Identificação e classificação de comentários tóxicos utilizando processamento de linguagens naturais e técnicas de aprendizagem profunda

Miguel Angelo Cece de Castro Neto[1], **Paulo Alves dos Santos Junior**[1]
Alceu de Souza Britto Junior[2]

[1] Departamento de Informática – Universidade Estadual de Ponta Grossa (UEPG)
Avenida General Carlos Cavalcanti, 4748
CEP 84030-900 – Ponta Grossa, PR – Brasil

[2]Programa de Pós-Graduação em Informática (PPGIa) – Pontificia Universidade Católica do Paraná (PUCPR)
Rua Imaculada Conceição, 1155
CEP 80215-901 – Curitiba, PR – Brasil

miguelceccineto@gmail.com, contato@pauloalvesjr.com, alceu@ppgia.pucpr.br

Abstract. *The paper addresses the problem of multiclass sentiment analysis at the sentence level. Recurrent neural networks and their demonstrations have demonstrated successful modeling of feeling classifiers and last generation in language modeling with the same demonstration demonstrating efficiency and performance in various tasks. In the paper we propose the use of combinations of architecture together with a combination of convolutional techniques.*

Resumo. *Este artigo aborda o problema da análise de sentimentos multiclasse no nível de sentença. Redes neurais recorrentes e suas variações demonstraram sucesso na modelagem de classificadores de sentimentos e recentemente a modelagem de linguagens através dessa arquitetura demonstrou-se eficaz atingindo o estado da arte em várias tarefas. Nesse trabalho propomos o uso destas variações arquitetura em conjunto com a combinação de técnicas convolucionais.*

1. Introdução

Discutir assuntos na internet pode ser difícil. Ameaças, ofensas pessoais, e assédio, podem criar um ambiente tóxico dentro de fóruns e redes sociais, e até afastar usuários. As plataformas lutam para combater efetivamente discussões tóxicas e limitar ou desligar usuários com essas práticas, porém essa é uma tarefa de difícil automação exigindo milhares de sentenças rotuladas e métodos capazes de analisar de maneira eficaz o contexto e significado.

Recentemente, em resposta ao crescimento do uso de redes sociais, comentários são publicados massivamente. Como a escrita compõe grande parte de todos os dados gerados pela humanidade, que até então, apenas serviam para consultas, hoje servem como base de dados para o desenvolvimento de sistemas de processamento de linguagens naturais. Aplicando técnicas de aprendizagem profunda são realizadas automações de processos, análises e classificação de textos.

No problema a ser discutido nesse artigo, com base de dados contendo comentários tóxicos, a análise de sentimentos indicará em quais classes de toxicidade a sentença será rotulada. Formalmente, o objetivo neste trabalho é extrair a seguinte sêxtupla de rótulos:

$$(a\lambda 1, a\lambda 2, a\lambda 3, a\lambda 4, a\lambda 5, a\lambda 6)$$

Onde $a\lambda i$ se refere a probabilidade de cada classe, que respectivamente representam: Tóxico; Muito Tóxico; Obsceno; Ameaça; Insulto; Ódio de Identidade.

A análise de sentimentos é determinada comumente de maneira binária (positiva e negativa), porém abordagens multiclasse também são possíveis. Em uma abordagem multiclasse determinado comentário pode possuir diversos rótulos simultaneamente. Para definir a ativação de determinada classe, é definido um limiar. Tal limiar só é válido devido a função de ativação utilizada como saída na rede.

A utilização de redes neurais artificiais voltou a popularizar-se no reconhecimento de padrões em imagens devido a evolução de hardware, tais métodos de aprendizagem de máquina combinados ao processamento de linguagens naturais, permitem a avaliação automática de padrões e extração de conhecimento de bases de texto. A identificação de comentários ofensivos em textos é um derivado do caso geral de análise de sentimentos, na era da internet, identificar e classificar sentenças permite tomada de decisões e maior controle e filtragem de conteúdo.

Esse trabalho tem como principal interesse, o estudo e experimento de técnicas de aprendizagem profunda (*deep learning*) aplicada à área de processamento de linguagens naturais. Abordagens de aprendizagem de máquina tradicionais necessitam de intervenção humana para definição de características, existe a possibilidade de delegar tal tarefa à algoritmos simples de extração de características línguisticas ou sintáticas disponíveis na literatura, como utilizando contagem [Ma et al. 2018], ou processamento esparso [Ramos et al. 2003]. A tarefa realizada utilizando métodos clássicos, por fim utilizam algum método superficial de aprendizado, como por exemplo, máquinas de vetores de suporte [Cortes and Vapnik 1995], naive bayes e máquinas de vetores de suport naive bayes [Wang and Manning 2012], que tem como o objetivo classificar características disponíveis.

Embora os métodos descritos anteriormente já tenham sido validados, o foco desse trabalho é utilizar um método automátizado de extração de características que seja eficaz. Para a análise de sentimentos em comentários e classificação de toxicidade, utilizaremos utilizaremos redes neurais recorrentes em conjunto com estruturas celulares específicas, como células recorrentes bloqueadas (*gated recurrent units*) [Pascanu et al. 2013] e células de longa memória de curto termo (*long-short term memory*) [Hochreiter and Schmidhuber 1997]. Abordaremos técnicas de aplicação de módulos convolucionais [Y. LeCun and Haffner 1998]. A estrutura recorrente é uma variação da rede neural clássica para aceitar entradas com tamanhos e saídas arbitrários, possuindo também registro de contexto em relação a posição do elemento em seu conjunto de entrada. Redes recorrentes comuns não são efetivas devido a problemas de esquecimento de aprendizado ou pouca capacidade de aprendizado devido a limitação imposta pela quantidade de parâmetros. Para resolver essa situação existem diferentes células que

podem ser incorporadas nas arquiteturas, como por exemplo células recorrentes bloqueadas (*gated recurrent units*) e células de longa memória de curto termo (*long-short term memory*) [Hochreiter and Schmidhuber 1997].

As redes neurais recorrentes possuem como característica a capacidade de trabalhar com entradas e saídas de tamanhos arbitrários, sendo uma ótima arquitetura para solução de problemas de múltiplas sentenças. É apenas necessário o pré-processamento de cada elemento, com o objetivo de remoção de ruídos e conversão para tensores numéricos [Karparthy 2015].

