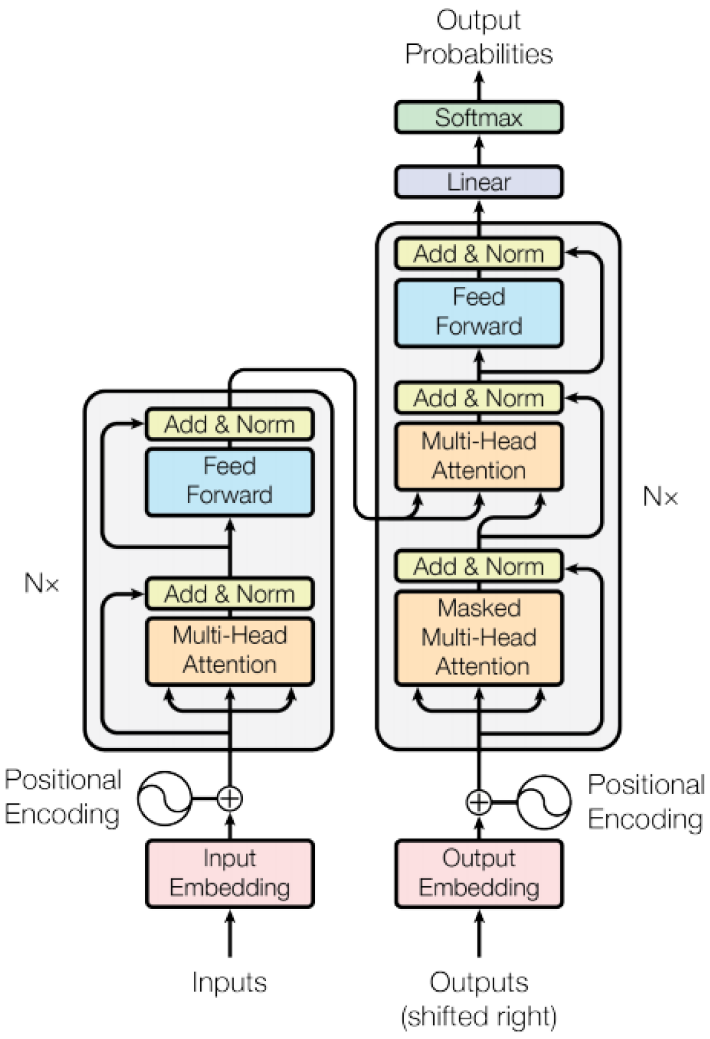


Figura 17 – Arquitetura do modelo *transformer* (VASWANI et al., 2017)

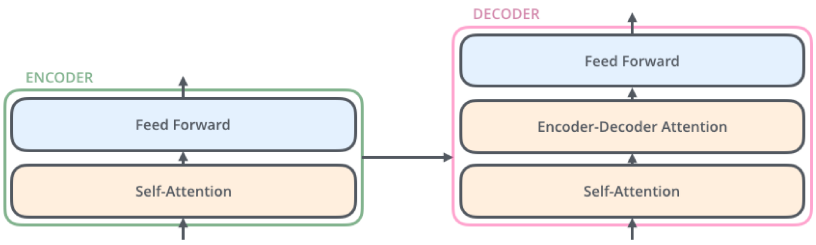


Figura 18 – *Encoder* e *decoder* da arquitetura *transformer* em uma visão simplificada (Alammar, Jay , 2019)

especiais (utilizados para representar palavras faltantes ou que tem algum significado especial).

O mecanismo de *self-attention* difere do mecanismo de *attention* global, descrito na seção anterior, pois este não se dá entre a camada do *encoder* para o *decoder*. Em vez disso, o mecanismo de *self-attention* tenta elencar quais palavras, dentro de um documento, são importantes para representar a palavra analisada no momento em que esta é processada, como ilustrado na Figura 19. Isso ajuda a construir uma representação para esta palavra que é mais robusta e que leva informação contextual das palavras que a cercam para a construção de sua representação.

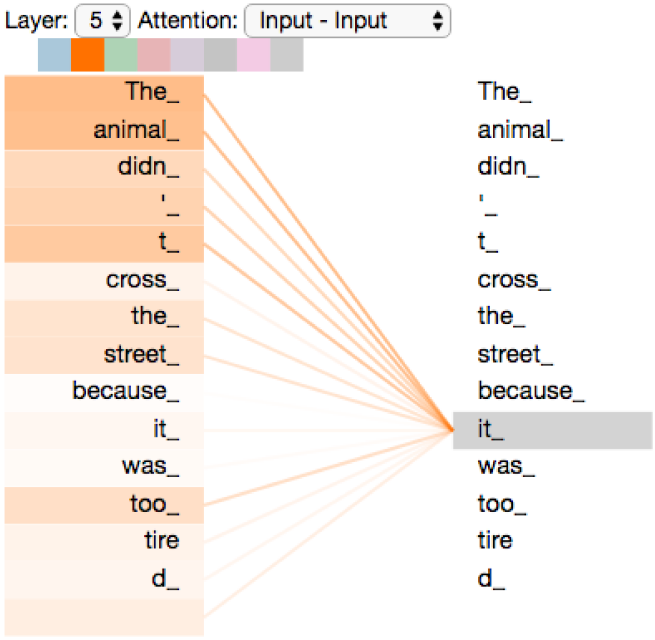


Figura 19 – Mecanismo de *self-attention* (Alammar, Jay , 2019)

Para calcular a matriz de *self-attention* são realizadas uma série de operações matriciais, que podem ser explicadas através de três abstrações. Estas são as matrizes *query, key e value*. Assumindo uma matriz X, onde cada linha é representa uma palavra e cada coluna é uma dimensão n da representação em formato de *embedding* desta mesma palavra, são utilizadas três matrizes, Wq, Wk e Wv, que quando multiplicadas pela matriz origem X, dão origem às abstrações anteriormente descritas. As matriz Wq, Wk e Wv são de tamanho n, onde n é o tamanho do vetor de *embeddings*, por l, onde l é um hyperparametro do modelo. Essas matrizes tem seus parâmetros treinados com o treinamento do modelo e são mostradas na Figura 20.

Com as matrizes *query, key e value* a matriz de *self-attention* é gerada conforme ilustrado na figura 21, onde é representada pela matriz Z. d_k é uma constante que equivale ao numero de dimensões de cada vetor da matriz de *keys*.

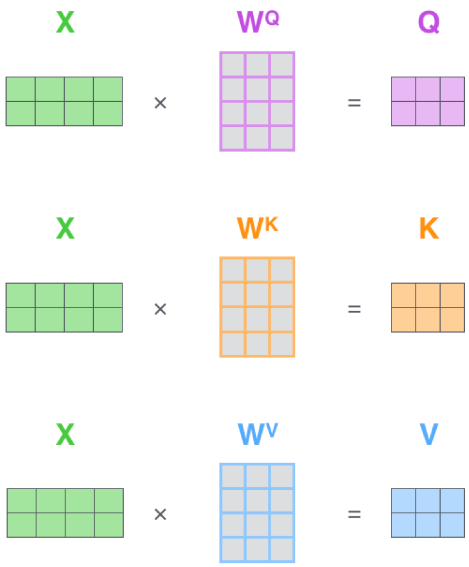


Figura 20 – Cálculo das matrizes *Query, Key, Value* para o mecanismo de *self-attention* (Alammar, Jay , 2019)

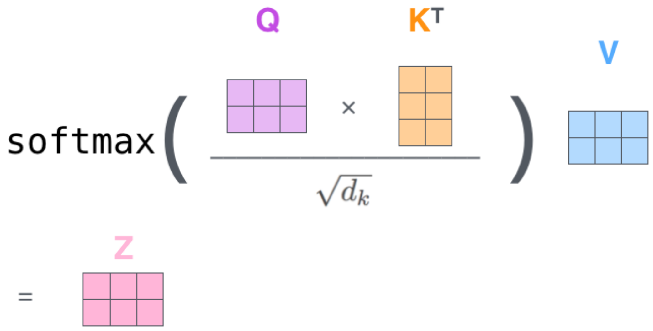


Figura 21 – Calculo da matriz de *self-attention* usando as matrizes *Query, Key, Value* (Alammar, Jay , 2019)

Uma vez gerada, a matriz Z de *self-attention* passa pela rede *feed-foward*. Entretanto, no processo são geradas várias matrizes Z para serem utilizadas pela arquitetura *transformer*, de forma paralela. Esse procedimento tem o nome de *multi-headed attention*. Basicamente, o mesmo processo de *self-attention* é replicado várias vezes de forma a garantir que diferentes *attention heads*, que é o nome dado a cada etapa paralela da aplicação do mecanismo de *self-attention*, aprendam a focar em diferentes partes importantes para representar um *token* dentro da frase. Esse procedimento pode ser visto na figura 22.

Na figura 23 é mostrado a importância que é dada a cada *token* de uma sentença na representação de um *token* específico, dado o mecanismo de *self attention multi head*. É possível notar que cada *attention head* foca em uma parte especifica da sentença para representar o *token* desejado.

Algo notável da arquitetura *transformer* é que essa, ao realizar a tarefa de *machine*

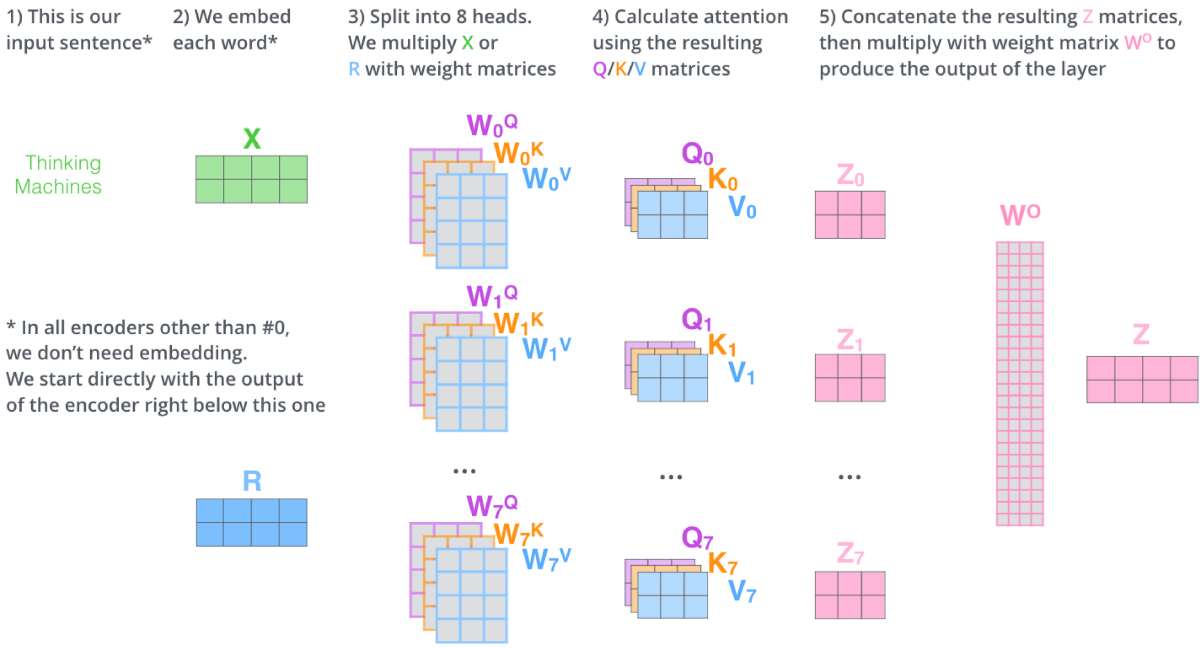


Figura 22 – *Multi-headed self-attention* (Alammar, Jay , 2019)

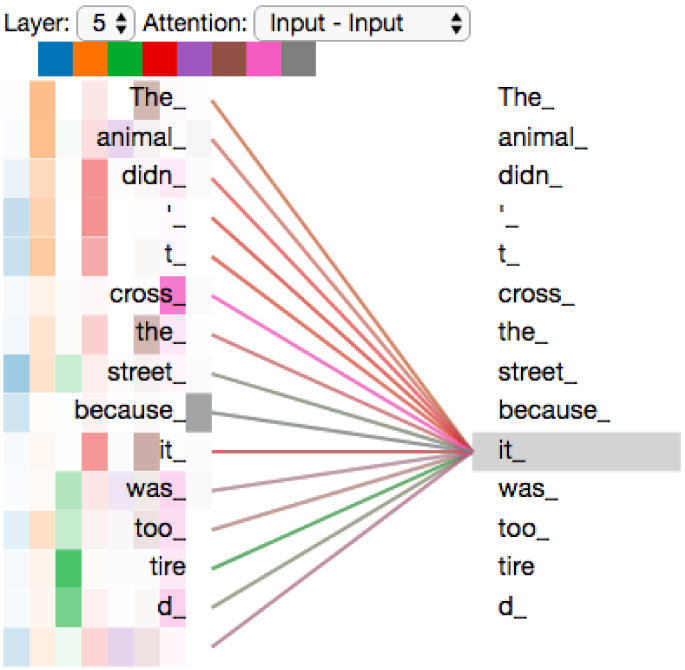


Figura 23 – Visualização do mecanismo de *Multi-headed self-attention* (Alammar, Jay , 2019)

translation, se baseia apenas no mecanismo de atenção e elimina a característica recorrente, presente nas recorrentes LSTM, por exemplo. No entanto, é importante para esta tarefa levar em consideração a ordem do texto e, por isso, na representação inicial no formato de *embeddings*, na primeira camada do *encoder*, é acrescentado a esse *embedding* um sinal posicional (VASWANI et al., 2017), como mostrado na figura 24. Com isso, o *transformer* consegue não perder a informação de posição das palavras no documento e ainda sim ser altamente paralelizavel, ao eliminar mecanismos de recorrência. Por esse motivo, como discutido em Vaswani et al. (VASWANI et al., 2017), a arquitetura *transformer* consegue fazer um uso bem mais otimizado de hardware, especialmente na forma de GPUs (VASWANI et al., 2017).

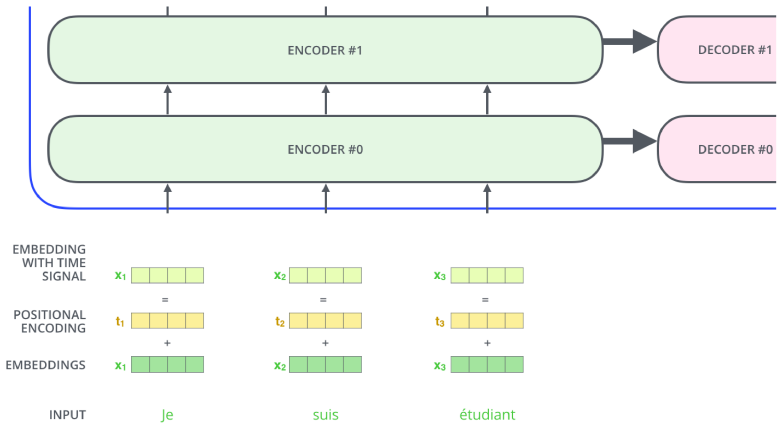


Figura 24 – *Positional Encoding* (Alammar, Jay , 2019)

Na figura 17, é possível observar que entre cada camada de *attention* existe uma conexão direta entre a própria entrada e a saída da camada anterior e também uma operação de *add & norm*. Essas duas etapas são a aplicação de procedimentos de *residual conections* (HE et al., 2016) e *layer normalization* (BA; KIROS; HINTON, 2016). A figura 25 mostra com mais detalhes estas duas etapas, que por sua vez ajudam com o processo de treinamento do modelo *transformer*, e são boas práticas para arquiteturas de redes profundas (HE et al., 2016). Os principais benefícios destes procedimentos são ajudar com a estabilidade durante o processo de treinamento do modelo e, consequentemente, reduzir o tempo de treinamento do modelo no geral(BA; KIROS; HINTON, 2016). Como a arquitetura *transformer* é uma arquitetura bem profunda, estas etapas causam melhora na estabilidade da propagação do sinal de treinamento durante o processo de aprendizado (VASWANI et al., 2017).