A adição de convoluções, também podem ser utilizadas, para melhorar a detecção de padrões nas saídas de redes recorrentes. As redes convolucionais normalmente são utilizadas para extração de características em imagens [Y. LeCun and Haffner 1998] ou quando as dimensões estão extremamente relacionadas. No caso imagens, são utilizadas convoluções de duas dimensões, no caso de vídeo ou objetos tridimensionais, três dimensões. Nesse artigo será avaliado o desempenho de convoluções de uma dimensão buscando o relacionamento dessas saídas.

Nesse trabalho as arquiteturas utilizadas aspiram utilizar os diversos conceitos disponíveis em busca de uma solução melhor que o aprendizado de máquina clássico e conceitos individuais. Na seção 2 é descrito brevemente trabalhos relacionados na área, tanto em aprendizado de máquina clássico, quanto aprendizado de máquina profundo. Na seção 3 estão os principais conceitos para o entendimento da seção 4, que se refere as arquiteturas utilizadas nesse artigo. Na seção 5 temos a análise e comparação de todos os modelos treinados pelos autores, já na seção 6 nossas considerações finais e trabalhos futuros.

Portanto o objetivo desse trabalho é a comparação de diversas arquiteturas de redes recorrentes e técnicas de processamento linguagens naturais, aplicadas à resolução do problema de identificação de comentários tóxicos.

2. Trabalhos Relacionados

A análise de sentimentos está sob grande aprimoramento devido a aplicação de redes recorrentes [Karparthy 2015] e redes convolucionais [Y. LeCun and Haffner 1998]. Esses casos descritos anteriormente se referem ao estado da arte dessa atividade.

Métodos clássicos de aprendizagem de máquina aplicados a análise e classificação de sentimento, como por exemplo *Suport Vector Machines* [Cortes and Vapnik 1995], apresentam resultados inferiores ao estado da arte. Essa abordagem necessita que as características sejam extraídas de maneira não automátizada, com ajuda de análise humanada, ou utilizando segmentações pouco eficientes, como por exemplo contagem de palavras e transformações esparsas.

O principal problema do método superficial citado acima, é que ele não é capaz de identificar o contexto de palavras, não sendo uma opção robusta para classificação.

Redes recorrentes e suas variações são aplicadas utilizando comumente matrizes que definem significado próximos ao real para cada palavra, como por exemplo *Word2Vec* [Mikolov et al. 2013] ou *GloVe* [Pennington et al. 2014]. Tais métodos provem contexto utilizando palavras próximas para modelas matrizes e consequentemente contexto para redes recorrentes.

Transferência de aprendizado é uma técnica onde um modelo já treinado é reajustado para detectar padrões diferentes dos iniciais. Geralmente usado quando a base de dados disponível é menor do que a usada no modelo já treinado. Trabalhos recentes propõem a utilização de modelos de linguagem universais, que sofrerão afinação (fine tuning) para atender uma tarefa específica [Howard and Ruder 2018] com a possibilidade de utilização de dados não supervisionados para gerar um aumento de performance (Radford et al. 2018). A utilização de dados não rotulados permite reduzir drasticamente a necessidade de exemplos rotulados. O estado da arte da maioria das tarefas na área de linguagens naturais está relacionado ao uso de transferência de aprendizado e aprendizado não supervisionado, que é um dos maiores desafios atualmente.

3. Fundamentação Teórica

3.1. Redes Artificiais Profundas (*deep feedforward networks*)

Redes *feedforward* profundas, também chamadas de redes neurais feedforward, ou perceptrons multicamadas (*multilayer perceptrons*), são essenciais nos modelos de aprendizagem profunda. O objetivo de uma rede feedforward é aproximar uma função f. Por exemplo, para um classificador, $y = f(x)$ mapeia uma entrada x para uma categoria y. Uma rede feedforward define um mapeamento $y = f(x)$ e aprende o valor dos parâmetros que resultam na melhor aproximação da função.

Esses modelos são chamados *feedforward* porque a informação flui através da função que está sendo avaliada de x, através dos cálculos intermediários usados para definir f e, finalmente, para a saída y. Não há conexões de retroalimentação nas quais as saídas do modelo são realimentadas. Quando redes neurais *feedforward* são estendidas para incluir conexões de retroalimentação, elas são chamadas de redes neurais recorrentes.

As redes feedforward são de extrema importância para os profissionais de aprendizado de máquina. Eles formam a base de muitas aplicações comerciais importantes. Por exemplo, as redes convolucionais usadas para reconhecimento de objetos em imagens são um tipo especializado de rede *feedforward*. As redes da *feedforward* são um trampolim conceitual no caminho para redes recorrentes, que potencializam muitas aplicações de linguagem natural. [Goodfellow et al. 2016].

3.2. Redes Neurais Recorrentes

Redes recorrentes possuem o mesmo conceito básico de redes neurais artificiais, com o objetivo de otimizar parâmetros para retornar uma saída coerente em relação a entrada e ao problema. O que faz as redes recorrentes serem especiais, é o fato de elas não possuirem a restrição de tamanho fixo de entrada ou saída [Karparthy 2015].

O conjunto de dados utilizados nesse artigo se adequa a arquitetura muitos para muitos, onde a entrada são as sequencias vetorizadas e tratadas através de *word embeddings*) e a saída é um vetor de probabilidades das classes.

O modelo muitos para muitos nesse conjunto de dados, possuirá a entrada onde cada bloco terá o número de *minibatches* de matrizes de *word embeddings* para cada palavra.

A saída se comportará como representado anteriormente, sendo a sêxtupla:

$$(a\lambda 1, a\lambda 2, a\lambda 3, a\lambda 4, a\lambda 5, a\lambda 6)$$

Onde $a\lambda i$ se refere a probabilidade de cada classe, que respectivamente representam: Tóxico; Muito Tóxico; Obsceno; Ameaça; Insulto; Ódio de Identidade.

A imagem a seguir demonstra alguns casos de redes recorrentes possíveis:

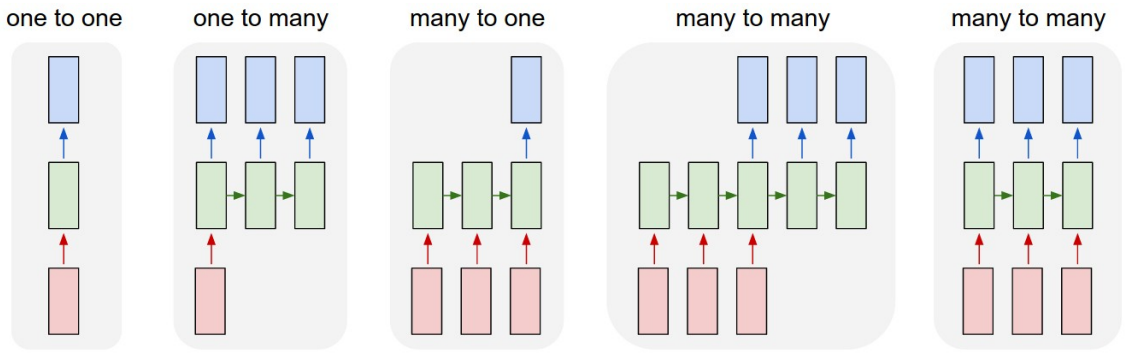

Figura 1. Possibilidades de entrada de saída em redes recorrentes. [Karparthy 2015]

3.3. Funções de ativação

Funções de ativação não lineares dão capacidades não lineares para redes neurais [Y. LeCun and Haffner 1998]. As funções de ativação nos permitem ir além de problemas linearmente separáveis. Caso essas funções não sejam adicionadas, o modelo apenas terá a capacidade de resolver problemas simples, sendo que todas as camadas formarão apenas um modelo linear.

É possível visualizar de maneira trivial na representação abaixo:

$$z_1(z_0; w, b) = z_0^T * w_1 + b \tag{1}$$

$$z_2(z_1; w, b) = z_1^T * w_2 + b \tag{2}$$

$$z_2(z_1; w, b) = (z_0^T * w_1 + b)^T * w_2 + b \tag{3}$$

A função (3), mesmo após a progressão através das camadas, continua sendo um modelo linear.

3.3.1. ReLU - *Rectified Linear Activation Function*

Dada pela fórmula:

$$ReLU(Z) = max(0, Z) \tag{4}$$

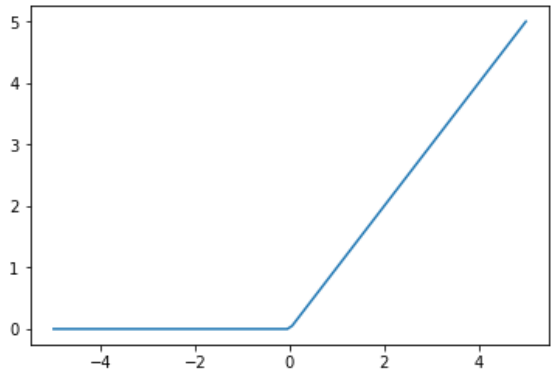

Figura 2. Unidade Linear Retificada

A imagem abaixo descreve expressão da ReLU em um intervalo de 5 a –5.

É a função de otimização padrão recomendada para uso na maioria das redes neurais. Aplicando essa função na saída de uma transformação linear produz uma transformação não linear [Nair and Hinton 2010].

A função permanece muito próxima de ser linear, contudo, no sentido de que é uma função linear por partes com duas peças lineares. Como as unidades lineares retificadas são quase lineares, elas preservam as propriedades que tornam os modelos lineares fáceis de otimizar com métodos baseados em gradiente. Eles também preservam muitas das propriedades que fazem modelos lineares generalizarem sistemas a partir de componentes mínimos [Goodfellow et al. 2016].

3.3.2. Sigmoide

A função sigmoide é dada pela formula:

$$\sigma(Z) = 1/(1 + e^{-Z}) \tag{5}$$

A imagem abaixo descreve a curva sigmoidal em um intervalo de 5 a –5.

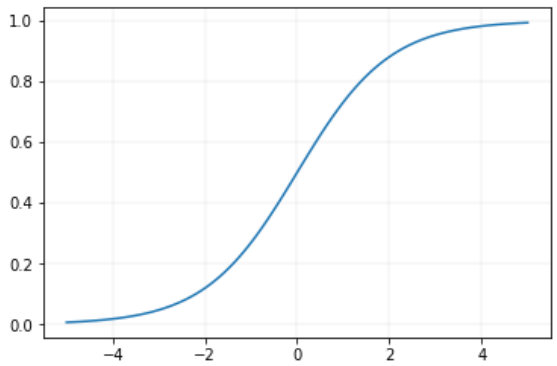

Figura 3. Curva Sigmoidal

Aplicando essa função no output de uma transformação linear, teremos como resultado um valor entre 0 e 1.

A função sigmoide é utilizada na estrutura de células de longa memória de curto termo (*long-short term memory*) e células recorrentes bloqueadas (*gated recurrent units*), também pode ser usada na saída do modelo. Por exemplo, na classificação de sentimentos em texto, considerar valores próximos de 0 para um sentimento negativo, e próximos de 1, para ositivo. Os valores maiores iguais a 0.5 serão classificados como positivos, enquanto 0.5 serão classificados como negativos.

3.3.3. Tangente Hiperbólica

Dada pela formula:

$$\sigma(Z) = 1/(1 + e^{-Z}) \tag{6}$$

A imagem abaixo descreve a curva da tangente hiperbolica em um intervalo de 5 a -5.

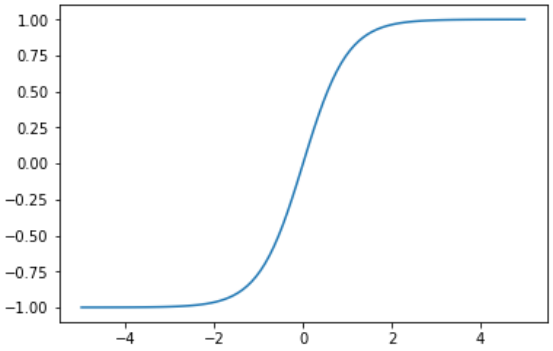

Figura 4. Curva Tangente Hiperbólica

Produz um resultado entre 1 e -1, e é usada na estrutura de células simples de redes recorrentes, células de longa memória de curto termo (*long-short term memory*) e células recorrentes bloqueadas (*gated recurrent units*). Em comparação a função de ativação sigmoide, possui a vantagem de ser simétrica em relação ao eixo x.