A conexão entre o *encoder* e o *decoder* do modelo *transfomer* utiliza os estados gerados pela ultima camada do *encoder* através de uma camada de *encoder-decoder attention*, que faz com que o *decoder* foque nas partes apropriadas da representação do *encoder* obtida, como pode ser visto através da Figura 17. A partir disso, o *decoder* então gera a tradução apropriada para cada instante de tempo de decodificação até que um

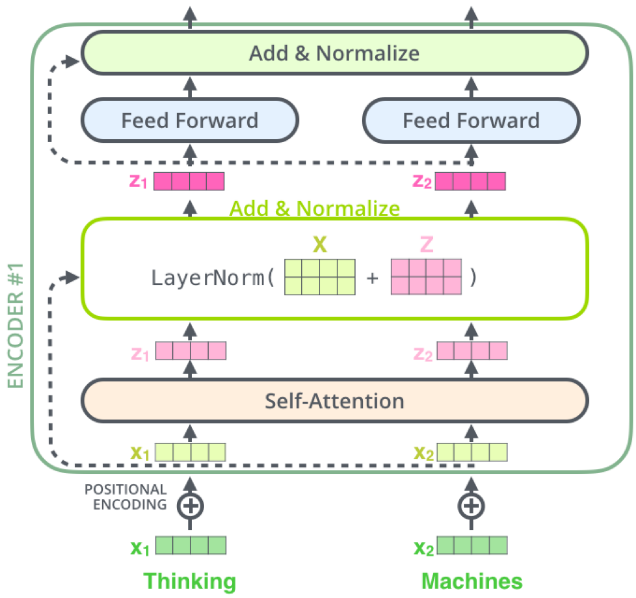


Figura 25 – *Residual Conections* e *Layer Normalization* (Alammar, Jay , 2019)

simbolo de fim de sentença seja gerado, sinalizando que a tradução daquela sentença chegou ao fim.

As diferenças fundamentais entre as camadas do *decoder* e o *encoder*, como pode ser visto através da figura 17, é que a primeira etapa no processo de decodificação usa um processo de *masked multi-head attention*, em contraste com o processo de *multi-head attention* do *encoder*, discutido anteriormente. Além disso, a camada de *multi-head attention* subsequente usa como entrada a saída do *encoder* e a representação da etapa anterior de *masked multi-head attention*, descrita no paragrafo anterior como camada de *encoder-decoder attention*.

A camada de *masked multi-head attention* funciona de maneira análoga ao mecanismo já explicado de *self-attention* com a diferença de que o *decoder* é permitido observar apenas elementos anteriores ao *token* analisado na sentença de saída, para evitar vazamento de informação, que tornaria a decodificação trivial. Dessa forma, *tokens* subsequentes são mascarados durante o processo de decodificação.

A camada de *encoder-decoder attention* difere do mecanismo de *multi-headed self attention* por considerar a saída do *encoder* como entrada. Esta por sua vez opera de forma análoga ao procedimento convencional de *self-attention* utilizado na arquitetura *transformer* porém, as matrizes K e V são obtidas através da saída do *encoder* e a matriz Q é gerada através do próprio *decoder*. Esse processo pode ser visto através da Figura 26.

Por fim, o vetor gerado pelo *decoder* passa por uma camada linear e posteriormente por uma camada *softmax*, onde são geradas a palavras traduzidas previstas.

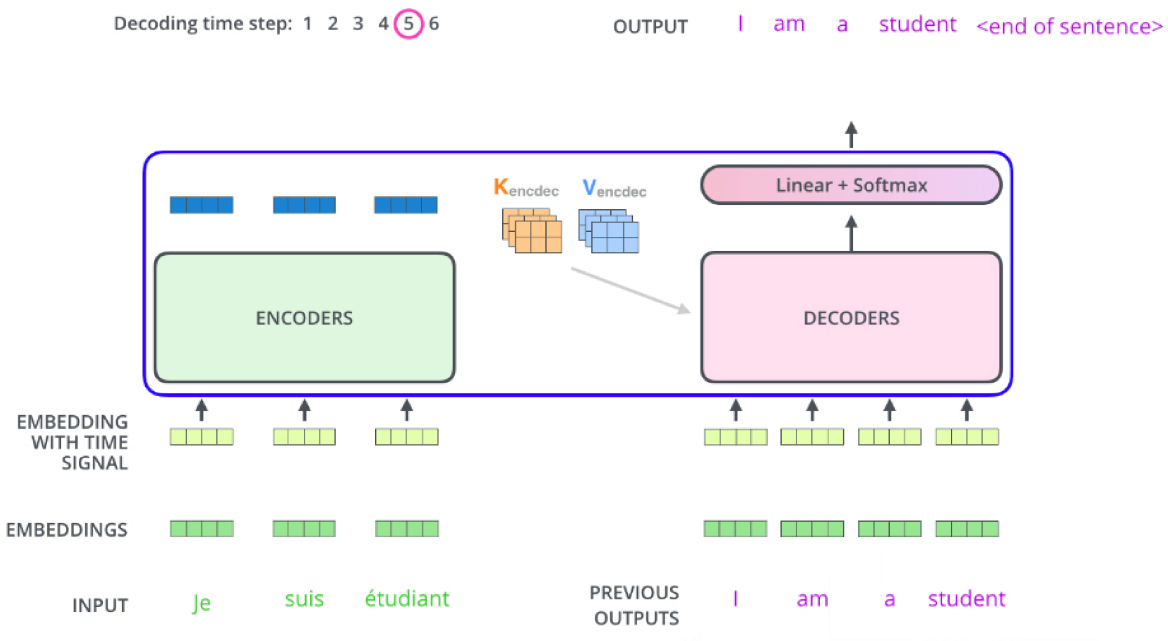


Figura 26 – *Encoder-decoder attention* (Alammar, Jay , 2019)

2.11 BERT

BERT (DEVLIN et al., 2018) é um modelo baseado na arquitetura *transformer* de Vaswani et al. (VASWANI et al., 2017). No entanto, enquanto o *transformer* originalmente tem o intuito de realizar tarefas de *machine translation*, BERT é treinado como um modelo de linguagem (DEVLIN et al., 2018). Para tal, este utiliza a camada correspondente ao *encoder* da arquitetura *transformer*. Assim como o *transformer*, o modelo BERT é baseado inteiramente no mecanismo de atenção e não apresenta nenhum tipo de característica recorrente (DEVLIN et al., 2018). A ideia fundamental por trás da utilização do BERT, cuja sigla significa *Bidirectional Encoder Representations from Transformers*, está na geração de representações robustas de palavras e *tokens* em um texto, produzindo *embeddings* contextuais à partir de representações profundas e bidirecionais. Estes *embeddings*, por sua vez, serviriam de insumo para tarefas subsequentes, de forma que o treinamento do modelo em si e a geração dos *embeddings* contextuais seriam uma maneira robusta de representar a linguagem. O modelo de linguagem pré-treinado resultante é usualmente utilizado em conjunção com uma etapa de ajuste fino para realização das tarefas subsequentes.

A ideia de extrair representações robustas, a partir de um modelo de linguagem para realização de tarefas subsequentes não é advinda do modelo BERT. Outros trabalhos anteriores exploram essa ideia, como, por exemplo em Peters et al (PETERS et al., 2018) e em Howard et al (HOWARD; RUDER, 2018). Esta mesma ideia se provou eficiente em melhorar a performance de várias tarefas de linguagem natural (DEVLIN et al., 2018; PETERS et al., 2018; HOWARD; RUDER, 2018). No entanto, o modelo BERT de Devlin et al (DEVLIN et al., 2018) é o primeiro modelo a fazer isso baseado em uma

arquitetura baseada no modelo *transformer* e, não obstante, a performance obtida com este, principalmente após o processo de ajuste fino, superou o estado da arte em tarefas importantes dentro do universo de NLP (DEVLIN et al., 2018), o que o tornou bastante proeminente.

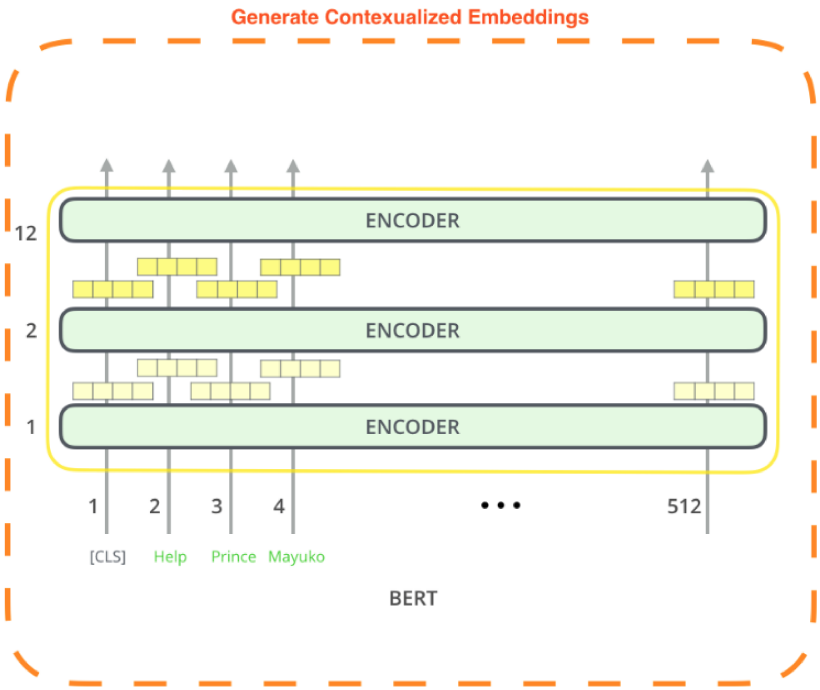


Figura 27 – O modelo BERT, visto como um *encoder stack* (Alammar, Jay , 2019)

Conceitualmente simples, porém empiricamente poderoso, o modelo BERT usa uma técnica chamada de *masked language modeling* (DEVLIN et al., 2018), que mascara determinados *tokens* da entrada com o objetivo de prever este mesmo *token*, utilizando o contexto em que este se encontra. Assim, utilizando a parte do *encoder* da arquitetura *transformer*, como ilustrado na Figura 27, e essa função objetivo para seu pré-treino, o modelo BERT é capaz de gerar um modelo de linguagem robusto que tem a propriedade importante de ser bidirecional por design, ou seja, enxergar os *tokens* antecedentes e subsequentes durante o processo de treinamento. Através do mascaramento dos *tokens*, o modelo BERT consegue fazer isso sem vazamento de informação (DEVLIN et al., 2018). O modelo de linguagem treinado pelo BERT, baseado em *masked language modeling*, é a principal contribuição deste (DEVLIN et al., 2018).

Em Devlin et al 2018 é discutido a importância da bidirecionalidade para modelos de linguagem. O autor acredita que modelos com essa característica são mais robustos que modelos que tentam realizar a mesma tarefa olhando os *tokens* em uma ordem da direita para esquerda ou o oposto ou mesmo a concatenação de ambos (DEVLIN et al., 2018). Para os autores do BERT, a capacidade de olhar de maneira bidirecional para a entrada, de maneira conjunta, é chave para um bom condicionamento do contexto que o modelo

captura (DEVLIN et al., 2018).

Apesar de contribuir majoritariamente através da proposta de um *masked language model*, o modelo BERT também faz uso de algumas outras técnicas, além de contar com alguns detalhes de implementação que o ajudam a obter sua performance elevada (DEVLIN et al., 2018). Primeiramente, seu processo de *tokenização* de texto não segue um padrão convencional de separação por palavras. Em vez disso, para o modelo BERT é utilizado um *tokenizer* conhecido como *WordPiece tokenizer* (KUDO; RICHARDSON, 2018), que faz o processo de *tokenização* utilizando sub-palavras (ou *word pieces*) quando não encontra a palavra desejada no vocabulário. Através desse processo de *tokenização* o modelo é mais robusto a erros ortográficos, além de ser menos propenso a problemas de palavras que estão fora de seu vocabulário, já que quando isto acontece ele quebra esta palavra em sub palavras até conseguir representa-la. Este tipo de *tokenização* é mostrado em Wu et al. (WU et al., 2016) para tarefas de *machine translation* onde palavras que estão fora do vocabulário são bastante problemáticas. É possível observar que, utilizando esta técnica de *tokenização* de texto, um vocabulário razoavelmente pequeno já tem um poder de representar um corpus bastante extenso.

Como entrada do modelo, BERT está adaptado para receber uma sentença única ou duas sentenças simultaneamente. Além disso, devido a falta de uma característica recorrente do modelo, dado sua arquitetura baseada no mecanismo de atenção, é necessário fornecer na entrada um *embedding* posicional, para que seja reconhecida a ordem do texto, da mesma forma como é feito para a arquitetura *transformer*. A Figura 28 mostra as entradas do modelo BERT. É possível observar que existe uma camada de *token embeddings*, que utiliza a técnica de *wordpiece tokenization*, onde podemos ver que a palavra *playing* é quebrada nos *word pieces play* e *##ing*. Além disso, à essa camada de *token embeddings*, é acrescentada uma camada de *embeddings* posicionais, descrita anteriormente e a mesma utilizada na arquitetura *transformer*, e uma camada de *segment embeddings*, que são responsáveis por dizer se está entrando uma ou mais de uma sentença para o modelo (que aceita no máximo duas). A capacidade de aceitar até duas sentenças faz com que o BERT consiga realizar, de maneira nativa, tarefas de QA (*question awnsering*), NLI (*natural language inference*) e *sentence pair classification*.

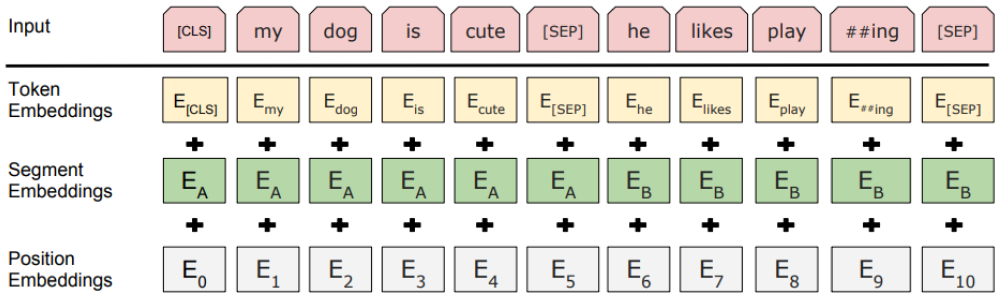


Figura 28 – Entradas do modelo BERT (DEVLIN et al., 2018)

Dado a capacidade do BERT de receber duas sentenças de entrada, durante seu processo de pré treino, além deste ser treinado como um *language model*, também é treinado para uma tarefa auxiliar de *next sentence prediction*. Essa tarefa, segundo os autores (DEVLIN et al., 2018), melhora a performance do modelo em tarefas de NLI e QA.

Como é possível observar pela Figura 28, existem alguns *tokens* especiais que são utilizados pelo BERT. Dentre eles, o *token* CLS é sempre inserido no início de cada exemplo de treinamento e, durante o processo de ajuste fino do modelo, a sua saída equivalente é utilizada para tarefas de classificação de texto, de forma que essa representa a informação agregada contida em uma sequência (DEVLIN et al., 2018). O *token* SEP é utilizado para separar duas sentenças, quando existem duas sentenças diferentes na entrada, ou para sinalizar o fim da sentença. Um outro *token*, que não está representado na Figura 28, mas que é de grande importância, é o *token* MASK.

O *token* MASK é utilizado durante o processo de pré treino do modelo, para a tarefa de *masked language modeling*. Durante esse processo, palavras aleatórias da entradas (*word pieces* aleatórios) são substituídas, com uma certa probabilidade, pelo *token* MASK, para que o modelo consiga, através da análise do contexto em que esse *token* mascarado esta presente, prever a palavra correta na saída (DEVLIN et al., 2018). Uma das desvantagens da utilização deste *token* durante o pre treino do modelo é que o mesmo nunca é visto durante o processo subsequente de ajuste fino do modelo, logo, isso gera uma discrepância entre o procedimento de pré treino e ajuste fino (DEVLIN et al., 2018). Segundo o autor, o processo de mascaramento de *tokens* é feito em 15% dos *tokens* de maneira aleatória, com um gerador aleatório que atua nos dados de treino. No entanto, como forma de mitigar a discrepância entre pré treino e ajuste fino, dentre esses 15% totais de *tokens* no qual esse procedimento é realizado, 80% são de fato substituídos pelo *token* MASK, 10% são substituídos por alguma outra palavra e outros 10% são preservados (DEVLIN et al., 2018). Na Figura 29 é mostrado o processo de *pre-training* do BERT, onde é possível notar as tarefas de *masked language modeling* e de *next sentence prediction*.

A Figura 30 ilustra os processos de ajuste fino do BERT para diversos tipos de tarefas subsequentes. Mais especificamente, na Figura 30b é possível ver o procedimento para tarefas de classificação de texto padrão. Neste tipo de tarefa, o *token* CLS, após propagado pela rede, representado como C na saída, é utilizado como insumo para se obter o rótulo da classe para tarefas de classificação. Isso é feito através da utilização de uma função *softmax* que conecta com este *token*. Para o ajuste fino do modelo para tarefas de classificação de texto, uma função de custo, baseada por exemplo na entropia cruzada binária é utilizada com a rede conectada através da camada *softmax* ao CLS token de saída (C) e o sinal é propagado para ajustar os parâmetros da rede (DEVLIN et al., 2018). Em Devlin et al. (DEVLIN et al., 2018) os hyperparametros recomendados para tarefas de ajuste fino são expostos e, segundo o autor, estes funcionam bem para a maioria das

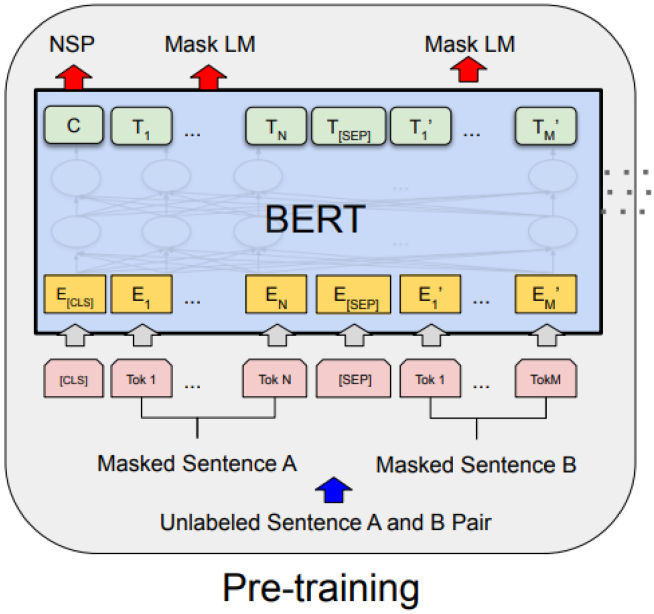


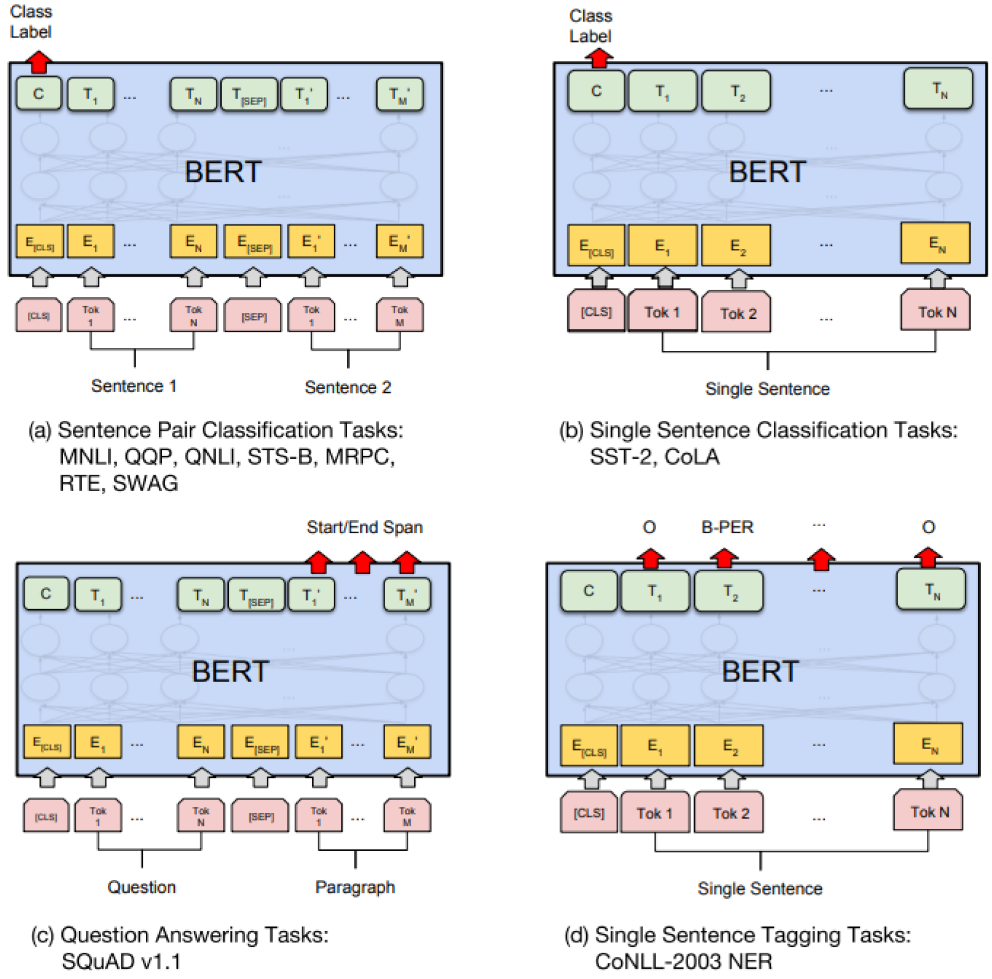
Figura 29 – BERT *pre-training* (DEVLIN et al., 2018)

tarefas (DEVLIN et al., 2018).

O *token* CLS, uma vez realizado o procedimento de ajuste fino do modelo para uma tarefa de classificação, se torna bastante representativo para a tarefa de classificação propriamente dita. No entanto, antes do procedimento de ajuste fino, a representação resultante do *token* CLS (vista como C na Figura 30) não é discriminativa para separar sentenças (DEVLIN et al., 2018). Embora isso aconteça, existem formas de utilização do modelo BERT como um extrator de características, sem que necessariamente o processo de ajuste fino do modelo tenha que ser realizado. Em Devlin et al (DEVLIN et al., 2018) estas formas são discutidas para alguns casos, embora o autor deixe evidente que a performance do modelo em tarefas subsequentes uma vez feita o ajuste fino do modelo é, na maioria dos casos, superior.

Na Figura 31 são mostradas as melhores maneiras de utilização dos *embeddings* contextuais gerados pelo modelo, quando este é usado como um extrator de características, ou seja, sem a realização do ajuste fino, (DEVLIN et al., 2018) para a tarefa de *NER (named-entity recognition)*. Neste cenário, o processo de ajuste fino garante uma performance de 96.4 de F1 score (SOKOLOVA; LAPALME, 2009), que não fica muito atrás de quando o modelo é usado como um extrator de características. No entanto, para outras tarefas, como por exemplo a tarefa de classificação de texto, a diferença de performance ao usar o modelo apenas como um extrator de características não é clara, sendo que usualmente os resultados são inferiores (DEVLIN et al., 2018).

O modelo BERT tem duas versões, *base* e *large*, com 110M e 340M de parâmetros respectivamente. Ambas versões são treinadas no mesmo corpus de texto, que consiste em uma união do *BookCorpus* e da Wikipedia em inglês, totalizando 3.2M de palavras



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figura 30 – BERT *fine-tunning* ou ajuste fino(DEVLIN et al., 2018)

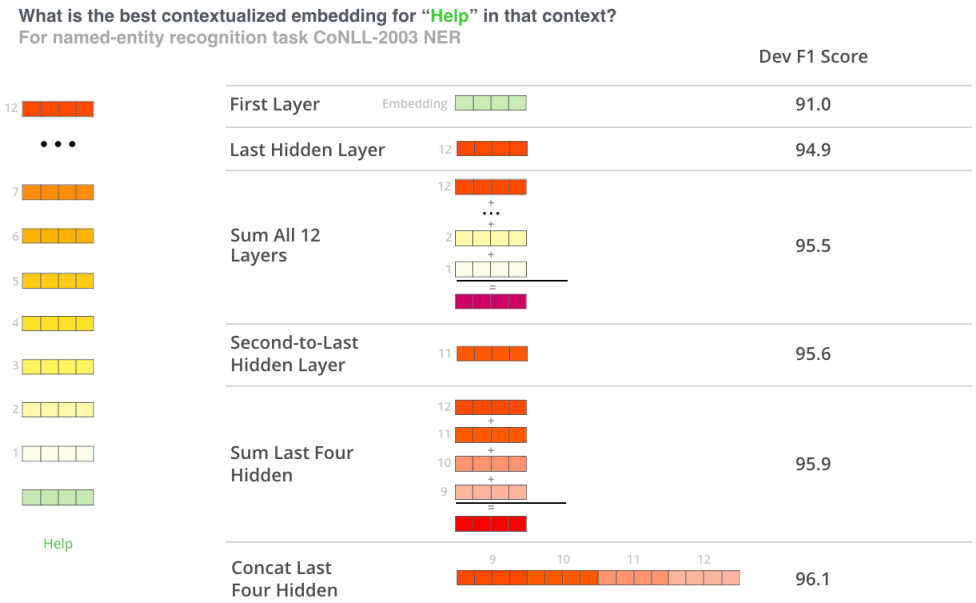


Figura 31 – Modelo BERT como um extrator de características (Alammar, Jay , 2019)

(DEVLIN et al., 2018). Fica claro, pelo número de parâmetros de ambos os modelos, que são modelos de alta capacidade. Em Devlin et al é discutido como a versão com maior número de parâmetros, BERT *large*, tem usualmente uma performance melhor para os conjuntos de dados testados pelo autor.

A capacidade de realizar a tarefa de ajuste fino pelo modelo BERT é notável, dado que, por ser um modelo de enorme capacidade, seria esperado que esse tivesse problemas de convergência ou que tivesse uma tendencia a sobreajustar aos dados. Em Devlin et al (DEVLIN et al., 2018) também é discutido como o modelo BERT tem uma probabilidade de gerar resultados degenerados após a tarefa de ajuste fino, principalmente quando o conjunto de dados é pequeno (DEVLIN et al., 2018).

A grande capacidade de BERT e seu grande número de parâmetros também faz com que o mesmo tenha um elevado custo computacional. Para executar a tarefa de ajuste fino, onde o mesmo é treinado por um número bastante reduzido de épocas, é necessária a utilização de hardware como GPUs ou TPUs, com alta capacidade de processamento paralelo. Desta forma este procedimento dura na casa de horas para conjuntos de dados de tamanho moderado. Apesar do alto custo, é notável a capacidade do modelo BERT de atingir resultados estado da arte, em diversas tarefas. Além disso, a criação de *embeddings* contextuais durante o processo de pré treino e ajuste fino do modelo fazem com que sua utilização seja benéfica e adequada para diversas tarefas de NLP, dentre elas classificação de texto, onde o contexto pode ser fundamental para o reconhecimento apropriado de padrões textuais.

2.12 XLNet

O modelo XLNet (YANG et al., 2019), assim como BERT, é um modelo de linguagem que usa uma abordagem baseada em pré treino e ajuste fino para geração de representações robustas de texto, através da geração de *embeddings* contextuais, e posteriormente, utiliza estas representações para realização de tarefas subsequentes. No entanto, embora tenha muito em comum com o BERT, o modelo XLNet trás algumas mudanças em sua arquitetura. Estas fazem com que este supere BERT em várias tarefas, para várias bases de dados, atingindo um novo estado da arte em termos de performance (YANG et al., 2019).

A maneira pela qual o modelo BERT, de Devlin et al (DEVLIN et al., 2018) utiliza o mascaramento de *tokens* durante o treinamento de seu modelo de linguagem é característico de um comportamento baseado em *autoencoder*. Mais especificamente, o modelo BERT funciona como um *denoising autoencoder* (VINCENT et al., 2008) na medida em que insere ruído em uma certa porcentagem de *tokens*, durante sua fase de pré-treino, e tenta prevê-los. O modelo XLNet muda esse paradigma. Ele faz isso através

do treinamento de seu modelo de linguagem baseando em um *framework* auto-regressivo, ao mesmo tempo que cobre lacunas que este tipo de *framework* historicamente apresentou (YANG et al., 2019). Além disso, o modelo XLNet ataca determinados pontos onde BERT é considerado limitado, de acordo com Yang et al (YANG et al., 2019).

Também baseado no *encoder* de uma arquitetura *transformer*, assim como BERT, o modelo XLNet não usa mecanismos baseados em recorrência. Além disso, este usa as mesmas entradas que BERT, em particular os *tokens* especiais (SEP e CLS), *embeddings* posicionais e o mesmo tipo de *tokenização*, baseado em *WordPieces* (WU et al., 2016).

No entanto, XLNet usa uma arquitetura *transformer* ligeiramente diferente da apresentada em Vaswani et al (VASWANI et al., 2017), que tem o nome de *Transformer-XL* (DAI et al., 2019). Essa arquitetura segue os mesmos fundamentos do *transformer* original, mas apresenta algumas melhorias que fazem com que seja capaz de capturar contextos mais longos, sendo capaz de representar sentenças maiores de forma mais eficiente. Para isso, o *Transformer-XL* usa dois mecanismos, *segment level recurrence* e *relative positional embeddings* (DAI et al., 2019). O mecanismo de *segment level recurrence* faz com que, durante o treinamento do modelo, as representações obtidas através de segmentos previamente processados sejam guardadas para serem usadas como uma espécie de contexto estendido para o próximo segmento (DAI et al., 2019). Isso estende o tamanho da dependência temporal que o modelo consegue capturar. O mecanismo de *relative positional embeddings*, por sua vez, faz com que o mecanismo anterior, baseado em recorrência, seja possível, fornecendo uma posição relativa de cada *token* processado (DAI et al., 2019). Isso é necessário pois, como na arquitetura *transformer* uma das entradas consiste em um *embedding* posicional, a utilização de um mecanismo de recorrência poderia fazer o modelo ver a mesma posição contextual mais de uma vez.