3.4. Algoritmos de treinamento/Otimizadores

3.4.1. Escolha do Algoritmo de Otimização

Infelizmente ainda não existe um consenso sobre qual é o melhor algoritmo de otimização. [Schaul et al. 2013] apresentou uma comparação valiosa com um grande número de algoritmos de otimização através de uma ampla variedade de tarefas. Enquanto os resultados sugeriam que a família de algoritmos com taxa de aprendizado adaptativa (como RMSProp e AdaDelta [Zeiler 2012]) apresentavam na maioria das tarefas desempenho superior aos outros métodos, porém, não houve um único algoritmo que emergiu sobre os outros.

Atualmente, os algoritmos de otimização mais utilizados na literatura são Gradiente Descendente Estocástico, Gradiente Descendente Estocástico com Momento, RMSProp, RMSProp com Momento, AdaDelta [Ruder 2016] e Adam [Kingma and Ba 2014]. A escolha de qual algoritmo de otimização usar, parece depender mais na familiaridade do desenvolvedor com o com o algoritmo (para afinar os hiperparametros mais facilmente) [Goodfellow et al. 2016].

3.4.2. Otimizador Adam

Com a utilização do método *minibatches*, cujo divide a base de dados em varios blocos chamados *minibatches*, o progresso no gradiente tende a oscilar, logo o aprendizado pela rede se torna lento. Várias soluções foram propostas para solucionar esse problema, e a mais eficaz é uma junção das anteriores. O otimizador Adam [Kingma and Ba 2014] utiliza a estimação do primeiro e do segundo momento dos gradientes, e terá a função de otimizar a função de custo.

3.5. Função de Custo

Um aspecto importante no desenvolvimento de redes neurais profundas é na escolha de função de custo. [Goodfellow et al. 2016]. As funções de custo de maneira geral são mecanismos para calcular a diferença entre os resultados do modelo, e os resultados desejados.

A escolha da função de custo foi baseada nos resultados alcançados em "Cross-Entropy vs. Squared Error Training: a Theoretical and Experimental Comparison"[Golik 2013], onde são aplicadas e comparadas funções de custo em modelos de classificação, como reconhecimento de fala e reconhecimento de caracteres.

3.5.1. Entropia Cruzada

O cálculo do custo por entropia cruzada é dado pela seguinte formula:

$$CE = -\sum_{x} p(x) \log q(x) \tag{7}$$

Onde $p(x)$ é o rótulo, e $q(x)$ o saída do modelo.

3.6. Regularização

Um problema central em aprendizado de máquina é como fazer um algoritmo que irá ter um bom desempenho não somente nos dados de treino, mas também em novos dados inseridos nele. Muitas estratégias usadas em aprendizado de máquina são explicitamente desenvolvidas para reduzir o erro no conjunto de teste, possivelmente à custa de um erro maior no conjunto de treino. Essas estratégias são conhecidas coletivamente como regularização [Goodfellow et al. 2016]. O Dropout oferece pouco custo computacional, porém um método poderoso para regularização de uma ampla família de modelos [Hinton et al. 2012].

Especificamente, dropout treina o conjunto consistindo através de sub-redes geradas pela remoção de unidades que fazem parte das camadas intermediarias ou de entrada, removendo uma proporção d de neurônios temporariamente com base em um atributo p, que indica a probabilidade. A remoção de neurônios pode desbalancear o aprendizado, logo o mesmo é normalizado para a quantidade de neurônios remanescente. A figura abaixo ilustra a técnica de dropout: [Goodfellow et al. 2016].

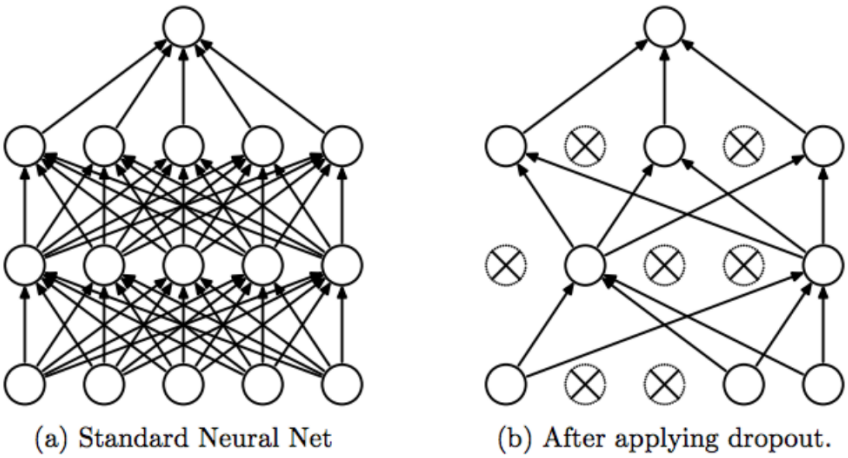

Figura 5. Dropout

3.7. Inicialização de parâmetros

Talvez a única propriedade conhecida com completa certeza é que parâmetros precisam de um início aleatório entre as diferentes unidades. Se duas unidades internas com a mesma função de ativação, estão conectadas às mesmas entradas, então essas unidades devem ter diferentes parâmetros iniciais. Caso elas tenham os mesmos parâmetros iniciais então um algoritmo de aprendizado determinístico aplicado à um modelo e função de custo também determinísticos, irá conseguintemente atualizar essas duas unidades da mesma forma. Mesmo se o modelo ou o algoritmo de treino for capaz de usar aleatoriedade para computar diferentes atualizações para diferentes unidades (por exemplo, dropout), na maioria das vezes é melhor inicializar cada unidade para computar uma função diferente de todas as outras unidades. Isso pode ajudar a garantir que nenhum padrão de entrada seja perdido no espaço nulo da forward propagation e nenhum padrão de gradiente seja perdido no espaço nulo da back-propagation. O objetivo de ter cada unidade computando uma funçao diferente motiva a inicialização aleatória de parâmetros [Goodfellow et al. 2016].