A característica mais fundamental que difere o XLNet de Yang et al do modelo BERT, no entanto, está em seu modelo de linguagem auto-regressivo (AR). Modelos de linguagem auto regressivos buscam estimar a distribuição de probabilidade de um documento. Isto é feito através da fatoração da probabilidade condicional de palavras subsequentes ou antecedentes para determinar a probabilidade da palavra atual, como pode ser visto na equação 2.15. Os problemas majoritários na utilização de um *framework* auto regressivo para esta tarefa é que este não tem a característica de ser bi-direcional. Como visto em Devlin et al (DEVLIN et al., 2018), com o modelo BERT, e reforçado por Yang et al (YANG et al., 2019), essa característica é de grande importância para um bom desempenho de um modelo de linguagem. Em contrapartida, a modelagem baseada em *autoencoder* de BERT, apesar de ser bi-direcional por *design*, assume implicitamente premissas de independência dos *tokens* mascarados, o que não ocorreria em um modelo AR, já que esse fatoriza probabilidades através da regra do produto (equação 2.15). Além disso, em BERT o uso do *token* MASK cria uma discrepância entre o processo de ajuste

fino e pré treino (DEVLIN et al., 2018), como discutido anteriormente.

$$P(x) = \prod_{t=1}^{T} P(x_t|x_{<t}) \tag{2.15a}$$

$$P(x) = \prod_{t=T}^{1} P(x_t|x_{>t}) \tag{2.15b}$$

De forma a solucionar os problemas da modelagem auto regressiva, mas ao mesmo tempo mantendo seus benefícios (como o de não assumir premissas de independência), os autores de XLNet propuseram uma formulação que tenta incorporar a característica bi-direcional de BERT em um modelo AR. Essa formulação é baseada em uma técnica que os autores chamam de *permutation language modeling* (YANG et al., 2019). Esta técnica consiste em prever o próximo *token* de uma sequência permutando a ordem em que estes são apresentados na função AR. Mais especificamente, para uma sequência x de tamanho T, existem T! maneiras de se permutar essa sequência e fazer uma fatoração de probabilidades auto regressiva seguindo a equação 2.15. Se durante esse processo o modelo compartilha parâmetros, na média ele aprenderá informações de contextos bi-direcionais (YANG et al., 2019). Sendo Z_t um conjunto das permutações possíveis de uma sequência x de tamanho T, a equação 2.16 expressa, de maneira reduzida, a formulação usada para o *permutation language modeling* (YANG et al., 2019), dado uma ordem de fatoração z.

$$max_{\Theta} \quad E_{z \backsim Z_T}\big[\sum_{t=1}^{T} logp_{\theta}(x_{zt}|x_{z<t})\big] \tag{2.16}$$

A Figura 32 ilustra o procedimento de *permutation language modeling*, onde x_3 é previsto com diferentes ordens de fatoração.

Embora a técnica de *permutation language modeling* mude a ordem com que os *tokens* são alimentados ao modelo, este ainda faz uso de *embeddings* posicionais que retem a informação de posição dentro da sequência analisada, portanto a ordem da sequência não é quebrada.

A figura 33 ilustra as diferenças conceituais entre os treinamentos dos modelos de linguagem de BERT e XLNet, onde fica explicita a diferença entre o *masked language model* e *permutation language modeling*. Embora este último tenha benefícios teóricos em relação ao primeiro, como discutido anteriormente, também é pontuado em Yang et al (YANG et al., 2019) que o *permutation language modeling* dá origem a um problema mais complexo de otimização na medida em que cobre mais dependências e contém um maior número efetivo de sinais de treinamento (YANG et al., 2019). Logo, as vantagens deste modelo tem um *trade-off* em relação à dificuldade de seu treinamento e ajuste fino, de um ponto de vista teórico.

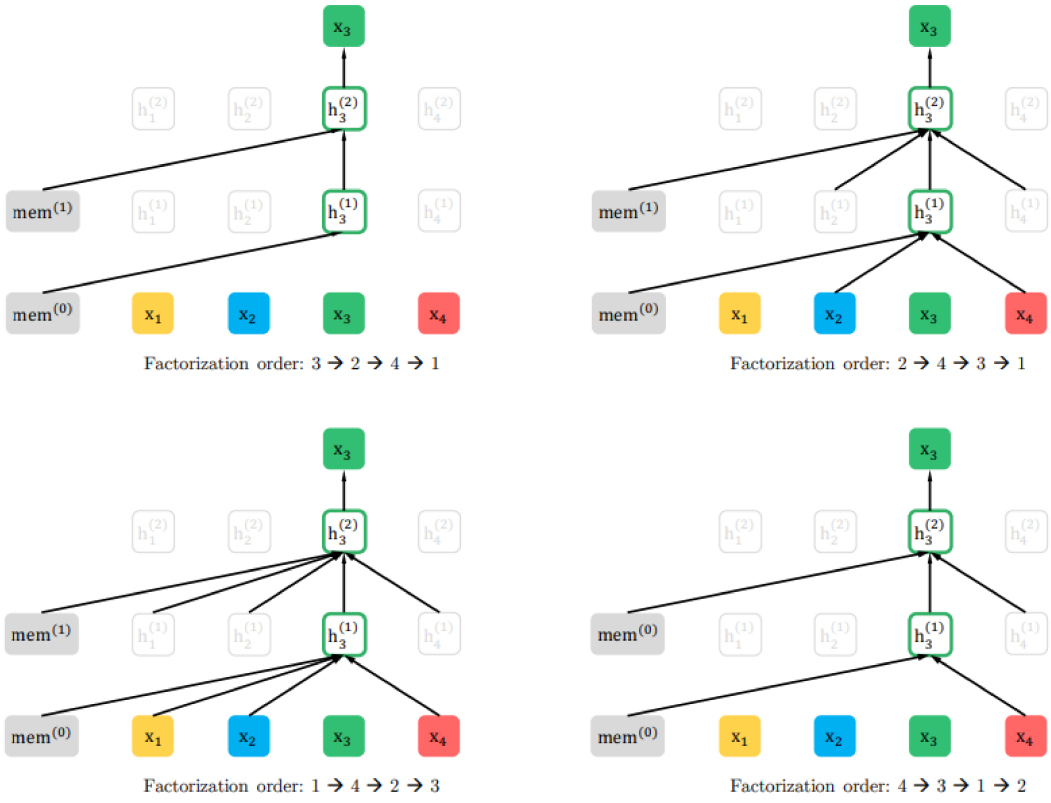


Figura 32 – *Permutation Language Modeling* (YANG et al., 2019)

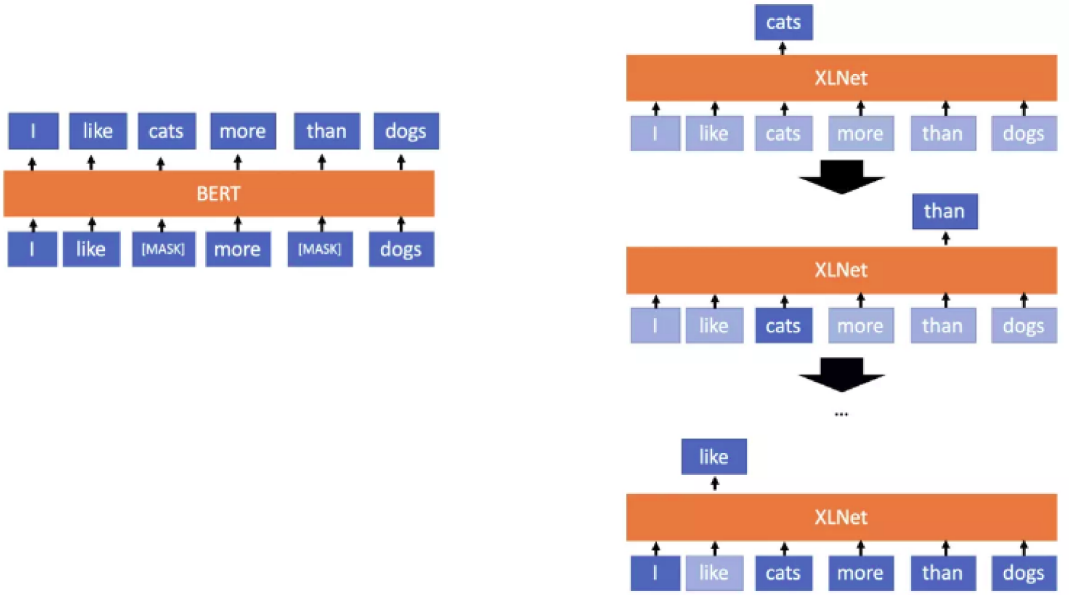


Figura 33 – Diferença entre o modelo auto regressivo de XLNet e o modelo baseado em *autoencoder* de BERT

Em relação às demais características, o modelo XLNet é idêntico ao modelo BERT, também contendo duas versões, *base* e *large*, com 110M e 340M de parâmetros respectivamente. Um detalhe adicional é que o modelo XLNet foi treinado em um corpus mais extenso de texto (YANG et al., 2019), em sua versão que atinge resultados estado da arte. Entretanto, segundo seu autor, através de testes comparativos com BERT, usando o mesmo conjunto de dados para seu pré-treino, o XLNet ainda obtém performance superior a BERT na maior parte das tarefas de NLP (YANG et al., 2019).

2.13 Embedddings contextuais

Embora arquiteturalmente diferentes, BERT e XLNet são dois modelos de linguagem capazes de derivar representações robustas de dados textuais através da criação de *embeddings contextuais*. Estes por sua vez incorporam, aos *tokens* que representam, informações pertinentes ao contexto das palavras que os acompanham. Através desta característica, os *embeddings* contextuais conseguem lidar com palavras polissêmicas e capturar melhor algumas outras características da linguagem como por exemplo sarcasmo. A palavra banco, por exemplo, pode assumir diferentes sentidos dependendo do contexto em que se encontra, podendo significar banco instituição financeira ou um banco para sentar, o que fica só claro através do contexto onde esta inserida.

Os ganhos em termos de poder representacional advindo da utilização desta técnica são de grande importância para o cenário de NLP. No geral, tarefas de NLP tem um alto nível de complexidade visto que existe muita ambiguidade e flexibilidade no uso da linguagem (JURAFSKY; MARTIN, 2019). Uma representação mais robusta é capaz de melhorar o poder de generalização para tarefas subsequentes além de reduzir a dubiedade na interpretação linguística. Não obstante, como discutido em Goodfellow et al (GOODFELLOW; BENGIO; COURVILLE, 2016), algoritmos de aprendizado de máquina dependem fortemente da representação dos dados, sendo que uma boa representação, principalmente no cenário de dados não estruturados (como dados textuais por exemplo), é chave para que um modelo/classificador consiga extrair correlações e padrões que modelem apropriadamente a distribuição dos dados.

Em contrapartida aos benefícios e ganhos em termos de poder de representação, os *embeddings* contextuais, diferentemente de seus predecessores, geram representações vetoriais diferentes para cada palavra que representam, dado que os contextos em que esta estão inseridas mudam. Isso faz com que esse tipo de *embedding* não possa ser facilmente utilizado através de uma tabela de consulta, onde cada palavra está associada a um único vetor, como é feito com modelos do tipo word2vec (MIKOLOV et al., 2013) ou Glove (PENNINGTON; SOCHER; MANNING, 2014). Em vez disso, os *embeddings* contextuais necessitam ser gerados de maneira dinâmica, pois são dependentes do contexto da frase.

Isso por sua vez significa que para a geração destes é necessário uma etapa de *forward propagation* pela arquitetura responsável por sua geração, seguida da extração dos estados escondidos do modelo. Este procedimento é obviamente mais custoso que o anterior e impõe um *trade-off* em relação a um possível ganho de performance *versus* custo computacional. Para etapa da extração dos *embeddings* contextuais à partir da camada escondida, pode ser adotado algum dos procedimentos ilustrados anteriormente através da Figura 31, sendo que este procedimento é igual tanto para os modelos BERT quanto XLNet.

3 Metodologia

Neste capítulo serão detalhadas as bases de dados e os experimentos propostos neste trabalho. Estes visam verificar a robustez dos métodos que geram *embeddings* contextuais, BERT e XLNet em um cenário de baixa volumetria de dados, quando usados para tarefas de classificação de texto. O comportamento destes, neste cenário, é pouco explorado na literatura, especialmente durante a tarefa de ajuste fino do modelo. O risco de sobreajuste durante esta etapa é alto porém, a possível capacidade destes modelos de generalizar com poucos dados é algo de grande valor prático. Outro aspecto de grande importância prática consiste na mensuração da diferença de performance da utilização destes modelos como extratores de características em relação à quando são ajustados via ajuste fino, visto que a primeira prática requer esforço computacional consideravelmente menor. Além disso, os experimentos também buscam verificar os benefícios da utilização de *embeddings* contextuais em relação a formas mais tradicionais de *embeddings* como Glove (PENNINGTON; SOCHER; MANNING, 2014). Por fim, a função de custo de perda de articulação (*hinge-loss*) será testada durante o procedimento de ajuste fino para averiguar se existe um benefício na sua utilização no cenário de poucos dados. Todas as comparações utilizarão a métrica de acurácia, visto que os problemas atacados tem todos duas classes e todos os conjuntos de dados utilizados são balanceados.

3.1 Experimentos

A fim de testar os métodos baseados em *embeddings* contextuais, em um cenário de escassez de dados, foram propostos 4 experimentos, que serão detalhados nas seções subsequentes.

3.1.1 Verificação da capacidade de transferência de aprendizado dos métodos BERT e XLNet via ajuste fino

Neste experimento, os modelos BERT e XLNet serão testados em relação a sua capacidade de realizar a tarefa de transferência de aprendizado via ajsute fino. Nesta etapa, os dois modelos serão utilizados para tarefas de classificação de texto em um cenário onde os conjuntos de dados utilizados vão sendo amostrados com volumetria progressivamente menor de dados, a fim de se verificar a característica da curva de transferência de aprendizado destes modelos. Para isso, estes também serão comparados com *baselines* baseados em uma representação mais simples de texto, que utilizam BOW ou TF-IDF seguidos de um classificador SVM. Os *baselines* escolhidos representam uma

maneira clássica e consolidada de realizar a tarefa de classificação de texto (JOACHIMS, 1998). Essa comparação também visa quantificar a diferença dos modelos estudados, baseados em técnicas de *deep learning*, dos modelos baseados em aprendizado estatístico convencional, onde a representação dos dados não é derivada à partir do próprio modelo (GOODFELLOW; BENGIO; COURVILLE, 2016), ou seja, que não realizam *representation learning*.

Os conjuntos de dados utilizados e as volumetrias amostradas dos mesmos serão detalhadas na Seção 3.2. No entanto, vale ressaltar que todos os modelos são avaliados no mesmo conjunto de teste, para assegurar uma comparação justa, e o mesmo tipo de preprocessamento é realizado aos dados antes de todos os experimentos, que consiste em remoção de acentuação, limpeza de caracteres especiais e transformação dos textos pra caixa baixa.

Os modelos utilizados neste experimento são o BERT e XLNet em suas versões *base* e *large* (com 110M e 340M de parâmetros respectivamente) e 2 *baselines* que utilizam BOW + SVM e TF-IDF + SVM para a tarefa de classificação do texto. Os *baselines* foram treinados em um processo de validação cruzada com 3 *folds* e *grid search* para seleção dos melhores hiper-parâmetros. A *tokenização* realizada para estes métodos foi a nível caractere e também teve seus hiper-parâmetros otimizados via *grid-search cross-validadion*. Os modelos BERT e XLNet utilizados foram os modelos *pre-treinados* disponibilizados por seus autores (DEVLIN et al., 2018; YANG et al., 2019) e, para a tarefa de ajuste fino, foram utilizados os hiper-parâmetros recomendados de *learning rate 2e-5 por 3 a 5 épocas)*. Além disso, foi utilizado *batch size* de 32 e máximo tamanho de sentença de 256, devido a restrições do *hardware* disponível (que não impactou os resultados do experimento). Para os modelos baseados em *embedding* contextuais não foi realizada uma busca de hiperparametros extensiva, dado que o objetivo do experimento não é extrair a maior performance possível dos métodos, mas sim observar seu comportamento em cenários de baixa volumetria de dados e sua capacidade de transferência de aprendizado via ajuste fino. Os modelos *baseline* foram executados em CPU, enquanto os modelos BERT e XLNet foram executados em GPU para as versões *base* com 110M de parâmetros e em TPU para as versões *large* com 340M de parâmetros.

Cada combinação de modelo, conjunto de dados e fração amostrada do conjunto de dados foi executada por 3 vezes a fim de garantir maior validade estatística para o experimento. Execuções de BERT ou XLNet que produziram resultados degenerados e não convergiram (que é um cenário possível e descrito em Devlin et al (DEVLIN et al., 2018)) não foram descartadas. Ao invés disso, a melhor performance de cada um dos modelos foi computada, junto com o desvio padrão destas execuções, que ajuda na quantificação deste fenômeno. A escolha em guardar e apresentar os melhores resultados de cada modelo também ajuda a desenviesar possíveis problemas de amostragem que

poderiam comprometer os resultados de algum modelo durante os experimentos.

3.1.2 Comparação do ajuste fino versus uso dos modelos como extratores de características

A partir dos resultados do experimento anterior, o modelo baseado em *embeddings* contextuais de melhor performance no cenário de baixa volumetria de dados, o BERT *base* (versão com 110M de parâmetros) é testado neste experimento. O objetivo deste experimento é verificar se há diferença de performance deste modelo quando realizado seu ajuste fino em comparação a quando usado como um extrator de características para a tarefa de classificação com poucos dados.

Para tal, os mesmos hiper-parâmetros da subseção anterior serão utilizados para o treinamento através de ajuste fino e nos mesmos conjuntos de dados. Entretanto, será apenas considerado o cenário de maior escassez de dados, com apenas 250 exemplos para treinamento. Novamente, a versão pré-treinada disponibilizada pelo autor será utilizada como ponto de partida.

A fim de compará-lo em sua utilização como um extrator de características, os *embeddings* contextuais derivados deste método, advindos apenas de seu treinamento prévio como um modelo de linguagem, serão extraídos e utilizados de insumo para classificadores baseados em arquiteturas LSTM e CNN.

Para a parte da extração dos *embeddings* contextuais, o procedimento de somar os estados das quatro ultimas camadas escondidas do modelo, ilustrado na Figura 31, será utilizado. Este tem a melhor performance dentre aqueles que mantém um tamanho de representação padrão e, portanto, tem uma relação de performance/custo computacional razoável. Isso é desejado pois os experimentos, para cada conjunto de dados e método, serão executados 30 vezes para realização de testes de hipóteses com um melhor poder estatístico.

Em relação aos classificadores usados com os *embeddings* extraídos, as redes LSTM e CNN foram escolhidas pois aplicam mecanismos de recorrência e convolução, e conseguem competir com o mecanismo de atenção utilizado por BERT. Isso ocorre devido ao fato de que estas arquiteturas são apropriadas para lidar com modelos de sequência (GOODFELLOW; BENGIO; COURVILLE, 2016). As redes LSTM e CNN foram testadas utilizando um processo manual de escolha de hiper parâmetros, focando no ajuste da taxa de aprendizado e número de épocas. Para a LSTM foi escolhida uma arquitetura bi-LSTM com 64 neurônios em uma única camada, e para a CNN uma arquitetura que faz uso de uma convolução 1D com 128 neurônios, kernel de tamanho 5, seguida de um operador de *max-pooling 1D* e uma camada escondida densa de 10 neurônios. Como saída da LSTM e CNN foi usado um neurônio com função sigmoidal, enquanto as camadas escondidas

usaram a função de ativação *relu*. A função de custo para ambos os modelos foi entropia binária cruzada.

Quando usados como características de treinamento, os *embeddings* contextuais não tiveram suas representações atualizadas com o treinamento das redes LSTM e CNN. Isso se deve ao fato de que o objetivo deste experimento é avaliar seu poder representacional advindo do pré treino do modelo que os originou, em comparação a quando estes são ajustados via ajuste fino do modelo original.

3.1.3 Comparação de embeddings contextuais com embeddings tradicionais

Neste experimento, usando as mesmas arquiteturas LSTM e CNN discutidas na subseção anterior, são comparadas as performances destes modelos quando recebem como entrada *embeddings* contextuais *versus embeddings* tradicionais. Para isso, as representações advindas do modelo BERT *base*, extraídas da mesma maneira do experimento anterior, serão comparadas com uma representação Glove (PENNINGTON; SOCHER; MANNING, 2014), quando usadas como insumo para estes modelos.

Da mesma maneira que no experimento anterior, essas representações não serão atualizadas durante o treinamento. Isso se deve ao fato de que o objetivo é avaliar o poder representacional destas diferentes formas de *embeddings* sem ajuste das representações em si via treinamento, o que poderia comprometer os resultados do experimento. Também de maneira análoga à anterior, serão utilizados 250 exemplos para treinamento dos modelos, visto que o objetivo é compará-los em um cenário de baixa volumetria de dados a fim de testar a capacidade de generalização e representação dos mesmos. A mesma parametrização e rotina de treinamento para os modelos LSTM e CNN, da seção anterior, são utilizadas neste experimento.

3.1.4 Uso de função de perda de articulação como função de custo durante ajuste fino

Este experimento visa comparar duas funções de custo durante o processo de ajuste fino, sendo estas a função de custo tradicional, baseada em entropia binária cruzada, *versus* uma função baseada em função de perda de articulação. A fundamentação teórica por trás deste experimento é verificar se a função de perda de articulação, que produz um efeito de gerar uma margem larga de separação entre os dados, ajuda na capacidade de generalização do modelo quando feito o ajuste fino, no cenário de escassez de dados. Para cada função de custo o experimento foi executado 30 vezes, de forma a garantir um maior poder para os testes estatísticos utilizados para avaliação dos resultados.

3.2 Conjuntos de dados

Os conjuntos de dados utilizados nos experimentos são mostrados na Tabela 1.

Tabela 1 – Conjuntos de dados usados nos experimentos

Dataset	N Classes	Tamanho médio da sentença	Tamanho train set	Tamanho test set	Subamostragem conjunto de treino
IMDb	2	230	25k	25k	[500 1k 2.5k 5k 10k]
CR	2	19	3016	755	[250 500 1k 2k 3k]
SST-2	2	19	6920	1821	[250 500 1k 1.5k 3k]
MPQA	2	3	8482	2121	[250 500 1k 2k 4k]

Estes são conjuntos de dados usualmente usados para a tarefa de classificação de texto. Eles foram escolhidos pois tem características interessantes quando comparados uns com os outros. Em particular, estes tem um tamanho médio de sentença variado. Essa escolha visa explorar as características dos *embeddings* contextuais e modelos baseados na arquitetura *transformer* na medida em que uma sentença maior é um indicativo de maior riqueza em termos de contexto do documento.

Todos os conjuntos de dados escolhidos são balanceados e tem 2 classes portanto, como métrica de avaliação é utilizada acurácia. Na Tabela 1 é possível ver a divisão escolhida para os conjuntos de treino e teste, assim como os tamanhos originais de cada conjunto de dados.

Também na Tabela 1 é possível ver as subamostragens realizadas em cada conjunto de dados para realização do experimento de verificação da capacidade de transferência de aprendizado dos métodos BERT e XLNet via ajuste fino. Embora a volumetria de dados utilizados para o treinamento do modelo varie de acordo com cada experimento, conforme descrito na seção anterior, os conjuntos de teste são mantidos fixos durante todos os procedimentos.

Todas as bases utilizadas, embora enquadradas dentro da tarefa de classificação de texto, realizam a sub-tarefa de análise de sentimento. Esta por sua vez é uma das aplicações mais comuns para modelos de classificação de texto a nível documento/sentença. Além disso, esta tarefa é de grande valor prático na indústria, o que corrobora sua utilização.

3.3 Recursos Computacionais

Nesta seção serão descritos os recursos computacionais necessários para a realização dos experimentos propostos e, ao final, será disponibilizado o link para o repositório que contem o código e os conjuntos de dados utilizados, para que os experimentos possam ser reproduzidos, caso desejado.

Os experimentos utilizaram *hardwares* diferentes para sua execução de acordo com o custo computacional de cada método explorado. Os modelos BERT e XLNet, devido ao seu elevado custo computacional, foram executados utilizando GPUs e TPUs. Mais especificamente, para as versões large de ambos os métodos foi utilizado o ambiente nuvem Google Colaboratory, onde uma TPU modelo TPUv2 com 8 núcleos de processamento foi utilizada. Para as versões base, uma GPU tesla K80 foi utilizada, também através do ambiente Google Colaboratory. Ao se executar experimentos usando TPUs é necessário conectar a um *bucket* Google Cloud que seja capaz de armazenar os parâmetros e outros metadados associados aos modelos treinados, na forma de *checkpoints*. O gargalo para o treinamento destes modelos está na quantidade de memória gráfica disponível para as GPUs/TPUs. Para o treinamento com o *batch size* utilizado (de 32) essa memória deve estar na casa de 12-16Gb, dependendo das características do conjunto de dados. A placa Tesla K80 e a TPUv2 atendem estas especificações. As características da CPU e memória RAM importam menos que a do acelerador de vídeo escolhido, sendo que estes devem apenas serem capazes de carregar os conjuntos de dados na memória e realizar algum pré-processamento mínimo. Em comparação, as GPUs e TPUs devem ter a capacidade de armazenamento dos 110M ou 340M de parâmetros dos modelos mais os *batches* de dados e portanto, devem ter uma memória ampla disponível.

Os demais experimentos utilizaram 4 vCPUs e 16GB de memória RAM. Durante a utilização do modelo BERT como extrator de características, é requirida uma quantidade elevada de memória RAM, dado que os vetores de *embeddings* tem dimensionalidade razoável, que gera custo de armazenamento em memória crescente com o tamanho do conjunto de dados e do comprimento dos documentos deste em termos do número de *tokens*.

Os conjuntos de dados e os códigos utilizados para a realização dos experimentos propostos se encontram no repositório a seguir: https://github.com/felipefreitas93/mestrado_ppgee

4 Resultados e análises

Neste capítulo, os resultados dos experimentos propostos na seção anterior são expostos e analisados. Em seguida, algumas ponderações gerais são feitas a respeito do resultado coletivo dos experimentos.

4.1 Resultados de cada experimento

4.1.1 Verificação da capacidade de transferência de aprendizado dos métodos BERT e XLNet via ajuste fino

As figuras 34 e 35 e a Tabela 2 apresentam os resultados do primeiro experimento, de verificação da capacidade de transferência de aprendizado dos métodos BERT e XLNet via ajuste fino.

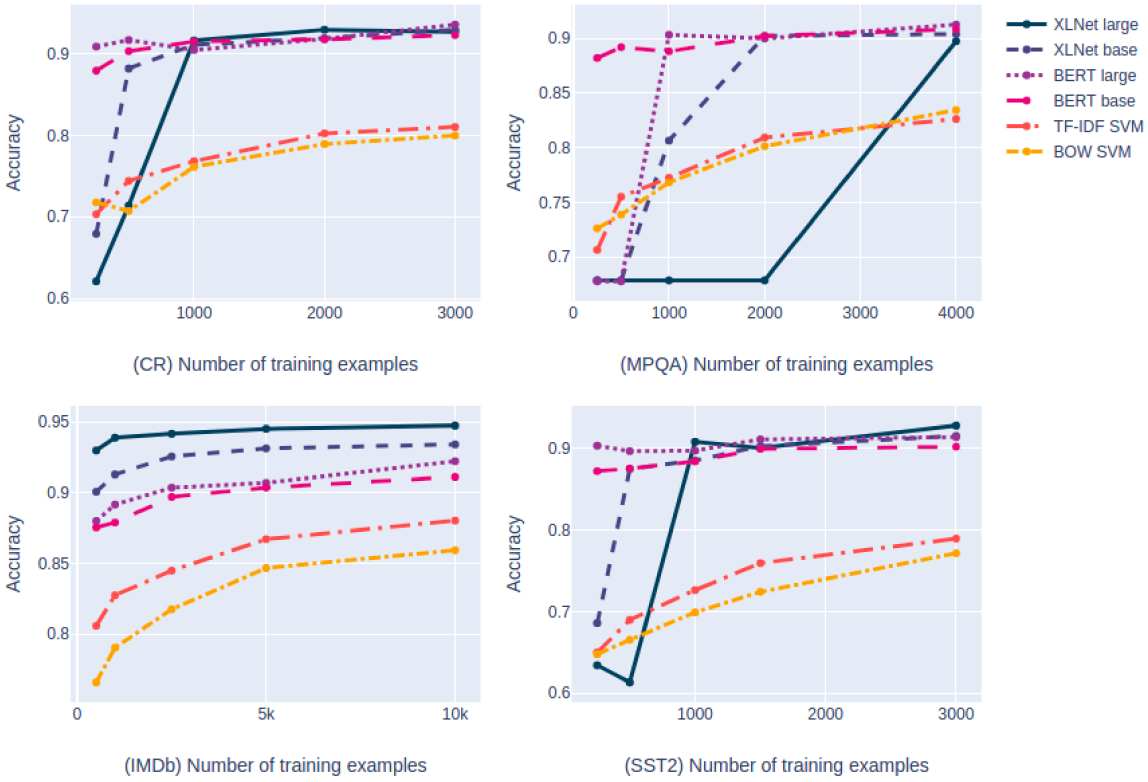


Figura 34 – Acurácia dos modelos nos conjuntos de dados usando volumes progressivamente menores de dados

Analisando a Figura 34 é possível observar que os modelos baseados em *embeddings* contextuais são capazes de ter boa performance em um cenário de baixa disponibilidade

Tabela 2 – Resultados do experimento de ajuste fino com volumes de dados progressivamente menores - Melhor acurácia e desvio padrão

CR	**Number of examples**				
Model	**250**	**500**	**1k**	**2k**	**3k**
BERT	0.8794	0.9033	0.9152	0.9178	0.9231
base	0.0022	0.0112	0.0060	0.0026	0.0095
BERT	**0.9089**	**0.9171**	0.9048	0.9184	**0.9361**
large	0.1515	0.0133	0.1631	0.0142	0.0027
XLNet	0.6794	0.8821	0.9112	0.9192	0.9298
base	0.0440	0.0159	0.0084	0.0037	0.0028
XLNet	0.6211	0.7139	**0.9165**	**0.9298**	0.9271
large	0.0018	0.0674	0.0327	0.0056	0.0037
BOW	0.7178	0.7072	0.7615	0.7894	0.8000
SVM	0.0200	0.0272	0.0235	0.0249	0.0072
TFIDF	0.7033	0.7443	0.7682	0.8026	0.8105
SVM	0.0199	0.0173	0.0094	0.0144	0.0058
MPQA	**Number of samples**				
Model	**250**	**500**	**1k**	**2k**	**4k**
BERT	**0.8821**	**0.8920**	0.8882	**0.9024**	0.9085
base	0.0043	0.0046	0.0043	0.0053	0.0013
BERT	0.6780	0.6780	**0.9034**	0.9000	**0.9128**
large	0.0000	0.0000	0.0944	0.1570	0.0003
XLNet	0.6789	0.6789	0.8066	0.9024	0.9042
base	0.0000	0.0000	0.0066	0.0020	0.0053
XLNet	0.6789	0.6789	0.6789	0.6789	0.8976
large	0.0000	0.0000	0.0000	0.0000	0.0333
BOW	0.7265	0.7388	0.7685	0.8015	0.8345
SVM	0.0191	0.0097	0.0070	0.0035	0.0095
TFIDF	0.7067	0.7553	0.7727	0.8095	0.8264
SVM	0.0115	0.0123	0.0074	0.0195	0.0038
IMDb	**Number of samples**				
Model	**500**	**1k**	**2.5k**	**5k**	**10k**
BERT	0.8754	0.8789	0.8970	0.9036	0.9111
base	0.0038	0.0034	0.0033	0.0016	0.0015
BERT	0.8798	0.8916	0.9036	0.9070	0.9222
large	0.0066	0.0005	0.0012	0.0052	0.0014
XLNet	0.9007	0.9129	0.9256	0.9313	0.9342
base	0.0097	0.0058	0.0023	0.0009	0.0014
XLNet	**0.9286**	**0.9412**	**0.9440**	**0.9452**	**0.9474**
large	0.0071	0.0011	0.0013	0.0003	0.0004
BOW	0.7659	0.7904	0.8175	0.8467	0.8593
SVM	0.0066	0.0069	0.0038	0.0054	0.0033
TFIDF	0.8058	0.8275	0.8448	0.8672	0.8802
SVM	0.0109	0.0014	0.0098	0.0021	0.0017
SST2	**Number of samples**				
Model	**250**	**500**	**1000**	**1500**	**3000**
BERT	0.8720	0.8753	0.8841	0.8995	0.9022
base	0.0209	0.0248	0.0027	0.0081	0.0023
BERT	**0.9034**	**0.8967**	0.8973	**0.9112**	0.9140
large	0.0098	0.0015	0.0005	0.0323	0.0007
XLNet	0.6858	0.8747	0.8852	0.9028	0.9154
base	0.0221	0.0069	0.0112	0.0023	0.0007
XLNet	0.6342	0.6133	**0.9082**	0.9006	**0.9280**
large	0.0935	0.0710	0.0896	0.0291	0.0081
BOW	0.6479	0.6655	0.6990	0.7243	0.7715
SVM	0.0198	0.0144	0.0190	0.0214	0.0092
TFIDF	0.6501	0.6897	0.7265	0.7594	0.7896
SVM	0.0338	0.0129	0.0138	0.0121	0.0123

Figura 35 – Diferença de performance entre o melhor modelo em um cenário de poucos dados, BERT base, e o *baseline* TF-IDF + SVM

de dados e seus resultados são, na maioria dos casos, melhores que os *baselines*.