Comumente, estabelecemos constantes os valores dos biases escolhidos de maneira heurística, e inicializamos apenas os pesos aleatoriamente [Goodfellow et al. 2017].

3.8. Word Embeddings

Word embedding é o nome coletivo de um conjunto de técnicas de modelagem de linguagem e de aprendizado de recursos no processamento de linguagem natural, em que palavras ou frases do vocabulário são mapeadas para vetores de números reais. Com base na proximidade entre palavras é possível codificar-las em um conjunto de parâmetros que definam aproximadamente seu significado. Nesse espaço de palavras, é possível realizar operações, sendo que cada dimensão indica alguma característica pertencente a determinado elemento. A figura 6 ilustra como funcionam esses vetores.

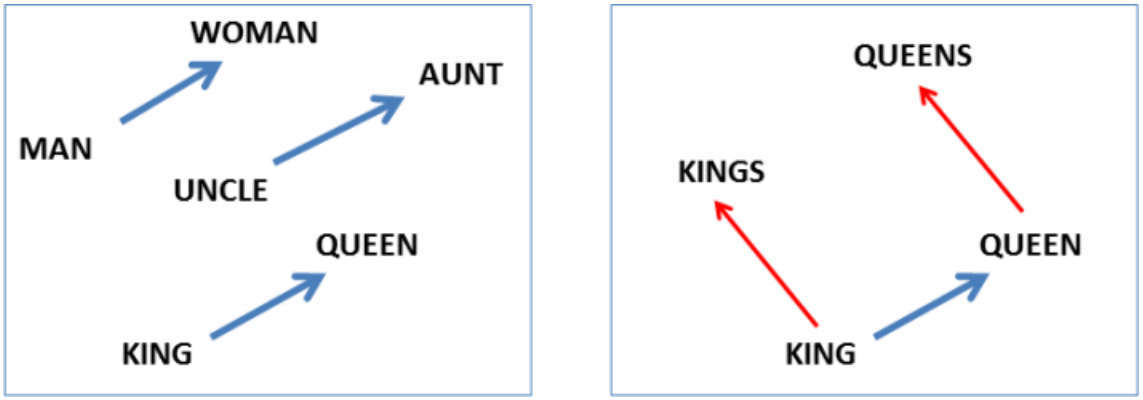

Figura 6. Representação de word embeddings. [Mikolov et al., NAACL HLT, 2013]

3.8.1. Células recorrentes bloqueadas (GRU)

Células normais de redes recorrentes não conseguem absorver grande as principais características de sentenças e sofrem de problemas como o desaparecimento do gradiente, durante o treino. As células recorrentes bloqueadas (*gated recurrent units*) apresentam uma solução para esse problema.

São definidas pelas seguintes formulas:

$$\text{Ç} < t >= tanh(Wc[\Gamma_r * c < t-1 >, x < t >] + b_c) \tag{8}$$

A equação (8) se refere ao cálculo do valor de ativação que será inserido na próxima célula recorrente bloqueada.

$$\Gamma_u = \sigma(W_u[c < t-1 >, x < t >] + b_u) \tag{9}$$

A equação (9) se refere ao *gate*, que dá nome a célula. É utilizado o valor da ativação da célula recorrente anterior e aplicada a função de ativação sigmoide, já detalhada na seção 3.3.2. A função sigmoide pode assumir apenas valores entre 0 e 1, sendo extremamente sensível aos valores de entrada. Logo essa operação tende a retornar valores ou muito próximos de 0 ou muito próximos de 1. Os resultados dirão quais elementos de c serão atualizados por Ç, ou seja, a saída de uma célula para outra, será apenas atualizada nos valores definidos por Γ_u.

Essa característica da célula recorrente bloqueada permite a memorização de características da sentença, como por exemplo, plural, gênero contexto, etc...

$$\Gamma_r = \sigma(W_r[c < t-1 >, x < t >] + b_r) \tag{10}$$

A equação (10) se refere ao *gate*, que define a relevância de c na formação da ativação substituta Ç.

$$c < t >= \Gamma_u * \text{Ç} < t > +(1 - \Gamma_u) * c < t-1) \tag{11}$$

A equação (11) define a equação completa da célula recorrente bloqueada. É possível visualizar graficamente as equações descritas anteriormente através da figura (7).

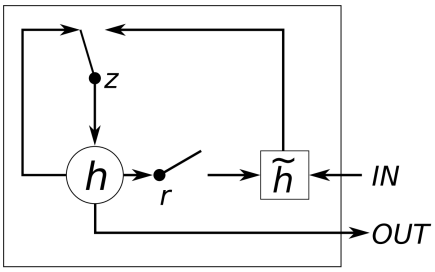

Figura 7. Esquema representativo da célula recorrente bloqueada [Chung et al. 2014]

3.8.2. Longa memória de curto termo (LSTM)

Células de longa memória de curto termo (*long-short term memory*) são definidas pelas seguintes formulas:

$$\text{Ç} < t >= tanh(Wc[a < t-1 >, x < t >] + b_c) \tag{12}$$

A equação (12) é semelhante a equação (8) da célula recorrente bloqueada, porém não há o *gate* referente a relevância da ativação anterior.

$$\Gamma_u = \sigma(W_u[c < t-1 >, x < t >] + b_u) \tag{13}$$

A equação (13) representa o *gate* que será utilizado para limitar a ativação *Ç* na equação (16). É responsável por selecionar características da célula atual.

$$\Gamma_f = \sigma(W_f[c < t-1 >, x < t >] + b_f) \tag{14}$$

A equação (14) representa o *gate* que será utilizado para limitar a ativação *c(t-1)* na equação (16).

É responsável por selecionar características da célula anterior.

$$\Gamma_o = \sigma(W_o[c < t-1 >, x < t >] + b_o) \tag{15}$$

A equação (15) representa o *gate* que será utilizado para limitar a ativação *Ç* na equação (17).