É notável que existe um efeito positivo em realizar o ajuste fino com maior quantidade de dados, no entanto, a utilização de volumetria relativamente pequenas de dados já faz com que os métodos analisados sejam capazes de ter performance igual ou superiores a modelos clássicos, baseados em aprendizado estatístico e treinados com quantidade substancialmente maior de dados.

Esses resultados são indícios da capacidade dos modelos profundos, baseados na arquitetura *transformer*, de terem boa capacidade de generalização, mesmo em um cenário de baixa volumetria de dados. Os métodos clássicos, comparados neste experimento, usualmente sofrem do fenômeno conhecido como maldição de dimensionalidade, pois tem uma representação muito esparsa (GOODFELLOW; BENGIO; COURVILLE, 2016). Além disso, como estes derivam suas representações do modelo de *bag of words*, são muito dependentes do conjunto de treinamento utilizado, sendo ainda menos capazes de generalizar. Ademais, como esta classe de modelo não é capaz de realizar pré treino e transferência de aprendizado, o treinamento destes com poucos dados é usualmente pouco proveitoso, pois estes aprendem um número bastante limitado de padrões. Em contraste, a capacidade de aprender uma representação flexível dos dados de texto através

dos *embeddings* contextuais tem efeito bastante positivo no cenário adverso de escassez de dados. Além disso, no processo de pré treino, feito por esses modelos de linguagem, é seguro dizer que são aprendidos diversos *manifolds* dos dados representados, logo é cabível que de fato esses retenham boa capacidade de generalização, advinda do treinamento anterior.

Ainda é notável que estes sejam capazes de performar bem no cenário de poucos dados devido a seu grande número de parâmetros e baixa massa de dados disponíveis para ajuste dos modelos. Via processo de ajuste fino, o problema de otimização a ser resolvido durante etapas de aprendizado destes modelos é de alto grau de dificuldade. O fato de um modelo como o BERT base, de 110M de parâmetros, performar bem durante seu ajuste com 250 exemplos de treinamento, é um resultado que chama a atenção.

Como pode ser visto através da Figura 34, o modelo BERT base, tem boa performance em todos os conjuntos de dados e todas as volumetrias de dados. Em contrapartida, o modelo BERT na sua versão large, não consegue o mesmo feito em alguns casos de menor volume de dados. As diferenças entre os dois se dão em relação ao número de parâmetros (340M para a versão large e 110M para a versão base), enquanto que o restante, incluindo a arquitetura do modelo, é igual.

Em Devlin et al (DEVLIN et al., 2018), é discutido como o modelo BERT tem uma probabilidade de gerar resultados degenerados quando o conjunto de dados é menor. Como é possível ver, BERT large sofre mais que o BERT base com poucos dados, o que leva a conclusão de que sua maior capacidade o deixa mais propenso para sofrer de fenômenos de sobreajsute ou de geração de resultados degenerados, dado a maior dificuldade em seu processo de otimização. Embora seja difícil isolar estes dois fatores, vale ressaltar o fato de que a confiabilidade do modelo é um fato importante ao se considerar sua performance, logo, a característica de mais comumente gerar resultados degenerados torna BERT large menos confiável, da mesma forma que o fato de este ser mais propenso ao sobreajuste.

Embora alguns modelos sejam menos confiáveis em cenários extremos de carência de dados, todos tem em comum o fato de serem modelos baseados na arquitetura *transformer* e também de terem um objetivo comum de representar a linguagem através da construção de um modelo de linguagem que gera representações robustas e contextuais. Além disso, neste experimento em específico, estes modelos são utilizados em um cenário de ajuste fino. Como é possível ver através dos experimentos, essa arquitetura de modelos, representada por BERT base, é capaz de ser muito eficiente em cenários com baixa disponibilidade de dados como um todo. Em particular, no conjunto de dados MPQA, que tem um tamanho de sentença médio de 3 palavras (*tokens*), ainda são visíveis os benefícios da utilização destas arquiteturas e do uso de representações contextuais. Isso é particularmente interessante pois este é um cenário relativamente desfavorável para representações na forma de *embeddings* contextuais, dado que existe pouco contexto disponível em sentenças curtas

e, mesmo assim, esses métodos ainda tem performance superior aos *baselines*. Isso pode ser devido ao pré treino destes modelos, que os garante boa capacidade de generalização. Isso atua em conjunto com a capacidade de serem levemente perturbados durante o processo de ajuste fino, de forma a se adaptarem de maneira muito robusta para a tarefa subsequente de classificação.

O modelo XLNet também é relativamente robusto para lidar com um cenário de poucos dados. Como discutido por Tang et al (YANG et al., 2019), o modelo XLNet é capaz de cobrir mais dependências que o modelo BERT, dado sua arquitetura e formulação auto-regressiva (YANG et al., 2019). De forma notável, no conjunto de dados IMDb, que tem o comprimento médio de sentença consideravelmente maior que os demais, o XLNet apresenta os melhores resultados, com uma boa margem. Entretanto, o *permutation language modeling*, como discutido pelo autor (YANG et al., 2019), é um problema de otimização relativamente mais difícil que o de BERT, apesar de suas vantagens. Isso fica evidente na medida em que, em cenários de menor disponibilidade de dados, o modelo XLNet sofre mais que os demais. Através dos resultados empíricos deste experimento, é possível notar, a partir do comportamento teórico previsto por Yang et al (YANG et al., 2019), que ao cobrir mais dependências e conter mais sinais que BERT, o modelo XLNet apresenta um deterioramento da performance mais acentuado em um cenário progressivo de baixa disponibilidade de dados. Isso é um *trade-off* observável na medida em que a complexidade adicional de sua formulação, ao mesmo tempo que traz benefícios em alguns cenários com volumetria significativa de dados, se mostra desfavorável no cenário onde este não é o caso. Os piores resultados deste modelo no cenário de ausência de dados podem ser vistos nos demais conjuntos da dados além do IMDb. Em especial, no conjunto de dados MPQA, onde o comprimento médio da sentença é o menor (3 *tokens*), a degradação da performance é mais acentuada. Esses argumentos não são suficientes para afirmar que o modelo XLNet não é de utilização favorável em um cenário de poucos dados, no entanto, é possível afirmar que este não é tão robusto quanto o modelo BERT.

Em particular, baseado nos resultados obtidos, o modelo BERT base foi o modelo de melhor performance durante este experimento quando é analisado o cenário como um todo. Este manteve um desempenho e capacidade de transferência de aprendizado via ajuste fino em todos os volumes de dados analisados e, quando comparado sua performance com os demais métodos, em cenários de maior volumetria, este apresenta resultados bem competitivos. Tendo isso em mente, a figura 35 compara a performance deste modelo com o *baseline* composto de TF-IDF e SVM. Fica evidente sua capacidade de transferência de aprendizado, mesmo em cenários de muito poucos dados, e também é notável a diferença considerável de performance deste em relação ao *baseline*, em todos os casos.

Portanto, como resultado do experimento é bem visível que estes modelos, liderados pelo modelo BERT base, são no geral capazes de realizar a tarefa de transferência de

aprendizado via ajuste fino em um cenário de poucos dados. Olhando as Figuras 34 e 35 é notável o quão rápido as curvas de transferência de aprendizado destes modelos saturam, o que reforça que são modelos capazes de performar bem no cenário proposto e, portanto, tem uma boa capacidade de generalização.

4.1.2 Comparação do ajuste fino versus uso dos modelos como extratores de características

Como pode ser visto através dos resultados do experimento anterior, o modelo BERT base obteve os melhores resultados no cenário de mais baixa disponibilidade de dados para treinamento. O fato do mesmo ter sido capaz de realizar o procedimento de ajuste fino com volumes tão pequenos para treinamento é um resultado interessante. Na prática, quando se tem poucos dados para o treinamento de um modelo onde se deseja realizar o procedimento de transferência de aprendizado para uma tarefa subsequente, este é mais comumente utilizado como um extrator de características, como apontado em Karphathy et al (Karpathy, Andrej, 2019).

Dado a capacidade do modelo BERT, em sua versão base, de realizar a tarefa de ajuste fino com exito e sem sofrer sobreajuste, este experimento buscou quantificar a diferença de performance quando este é usado como um extrator de características em contraste quando o mesmo é usado após ajuste fino.

Primeiramente, foi realizado um teste ANOVA (MONTGOMERY, 2003) para verificar se existe uma diferença entre a performance média em termos de acurácia do modelo BERT base após o ajuste fino em comparação com a utilização de redes LSTM e CNN que tem como entrada características extraídas do próprio BERT base. Para um nível de significância de 5%, a hipótese nula (de médias iguais) foi rejeitada para todos os conjuntos de dados. Em seguida, foi realizado o teste HSD (*honest significant difference*) pareado de Tukey (MONTGOMERY, 2003), cujos resultados podem ser vistos na Figura 40.

Os testes realizados atendem as premissas de normalidade dos resíduos e homogeneidade de variâncias estabelecidas pelo ANOVA. Vale ressaltar que pelo fato dos experimentos serem balanceados e os dados obedecerem uma premissa de independência, assegurada pelo próprio desenho do experimento, o ANOVA é relativamente robusto a pequenas violações de homoscedasticidade. Para garantir que as premissas são cumpridas foram aplicados o teste de Levene (BROWN; FORSYTHE, 1974), para verificar homogeneidade de variâncias, e o teste de Shapiro Wilk (SHAPIRO; WILK, 1965), para verificar normalidade dos resíduos.

A partir do teste de Tukey, fica claro a disparidade de performance de quando o modelo é utilizado após o processo de ajuste fino em comparação com sua utilização como

extrator de características em conjunção com outro modelo.

A robustez do modelo BERT base em conseguir ser treinado (ajuste fino) em um cenário de baixa volumetria de dados, aliado a sua performance claramente superior, o tornam a alternativa óbvia, para ser utilizado como classificador de texto com poucos dados. No entanto, vale ressaltar o fato de que os benefícios quantitativos em termos de performance do modelo, quando realizado seu ajuste fino, vem com *trade-offs* em relação a algumas outras características mais qualitativas e de caráter prático.

Em particular, modelos como o BERT e XLNet tem um custo computacional bastante elevado e requerem hardware especializado (usualmente na forma de GPUs ou TPUs com alta memória disponível). Isso implica numa maior dificuldade de implantação deste modelo em um cenário produtivo assim como uma complexidade extra em seu treinamento, mesmo considerando apenas o cenário de ajuste fino.

Não obstante, a possibilidade de usar o BERT como extrator de características, como pode ser visto através da Figura 40, o coloca num patamar de performance similar ou ligeiramente superior aos métodos clássicos, o que é possível observar comparando os resultados com os da Figura 34. Os resultados da performance média nos conjuntos de dados propostos, quando são comparados BERT+CNN e TFIDF+SVM, estão expostos na Tabela 3. Como é possível observar, em cenários com baixa disponibilidade de recursos computacionais, a utilização do BERT como extrator de características pode ser uma alternativa competitiva em comparação aos métodos clássicos.

Tabela 3 – Comparação do BERT como extrator de características com métodos clássicas

Modelo\Dataset	**CR**	**MPQA**	**IMDb**	**SST2**
BERT + CNN	**0.7253**	**0.7086**	0.6810	**0.7069**
TFIDF + SVM	0.7033	0.7067	**0.6909**	0.6501

4.1.3 Comparação de embeddings contextuais com embeddings tradicionais

São visíveis os ganhos obtidos provenientes das técnicas baseadas na arquitetura *transformer* como um todo, cujo maior benefício teórico é derivar representações robustas para texto (DEVLIN et al., 2018; YANG et al., 2019). Estas, por sua vez, são mais profundas e capturam mais elementos que os *embeddings* tradicionais. No entanto, embora fique claro a capacidade destes modelos em extrair bons resultados, num cenário de poucos dados, através do procedimento de pré treino seguido do ajuste fino, é interessante verificar quanto deste ganho deriva do treinamento anterior e quanto a representação dos *embeddings* contextuais, sem influência do ajuste para a tarefa subsequente, consegue representar melhor os dados em comparação com *embeddings* tradicionais.

Para isso os *embeddings* extraídos de BERT base foram comparados com os *embeddings* Glove, usando as mesmas arquiteturas de rede para classificação do texto,

sendo estas as redes LSTM e CNN, descritas anteriormente na seção 3.

Primeiramente, foi realizado um teste ANOVA para verificação de diferença entre as médias dos resultados. Para um nível de significância de 5%, a hipótese nula (de médias iguais) foi rejeitada para todos os conjuntos de dados. Em seguida foi realizado o teste HSD (*honest significant difference*) de Tukey pareado, cujos resultados podem ser vistos na Figura 45. Os testes realizados atendem as premissas de normalidade dos resíduos e homogeneidade de variâncias estabelecidas pelo ANOVA. Para garantir que as premissas são cumpridas foram aplicados o teste de Levene, para verificar homogeneidade de variâncias, e o teste de Shapiro Wilk, para verificar normalidade dos resíduos. Pequenos desvios nos resultados deste experimentos foram ignorados devido ao fato da amostra escolhida ser balanceada e o ANOVA ser relativamente robusto nesta situação.

Através dos resultados obtidos no teste de Tukey, vistos na Figura 45, é possível observar que para todos os conjuntos de dados, com excessão do conjunto de dados MPQA, a combinação de características extraídas com o BERT seguido de uma rede CNN, apresenta performance significativamente superior às demais abordagens baseadas em *embeddings* tradicionais (representadas pelo modelo Glove neste experimento).

Para o conjunto de dados MPQA é possível ver um comportamento oposto ao obtido nos demais conjuntos de dados. Este por sua vez é bastante particular em termos de suas características, pois apresenta um comprimento médio de sequência de tamanho 3. Dado que o ganho maior da representação derivada de BERT esta na característica contextual dos *embeddings* gerados, os resultados inferiores neste conjunto de dados são razoáveis, pois, nesta situação, os *embeddings* contextuais não oferecem nenhuma vantagem do ponto de vista teórico. Por um lado, também vale ressaltar que a dinâmica de pré treino que gera cada uma dessas representações testadas é razoavelmente diferente, e que, neste caso, o *corpus* de texto utilizado para o treinamento e os *manifolds* aprendidos durante o mesmo podem ser mais relevantes que as características individuais, em termos de semântica ou de habilidade de derivar contexto, das técnicas em questão.

Através deste experimento também é possível observar, através da Figura 45, ao analisar os resultados dos métodos baseados na representação Glove, que a performance deste nos conjuntos de dados analisados é bastante similar à performance dos métodos baseados em aprendizado estatístico, mostrados na Figura 34. Isso deve ao fato de que, a quantidade limitada de dados faz com que as redes treinadas com esse tipo de representação tenham dificuldade em generalizar, o que é um resultado muito comum mesmo considerando o cenário de transferência de aprendizado, conforme discutido em Karphathy et al (Karpathy, Andrej, 2019). A Tabela 4 mostra o desempenho médio de GLOVE+CNN em comparação com TFIDF+SVM.

Tabela 4 – Comparação de *embeddings* tradicionais com métodos clássicas

Modelo\Dataset	**CR**	**MPQA**	**IMDb**	**SST2**
GLOVE + CNN	0.6599	**0.7866**	0.6277	**0.6615**
TFIDF + SVM	**0.7033**	0.7067	**0.6909**	0.6501

4.1.4 Uso de função de perda de articulação como função de custo durante ajuste fino

Através dos resultados dos experimentos anteriores fica claro que existe um efeito bastante positivo em usar as técnicas baseadas em *embeddings* contextuais após a realização do procedimento de ajuste fino do modelo. Este procedimento garante resultados superiores às outras formas de utilização de *embeddings* para classificação de texto, especialmente em um cenário de baixa disponibilidade de dados.

Este experimento explora uma opção como tentativa de melhorar ainda mais os resultados obtidos através do ajuste fino desta classe de modelos. O ajuste fino é, afinal, um procedimento bastante simples, e como detalhado na Seção 2, consiste apenas da conexão de uma camada *softmax* no *token* de saída equivalente ao *token* CLS destes modelos. A função de custo utilizada para este procedimento, por sua vez também é bastante simples, sendo usualmente utilizada a entropia cruzada. Como tentativa de gerar uma superfície de decisão que tenha maior capacidade de generalizar durante o ajuste fino do modelo, foi utilizada uma função de custo baseada em função de perda de articulação para este mesmo procedimento. Isso visa observar se a característica de obtenção de funções de decisão que utilizam do princípio de margem larga impactam positivamente o ajuste fino do modelo num cenário de escassez de dados.

Como pode ser visto através da Figura 50, onde são apresentados *box-plots* que mostram os resultados dos testes para os conjuntos de dados analisados, não há um ganho de performance aparente entre a utilização de uma ou outra função de custo para nenhum dos casos. Para confirmar estes resultados, t-tests pareados foram realizados e, a um nível de significância estatística de 5%, a hipótese nula não foi rejeitada para nenhum dos casos analisados.

Este resultado leva a conclusão que, embora simples, o processo de otimização e função de custo escolhida para o procedimento de ajuste fino do BERT base, é suficiente para se obter boa generalização. Isso é reforçado pelo fato de que, na execução destes experimentos, nenhum ajuste extensivo de hiper-parâmetro na parte do ajuste fino foi realizado e, mesmo com um conjunto de hiperparametros iguais para todos os conjuntos de dados, a performance nos experimentos foi melhor que nos demais cenários analisados.

4.1.5 Considerações gerais

Através dos resultados dos experimentos apresentados nesta seção é possível derivar algumas intuições a respeito dos métodos estudados, que geram *embeddings* contextuais. Vale ressaltar que, os testes empíricos apresentados não são suficientes para derivar conclusões gerais em cima do comportamento destes modelos, servindo apenas para o cenário apresentado. Não obstante, estes resultados servem como bons indicadores para avaliar a performance dos mesmos em cenários similares, que são usualmente encontrados na indústria.

O caráter experimental proposto, assim como as características dos experimentos, derivadas do fato de serem tarefas de classificação de texto com poucos dados, é de grande importância prática, pois este é um cenário muito comumente encontrado na indústria. Em particular a ausência de dados para treinamento de modelos ocorre por diversos motivos, incluindo muitas vezes fatores como um custo elevado para coleta e rotulação de dados.

A partir dos testes realizados é possível ver alternativas que viabilizam a utilização de modelos neste cenário, o que é de grande valor em diversos casos. Em particular, através do resultado da experimentação acima foi possível observar que os modelos baseados na arquitetura *transformer* apresentam um desempenho muito bom quando é realizado o procedimento de ajuste fino do modelo para a tarefa subsequente. Também é possível observar que os modelos com menor capacidade são mais performáticos e propensos a terem uma otimização bem sucedida que os demais, em cenários mais críticos de ausência de dados, como pode ser visto à partir da performance superior do modelo BERT base neste caso. Isso é coerente com a teoria clássica de aprendizado de máquina (HASTIE; TIBSHIRANI; FRIEDMAN, 2001), pois por ter menos parâmetros este se mostrou de mais fácil otimização e menos propenso ao sobreajuste. No entanto, ainda é impressionante a capacidade desta técnica em apresentar resultados tão bons e ter boa capacidade de generalização com tão poucos dados disponíveis para ajuste fino.

Também ficou visível que existe um ganho evidente na utilização do ajuste fino em relação à utilização dos modelos como extratores de características, o que leva a acreditar que existe uma correlação positiva na utilização da arquitetura *transformer* no que tange a capacidade desta de realizar de forma mais eficiente que as demais o processo de transferência de aprendizado. Não obstante os *embeddings* contextuais gerados por esta arquitetura também garantem, na maior parte dos conjuntos de dados testados, uma performance final melhor em comparação ao *embedding* tradicional. Isso é um indicativo de que este tipo de representação é de fato mais profunda e robusta, de forma que garante que os modelos que os utilizam tenham maior capacidade de generalização.

Por fim, a tentativa de mudar a função de custo do modelo BERT base em seu procedimento de ajuste fino não gerou ganhos de performance. Isso leva a conclusão de

que, em seu estado atual, o modelo já tem capacidade de generalização suficiente.

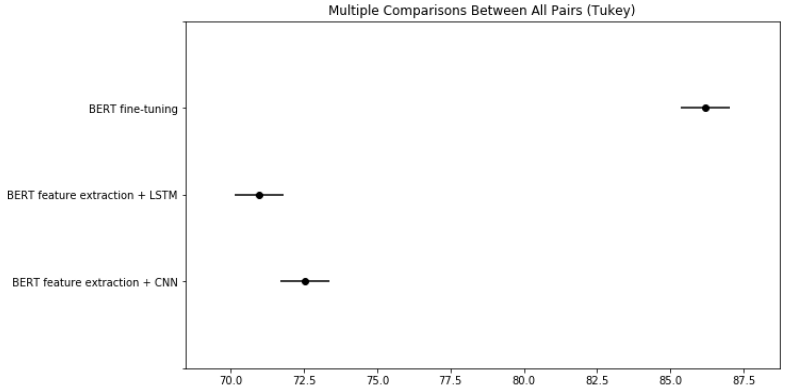


Figura 36 – CR

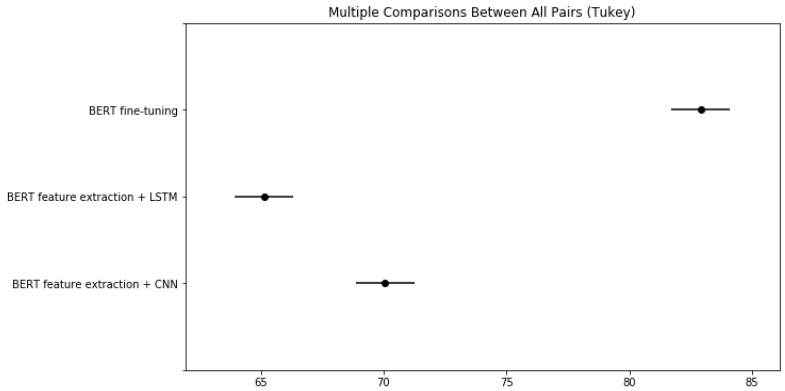


Figura 37 – MPQA

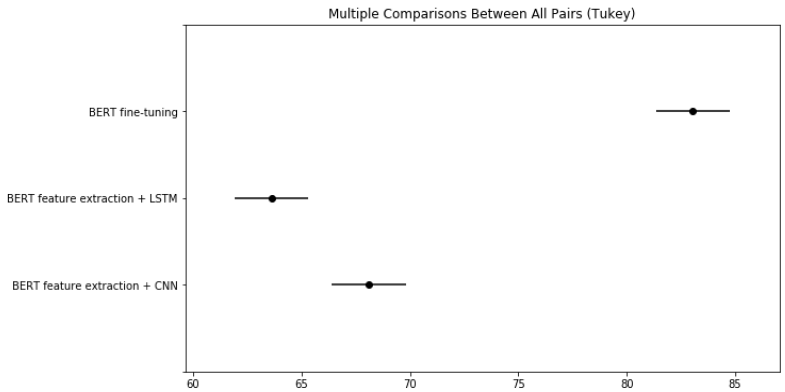


Figura 38 – IMDb

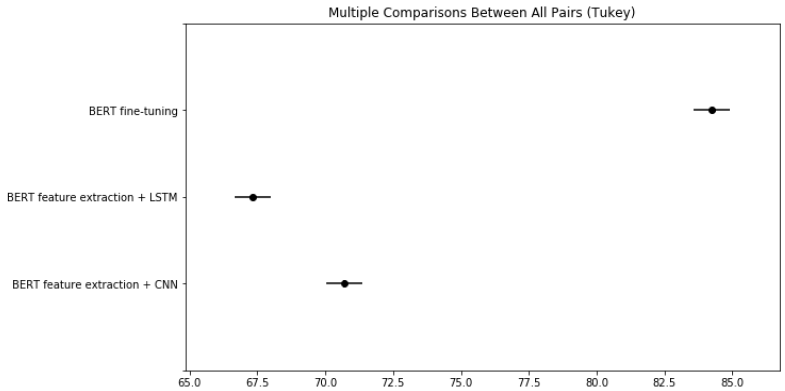


Figura 39 – SST2

Figura 40 – Resultados do teste Tukey HSD pareado para comparação da utilização do ajuste fino *versus* modelo como extrator de características

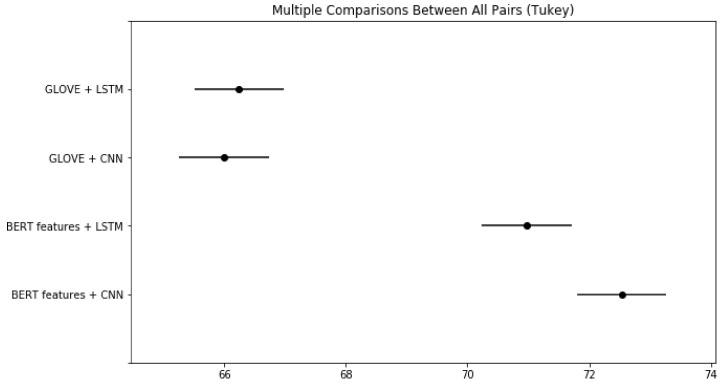


Figura 41 – CR

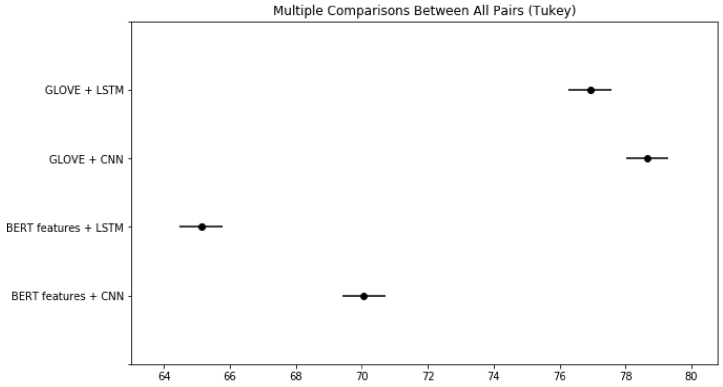


Figura 42 – MPQA

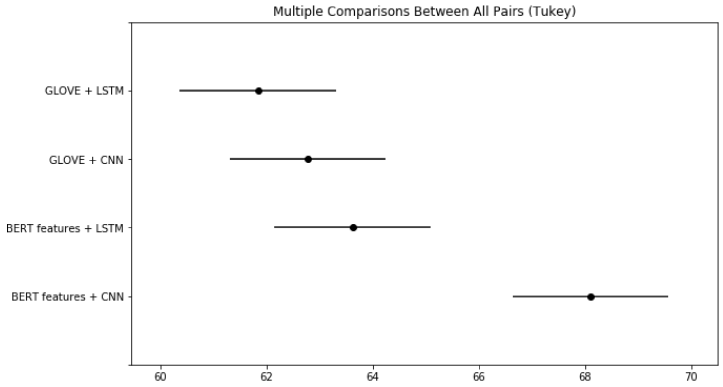


Figura 43 – IMDb

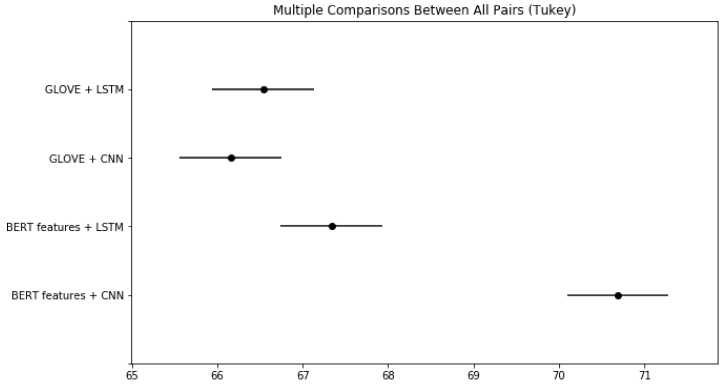


Figura 44 – SST2

Figura 45 – Resultados do teste Tukey HSD pareado para comparação de *embeddings* contextuais com *embeddings* tradicionais

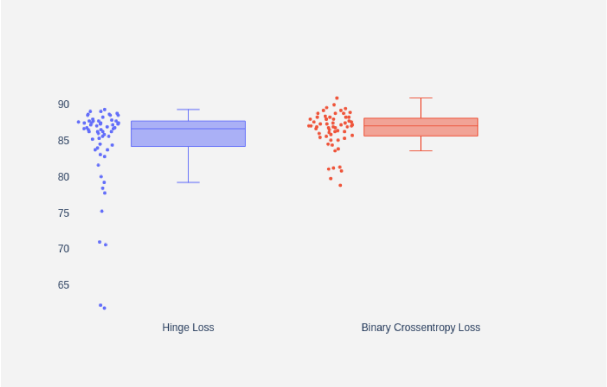


Figura 46 – CR

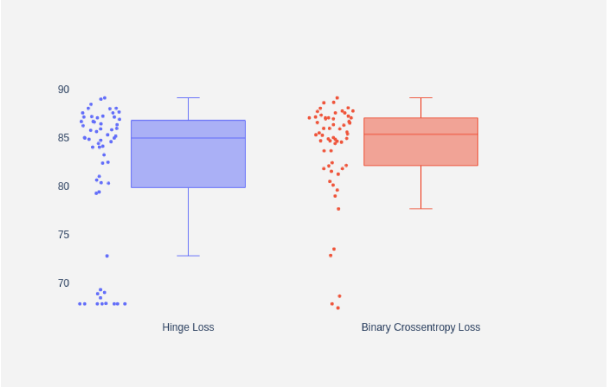


Figura 47 – MPQA

Figura 48 – IMDb

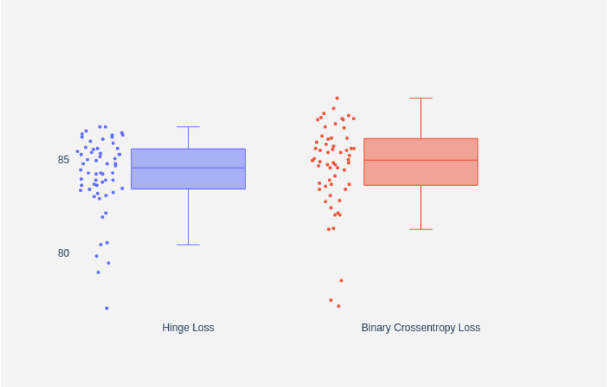


Figura 49 – SST2

Figura 50 – Diferença de acurácia da performance do modelo no cenário de ajuste fino com função de custo baseada em função de perda de articulação *versus* entropia binária cruzada

5 Conclusão

Neste trabalho foram avaliados dois modelos para tarefas de classificação de texto em um cenário de poucos de dados, BERT e XLNet. Estes modelos seguem uma arquitetura baseada na arquitetura *transformer* e tem a propriedade de gerar representações mais robustas através de *embeddings* contextuais.

Essa análise parte de premissas teóricas ou observáveis do comportamento desses modelos e de suas características arquiteturais, no entanto, a pergunta respondida é de caráter bem prático pois busca atacar o cenário de baixa disponibilidade de dados, que é um dos mais comuns na indústria e que ainda não é bem resolvido.