É responsável por limitar a ativação final.

$$c < t >= \Gamma_u * \text{Ç} < t > +(\Gamma_f) * c < t-1) \tag{16}$$

$$a < t >= \Gamma_o * c < t > \tag{17}$$

A equação (17) define a equação completa da célula recorrente bloqueada. É possível visualizar graficamente as equações descritas anteriormente através da figura (8).

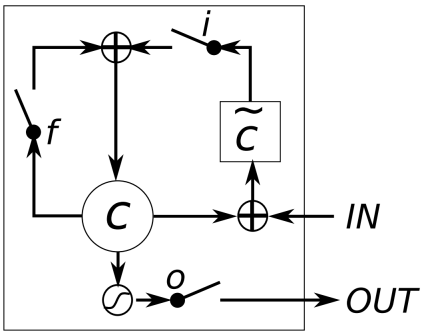

Figura 8. Esquema representativo da célula de longa memória de curto termo [Chung et al. 2014]

3.9. Bidirecionalidade

A bidirecionalidade em redes neurais recorrentes, é um conceito ao qual a adição de elementos é feito duas vezes por sentença, a primeira vez em ordem normal e a segunda em ordem contrária (descrito pelo autor de forma positiva e negativa). Com esse conceito é removida a limitação de prever apenas a janela futura. A principal contribuição da bidirecionalidade, é adquirir o contexto de palavras no início da sentença. [Schuster and Paliwal 1997].

4. Arquiteturas Propostas

Um modelo inicial foi definido rapidamente para obter o primeiro resultado, a seguir utilizamos a técnica de iteração. A técnica de iteração, consiste na ideia de definir uma arquitetura inicial, e definir hiperparâmetros rapidamente.

4.1. *Word Embeddings*

A matrix para representação de palavras indica os elementos de entrada de cada sentença, esse provém da técnica de Word Embeddings, detalhada na seção 3.7. Nesse trabalho foram utilizados vetores pré-treinados *GloVe* [Pennington et al. 2014], treinados em um *crawler* com aproximadamente 840 bilhões de tokens.

4.1.1. Pooling

As operações de *pooling* são outro importante bloco de construção nas CNNs. Operações de *pooling* reduzem o tamanho de mapas de características (*feature maps*) usando alguma função para resumir sub-regiões, como a média ou o valor máximo. O agrupamento funciona deslizando uma janela pela entrada e alimentando o conteúdo da janela para uma função de pooling. De certa forma, o pooling funciona como uma convolução, mas substitui a combinação linear descrita pelo kernel com alguma outra função. A figura 9 fornece um exemplo de pooling máximo (*max pool*) [Dumoulin and Visin 2016].

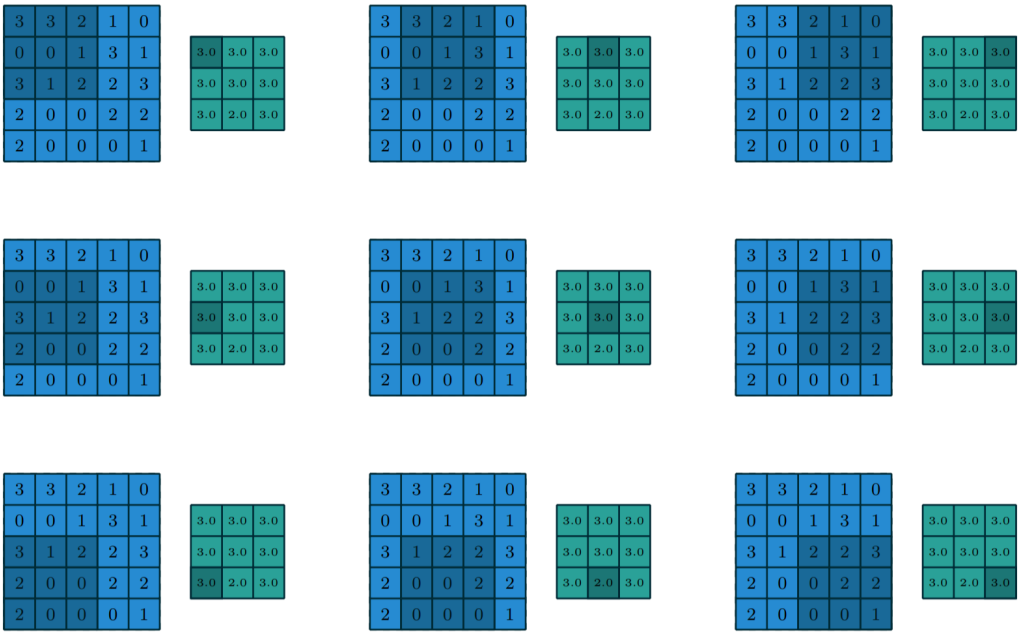

Figura 9. Cálculo a saída de uma operação de pooling máximo. [Dumoulin and Visin 2016]

4.1.2. Célula

As células utilizadas mudam o compartamento da rede neural recorrente, sendo que os casos utilizados nesse trabalho foram células recorrentes bloqueadas ou células de longa memória de curto termo, já definidas nas seções 3.8.1 e 3.8.2.

4.1.3. Predição

Predição faz uma rede neural totalmente conectada e utiliza a função de ativação sigmoide, já descrita na seção 3.3.2, sendo que retornos maiores iguais a 0.5 serão considerados verdadeiros, e menores como falso.

4.2. Rede recorrente

Apresenta células em sequência, formando uma rede recorrente. A camada de *embeddings* transforma a entrada de dados em matrizes de características, sendo que o *dropout* tem o objetivo de normalizar e evitar sobreajuste.

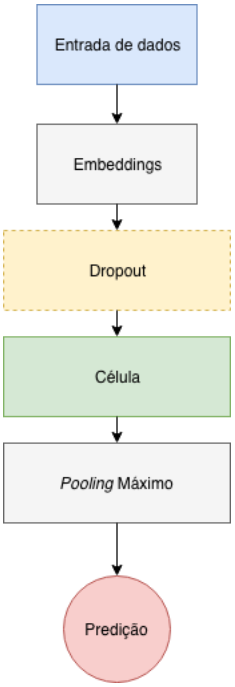

Figura 10. Rede recorrente simples

4.3. Rede recorrente com convolução

Apresenta a adição de câmadas convolucionais e de *pooling*, com a intenção de relacionar características de uma dimensão extraidas pela rede recorrente.