A partir de uma análise empírica, foi possível observar que esta classe de modelos, neste trabalho representada por BERT e XLNet, é capaz de performar bem no cenário proposto. Os testes demonstraram uma boa capacidade destes métodos de realizarem a transferência de aprendizado para a tarefa de classificação subsequente. Foi possível notar que existem ganhos na utilização destes modelos como extratores de características quando comparado a técnicas de *embeddings* tradicionais, porém o ganho é ainda maior quando são ajustados através da prática do ajuste fino do modelo para a tarefa subsequente.

O modelo BERT, na sua versão base, foi o de melhor performance para todos os cenários de dados apresentados, incluindo o mais escasso. O fato de modelos profundos serem capazes de tal capacidade de generalização com poucos dados para treinamento, mesmo considerando o cenário de transferência de aprendizado, é um achado interessante. Além disso, a consistência com que estes resultados foram gerados o tornam candidatos fortes para atacar este tipo de problema.

O trabalho apresentado tem como maior contribuição a constatação empírica sobre a performance dos métodos BERT e XLNet, capazes de gerar uma nova representação na forma de *embeddings* contextuais, em um cenário de poucos dados. Estes representam uma quebra de paradigma em relação ao uso comum de técnicas de *deep learning*, por serem modelos com grande capacidade mas ao mesmo tempo terem grande poder de generalização e não sofrerem de sobreajuste em um cenário de baixo volume de dados, mediante transferência de aprendizado.

6 Trabalhos Futuros

Através deste trabalho foi possível ver a eficiência dos modelos avaliados, BERT e XLNet, que apresentam boa performance no cenário de poucos dados. É notável perceber que a versão reduzida do modelo BERT, em particular, é aquela que apresenta os melhores e mais consistentes resultados, em especial em situações de mais baixa volumetria de dados. A diferença de capacidade entre as versões large e base destes modelos é grande, chegando a aproximadamente 230M de parâmetros.

Através dos resultados apresentados, é possível perceber que existe uma correlação entre os modelos com uma menor capacidade e uma melhor performance com a diminuição progressiva de dados para treinamento, dentre os que seguem a arquitetura analisada. Estes mantém a capacidade de generalização em cenários de menores volumetrias de dados disponíveis. Também é possível constatar que um dos maiores *trade-offs* na utilização dos modelos propostos está em seu alto custo computacional, mesmo durante o procedimento de ajuste fino.

Como trabalhos futuros, seria interessante achar um ponto ótimo de capacidade para esta classe de modelos, de forma que estes utilizem um número menor de parâmetros mas que os mesmos consigam performar bem no cenário de baixa volumetria de dados com menor custo computacional atrelado. Essa seria uma contribuição importante e poderia potencialmente fazer destes métodos um novo *baseline* para tarefas de classificação de texto com poucos dados.

7 Resultados dos testes estatísticos realizados

Nesta seção serão apresentados os resultados dos testes estatísticos realizados para os experimentos propostos.

```
ANOVA table
                    sum_sq    df           F        PR(>F)
C(treatments)  4200.131937   2.0  280.882559  1.104795e-38
Residual        650.470217  87.0         NaN           NaN
Levene results: 0.6980101114799195
Shapiro wilk results: 2.0413473976077512e-05
```

Figura 51 – Resultados teste ANOVA para o *dataset* CR para o experimento 4.1.2

```
ANOVA table
                    sum_sq     df          F        PR(>F)
C(treatments)  991.872690    3.0  69.146476  1.030684e-25
Residual       554.654599  116.0        NaN           NaN
Levene results: 0.31522828797411323
Shapiro wilk results: 0.0004022323992103338
```

Figura 52 – Resultados teste ANOVA para o *dataset* CR para o experimento 4.1.3

```
                    Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================================
           group1                        group2            meandiff p-adj   lower   upper  reject
-------------------------------------------------------------------------------------------------------------
 BERT feature extraction + CNN BERT feature extraction + LSTM  -1.5629 0.0745 -3.2464  0.1206  False
 BERT feature extraction + CNN              BERT fine-tuning  13.6468  0.001 11.9633 15.3303   True
BERT feature extraction + LSTM              BERT fine-tuning  15.2097  0.001 13.5262 16.8932   True
-------------------------------------------------------------------------------------------------------------
```

Figura 53 – Resultados teste Tukey HSD para o *dataset* CR para o experimento 4.1.2

```
                Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================
       group1               group2        meandiff p-adj   lower   upper  reject
---------------------------------------------------------------------------------------------
 BERT features + CNN BERT features + LSTM  -1.5629 0.0328 -3.0347 -0.0911   True
 BERT features + CNN          GLOVE + CNN   -6.543  0.001 -8.0149 -5.0712   True
 BERT features + CNN         GLOVE + LSTM  -6.3002  0.001  -7.772 -4.8284   True
BERT features + LSTM          GLOVE + CNN  -4.9801  0.001 -6.4519 -3.5083   True
BERT features + LSTM         GLOVE + LSTM  -4.7373  0.001 -6.2091 -3.2655   True
         GLOVE + CNN         GLOVE + LSTM   0.2428    0.9  -1.229  1.7146  False
---------------------------------------------------------------------------------------------
```

Figura 54 – Resultados teste Tukey HSD para o *dataset* CR para o experimento 4.1.3

```
ANOVA table
                     sum_sq    df           F        PR(>F)
C(treatments)  5044.316838   2.0  167.898014  1.356417e-30
Residual       1306.911124  87.0         NaN           NaN
Levene results: 0.0007965026159933681
Shapiro wilk results: 2.5917170631828412e-09
```

Figura 55 – Resultados teste ANOVA para o *dataset* MPQA para o experimento 4.1.2

```
ANOVA table
                     sum_sq     df           F        PR(>F)
C(treatments)  3531.716886    3.0  313.506191  1.844373e-55
Residual        435.588589  116.0         NaN           NaN
Levene results: 0.00029966869441296035
Shapiro wilk results: 0.42378368973731995
```

Figura 56 – Resultados teste ANOVA para o *dataset* MPQA para o experimento 4.1.3

```
                  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================================
           group1                         group2             meandiff p-adj  lower   upper  reject
-------------------------------------------------------------------------------------------------------------
 BERT feature extraction + CNN BERT feature extraction + LSTM  -4.9316 0.001 -7.3179 -2.5453   True
 BERT feature extraction + CNN              BERT fine-tuning  12.8304 0.001 10.4441 15.2167   True
BERT feature extraction + LSTM              BERT fine-tuning  17.7621 0.001 15.3758 20.1484   True
-------------------------------------------------------------------------------------------------------------
```

Figura 57 – Resultados teste Tukey HSD para o *dataset* MPQA para o experimento 4.1.2

```
              Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================================
       group1               group2        meandiff p-adj   lower   upper  reject
---------------------------------------------------------------------------------
 BERT features + CNN BERT features + LSTM  -4.9316  0.001 -6.2359 -3.6273   True
 BERT features + CNN          GLOVE + CNN   8.6013  0.001   7.297  9.9056   True
 BERT features + CNN         GLOVE + LSTM   6.8694  0.001  5.5651  8.1737   True
BERT features + LSTM          GLOVE + CNN  13.5329  0.001 12.2286 14.8372   True
BERT features + LSTM         GLOVE + LSTM   11.801  0.001 10.4967 13.1053   True
         GLOVE + CNN         GLOVE + LSTM  -1.7319 0.0041 -3.0362 -0.4276   True
---------------------------------------------------------------------------------
```

Figura 58 – Resultados teste Tukey HSD para o *dataset* MPQA para o experimento 4.1.3

```
ANOVA table
                     sum_sq    df           F        PR(>F)
C(treatments)  6207.814721   2.0  101.210619  1.961170e-23
Residual       2668.098888  87.0         NaN           NaN
Levene results: 0.13635674172107512
Shapiro wilk results: 9.111215471327427e-13
```

Figura 59 – Resultados teste ANOVA para o *dataset* IMDb para o experimento 4.1.2

```
ANOVA table
                     sum_sq     df           F        PR(>F)
C(treatments)   694.912213    3.0  12.187828  5.450430e-07
Residual       2204.653653  116.0         NaN           NaN
Levene results: 0.09330241679589728
Shapiro wilk results: 1.195845783508176e-10
```

Figura 60 – Resultados teste ANOVA para o *dataset* IMDb para o experimento 4.1.3

```
                    Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================================================
           group1                        group2            meandiff p-adj   lower   upper  reject
-------------------------------------------------------------------------------------------------
 BERT feature extraction + CNN BERT feature extraction + LSTM   -4.488 0.0065 -7.8976 -1.0784   True
 BERT feature extraction + CNN              BERT fine-tuning  14.9398  0.001 11.5302 18.3494   True
BERT feature extraction + LSTM              BERT fine-tuning  19.4278  0.001 16.0182 22.8374   True
-------------------------------------------------------------------------------------------------
```

Figura 61 – Resultados teste Tukey HSD para o *dataset* IMDb para o experimento 4.1.2

```
                Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================================
       group1               group2        meandiff p-adj   lower   upper  reject
---------------------------------------------------------------------------------
 BERT features + CNN BERT features + LSTM   -4.488  0.001 -7.4223 -1.5537   True
 BERT features + CNN          GLOVE + CNN  -5.3333  0.001 -8.2677  -2.399   True
 BERT features + CNN         GLOVE + LSTM  -6.2693  0.001 -9.2037  -3.335   True
BERT features + LSTM          GLOVE + CNN  -0.8453 0.8639 -3.7797   2.089  False
BERT features + LSTM         GLOVE + LSTM  -1.7813 0.3937 -4.7157   1.153  False
         GLOVE + CNN         GLOVE + LSTM   -0.936 0.8189 -3.8703  1.9983  False
---------------------------------------------------------------------------------
```

Figura 62 – Resultados teste Tukey HSD para o *dataset* IMDb para o experimento 4.1.3

```
ANOVA table
                    sum_sq     df           F         PR(>F)
C(treatments)  4802.745621    2.0  499.200565  2.093682e-48
Residual        418.508009   87.0         NaN           NaN
Levene results: 0.44800599825338605
Shapiro wilk results: 0.004053567070513964
```

Figura 63 – Resultados teste ANOVA para o *dataset* SST2 para o experimento 4.1.2

```
ANOVA table
                    sum_sq      df           F         PR(>F)
C(treatments)  384.290263     3.0  40.971479  3.949833e-18
Residual       362.672368   116.0         NaN           NaN
Levene results: 0.24322276015117636
Shapiro wilk results: 0.10777300596237183
```

Figura 64 – Resultados teste ANOVA para o *dataset* SST2 para o experimento 4.1.3

```
                    Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================================================
           group1                        group2            meandiff p-adj  lower   upper  reject
-------------------------------------------------------------------------------------------------
 BERT feature extraction + CNN BERT feature extraction + LSTM  -3.3553 0.001 -4.7057 -2.0049   True
 BERT feature extraction + CNN              BERT fine-tuning  13.5438 0.001 12.1935 14.8942   True
BERT feature extraction + LSTM              BERT fine-tuning  16.8991 0.001 15.5488 18.2495   True
-------------------------------------------------------------------------------------------------
```

Figura 65 – Resultados teste Tukey HSD para o *dataset* SST2 para o experimento 4.1.2

```
                Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================================
       group1               group2        meandiff p-adj   lower   upper  reject
---------------------------------------------------------------------------------
 BERT features + CNN BERT features + LSTM  -3.3553  0.001 -4.5454 -2.1652   True
 BERT features + CNN          GLOVE + CNN  -4.5341  0.001 -5.7243  -3.344   True
 BERT features + CNN         GLOVE + LSTM  -4.1534  0.001 -5.3435 -2.9633   True
BERT features + LSTM          GLOVE + CNN  -1.1788 0.0532  -2.369  0.0113  False
BERT features + LSTM         GLOVE + LSTM  -0.7981  0.304 -1.9882   0.392  False
         GLOVE + CNN         GLOVE + LSTM   0.3807 0.8175 -0.8094  1.5709  False
---------------------------------------------------------------------------------
```

Figura 66 – Resultados teste Tukey HSD para o *dataset* SST2 para o experimento 4.1.3

Tabela 5 – Resultados dos t-tests pareados do experimento 4.1.4

Paired t-test results	
Dataset	p-value
CR	0.059
MPQA	0.062
IMDb	0.228
SST2	0.261

REFERÊNCIAS

Alammar, Jay . *The Illustrated Word2Vec*. 2019. [Online; accessed 05-Jun-2019]. Disponível em: ⟨https://jalammar.github.io/illustrated-word2vec/⟩.

BA, J. L.; KIROS, J. R.; HINTON, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

BROWN, M. B.; FORSYTHE, A. B. Robust tests for the equality of variances. *Journal of the American Statistical Association*, Taylor & Francis, v. 69, n. 346, p. 364–367, 1974.

CER, D. et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

DAI, A. M.; LE, Q. V. Semi-supervised sequence learning. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2015. p. 3079–3087.

DAI, Z. et al. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. ⟨http://www.deeplearningbook.org⟩.

Gupta, Dishashree. *RNN cell*. 2019. [Online; accessed 05-Jun-2019]. Disponível em: ⟨https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/⟩.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc., 2001. (Springer Series in Statistics).

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.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HOWARD, J.; RUDER, S. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

JOACHIMS, T. Text categorization with support vector machines: Learning with many relevant features. In: SPRINGER. *European conference on machine learning*. [S.l.], 1998. p. 137–142.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2019. ISBN 0130950696.

Karpathy, Andrej. *Convolutional Neural Networks for visual recognition*. 2019. [Online; accessed 05-Jun-2019]. Disponível em: ⟨http://cs231n.github.io/transfer-learning/⟩.

KIM, Y. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

KUDO, T.; RICHARDSON, J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, 2018. p. 66–71. Disponível em: ⟨https://www.aclweb.org/anthology/D18-2012⟩.

Lee, Ceshine. *Bi-LSTM*. 2019. [Online; accessed 05-Jun-2019]. Disponível em: ⟨https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66⟩.

LEVI-MONTALCINISINEK, R. *Above all, don't fear difficult moments. The best comes from them*. [S.l.]: Internet, 2020.

LIU, X. et al. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*, 2019.

LUONG, M.-T.; PHAM, H.; MANNING, C. D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

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.

MONTGOMERY, D. C. Applied statistics and probability for engineers. 2003.

MOU, L. et al. How transferable are neural networks in nlp applications? *arXiv preprint arXiv:1603.06111*, 2016.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. In: *International conference on machine learning*. [S.l.: s.n.], 2013. p. 1310–1318.

PENNINGTON, J.; SOCHER, R.; MANNING, C. Glove: Global vectors for word representation. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. [S.l.: s.n.], 2014. p. 1532–1543.

PETERS, M. E. et al. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

POGGIO, T.; TORRE, V. *Ill-posed problems and regularization analysis in early vision*. [S.l.], 1984.

RADFORD, A. et al. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

ROBERTSON, S. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, Emerald Group Publishing Limited, v. 60, n. 5, p. 503–520, 2004.

SHAPIRO, S. S.; WILK, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, JSTOR, v. 52, n. 3/4, p. 591–611, 1965.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. *Information processing & management*, Elsevier, v. 45, n. 4, p. 427–437, 2009.

TORREY, L.; SHAVLIK, J. Transfer learning. In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. [S.l.]: IGI Global, 2010. p. 242–264.

VAPNIK, V. N. An overview of statistical learning theory. *IEEE transactions on neural networks*, Citeseer, v. 10, n. 5, p. 988–999, 1999.

VASWANI, A. et al. Attention is all you need. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 5998–6008.

VINCENT, P. et al. Extracting and composing robust features with denoising autoencoders. In: ACM. *Proceedings of the 25th international conference on Machine learning*. [S.l.], 2008. p. 1096–1103.

Wikipedia contributors. *Wikipedia, The Free Encyclopedia*. 2004. [Online; accessed 18-May-2019]. Disponível em: ⟨https://en.wikipedia.org/⟩.

WU, Y. et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

YANG, Z. et al. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

ZHANG, Y.; JIN, R.; ZHOU, Z.-H. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, Springer, v. 1, n. 1-4, p. 43–52, 2010.