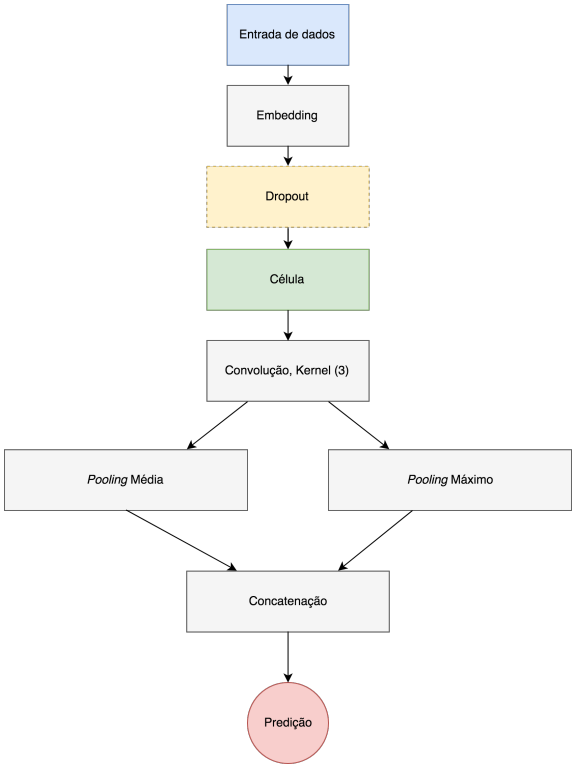

Figura 11. Rede recorrente com convolução.

4.3.1. Convolução

Uma convolução discreta é uma transformação linear que preserva uma noção de área. É esparso (apenas algumas unidades de entrada contribuem para uma determinada saída) e reutiliza parâmetros (os mesmos pesos são aplicados a vários locais na entrada). A Figura 12 fornece um exemplo de uma convolução discreta. A grade azul claro é chamada de mapa de características (*feature map*) de entrada. Para manter o desenho simples, uma única entrada de mapa de características é representado, mas não é incomum ter vários empilhados um sobre o outro.

Um kernel (área sombreada) desliza pelo mapa de características de entrada. Em cada local, o produto entre cada elemento do kernel e o elemento de entrada que ele sobrepõe é computado e os resultados são somados para obter a saída na localização atual. o procedimento pode ser repetido usando kernels diferentes para formar o recurso de saída mapas conforme desejado. Os resultados finais deste procedimento são chamados mapas de recursos de saída. [Dumoulin and Visin 2016]

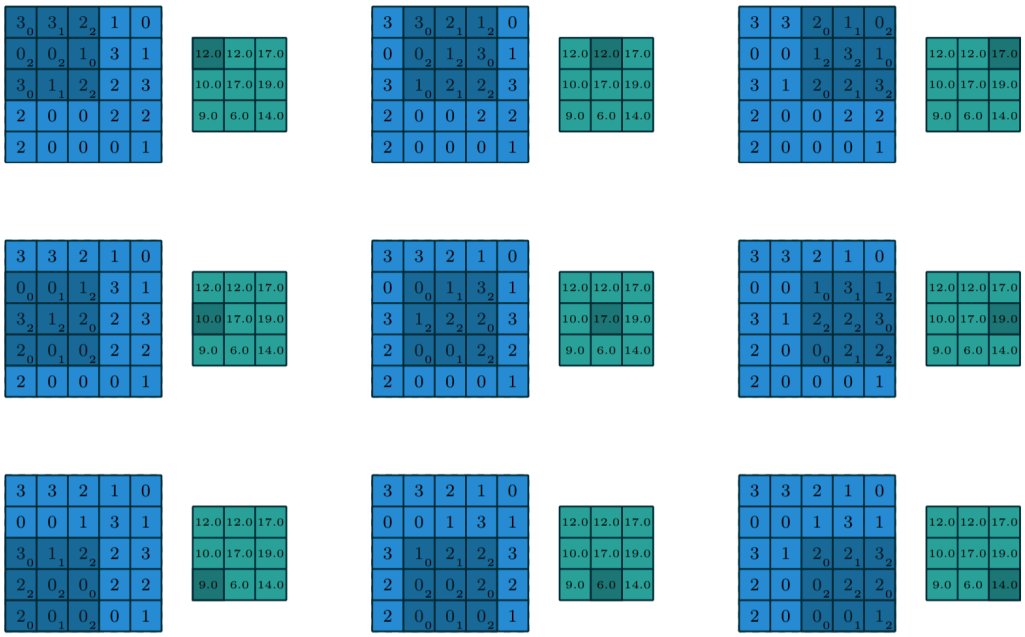

Figura 12. Cálculo a saída de uma operação de convolução. [Dumoulin and Visin 2016]

4.4. Hiper-parametros

Embora a arquitetura do modelo contenha grande parte da informação sobre nossos modelos (seção 4.2 e 4.3), ainda é necessário definir alguns hiper-parâmetros que farão o comportamento de treino e arquitetura de nossa rede recorrente.

- Durante a formação de matriz de *embeddings* é possível definir a quantidade de parâmetros que cada palavra armazenará, foi escolhida de maneira arbitrária o valor de 300, que é compatível com matrizes já treinadas como por exemplo o *GloVe* [Pennington et al. 2014]

- O número de nós na rede recorrente não possui uma métrica de escolha, foi escolhido com base no método iterativo, o valor de 256 nós.

- Regularização: Para evitar sobreajuste foi utilizada a técnica de *dropout* com probabilidade *p* de remover um nó de 0.5.

- O tamanho do *minibatch* escolhido foi 128, esse valor se restringe ao tamanho da memória da placa gráfica utilizada para treino. Logo esse valor provou-se ideal em nosso hardware.

- O número de épocas escolhido para treinamento foi cinco, devido a limitação de hardware e quantidade de modelos a serem treinados, esse número demonstrou-se suficiente para garantir a convergência do modelo e evitar o ajuste exagerado, garantindo generalização.

- A taxa de aprendizado durante cada adição do gradiente por *minibatch* foi escolhida arbitrariamente e de maneira iterativa, obtendo-se o valor de 10^{-3}.

Os hiperparâmetros do otimizador foram definidos através dos valores padrões indicados [Kingma and Ba 2014]. A taxa de aprendizado utiliza do processo de iteração, e o valor utilizado nas análises preliminares é arbitrário, sendo inicialmente 10^{-3}. Os pesos das células foram inicializador através da inicialização Glorot [Glorot and Bengio 2019]).

4.5. Organização dos dados

O banco de dados utilizou da técnica de separação de treino e teste, onde 90% dos dados foi atribuída para treino e 10% para validação. Não existe uma métrica fixa para a definição da quantidade a ser repartida, porém essa deve ser feita de acordo com a quantidade de dados rotulados. Como a validação segue apenas para verificar se não está acontecendo sobreajuste aos dados de treinamento, foi escolhido 10% como dos hiperparametros. Essa técnica é conhecida como holdout. Em abordagens clássicas de aprendizado de máquina, normalmente é utilizada a validação cruzada, porém, em casos de aprendizagem profunda (*deep learning*) isso se torna inviável devido a quantidade de processamento, já que nesse tipo de validação é necessário treinar o modelo *k* vezes, onde *k* é um hiperparâmetro. Como modelos de aprendizagem profunda (*deep learning*) possuem grande volume de dados, utilizaremos o holdout devido a viabilidade computacional.

A segmentação a seguir representa como os dados foram separados:

Figura 13. Separacão de dados

A segmentação de testes não está inclusa nos dados públicos. Tal teste é feita através de dados não rotulados na plataforma Kaggle. É possível realizar um submissão com os rotulos e obter a acurácia. Essa segmentação não é de acesso público para evitar que modelos sobreajustem seus parâmetros, assim o resultado retornado garante a validade de generalização do modelo.

5. Experimentos

5.1. Resultados

Em primeira instância os modelos obtiveram resultados de treino, validação e teste coerentes. Sendo possível visualizar que os resultados foram próximos indicando a ausência de sobreajuste.

Modelo monolítico, treinado por 5 épocas			
Célula	Acurácia Treino	Acurácia Validação	Acurácia Teste
RNN	97,99%	98,19%	97,38%
GRU	98,32%	98,35%	98,09%
LSTM	98,29%	98,41%	98,12%
GRU Bidirecional	98,37%	98,38%	98,15%
LSTM Bidirecional	98,37%	98,40%	98,22%
GRU + Convolução	98,37%	98,31%	98,22%
LSTM + Convolução	98,35%	98,42%	**98,25**%
GRU Bidirecional + Convolução	**98,41**%	98,44%	98,18%
LSTM Bidirecional + Convolução	**98,41**%	**98,48**%	98,15%

Tabela 1. Resultados

5.2. Discussão

Podemos observar que nesse conjunto de dados a bidirecionalidade não trás beneficios nestas instâncias. A convolução trouxe ganhos em todos os casos sem bidirecionalidade. Ainda é possível analisar que os modelos bidirecionais e com convolução tiveram os melhores resultados no treino e validação, porém não se generalizaram no caso de teste, tal situação demonstra sobreajuste por parte desses modelos nos dados de treino e validação.

Análises utilizando modelos de linguagem e transferência de aprendizado ainda estão sendo desenvolvidas.

O repositório com códigos fontes de todos os modelos analisados está disponível no Github, através do endereço: `https://github.com/pstwh/toxic-comments-keras`. Todos os modelos foram treinados pelos autores deste trabalho e seus parâmetros estão disponibilizados. É recomendado a utilização de placa gráfica para reprodução desses artigos. As configurações utilizadas pelos autores para treinamento e validação dos modelos: Placa gráfica NVIDIA Quadro M4000, 30 Gb memória RAM e processador Intel® Xeon® E5-2623 v4 8 núcleos.

6. Considerações Finais e Trabalhos Futuros

Nesse trabalho apresentamos métodos automátizados para análise de comentários tóxicos utilizando redes recorrentes e suas variações. O conjunto de dados utilizado foi adquirido da plataforma *Kaggle*, onde foram tratados e processados com a utilização de *word embeddings* provenientes da *GloVe* [Pennington et al. 2014].

A comparação dos modelos testados demonstra como a combinação de modelos recorrentes e camadas convolucionais melhoram a acurácia. Técnicas como transferência

de aprendizado em modelos de linguagem não foram aplicadas nesse trabalho, porém há ambição de aplica-los em trabalhos futuros. Há a possibilidade de reelaborar a célula que constrói a rede neural recorrente para atender melhor a este caso. Talvez a aplicação de uma rede de capsulas [Sabour et al. 2017] proporcionaria um resultado superior devido a não variação em translações, que é uma das características desta arquitetura.

Existe a ambição em trabalhos futuros de aplicar conceitos presentes nesse artigo em outros conjuntos de dados, como por exemplo conjuntos de dados em português do brasil, com ajuda de aprendizagem semi-supervisionada e transferência de aprendizado.

Referências

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Glorot, X. and Bengio, Y. (2019). Understading the difficulty of training deep deed-forward neural networks.

Golik, Pavel Doetsch, P. . N. H. (2013). Cross-entropy vs. squared error training: a theoretical and experimental comparison.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*.

Howard, J. and Ruder, S. (2018). Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

Karparthy, A. (2015). The unreasonable effectiveness of recurrent neural networks.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Ma, S., Sun, X., Wang, Y., and Lin, J. (2018). Bag-of-words as target for neural machine translation. *CoRR*, abs/1805.04871.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Pascanu, R., Gülçehre, Ç., Cho, K., and Bengio, Y. (2013). How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Ramos, J. et al. (2003). Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. *CoRR*, abs/1710.09829.

Schaul, T., Antonoglou, I., and Silver, D. (2013). Unit tests for stochastic optimization. *CoRR*, abs/1312.6055.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Wang, S. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification.

Y. LeCun, L. Bottou, Y. B. and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.