Pontifícia Universidade Católica
do Rio de Janeiro

Renato Sayão Crystallino da Rocha

Um filtro para arcos em Árvores de Dependência

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio.

Orientador: Prof. Ruy Luiz Milidiú

Rio de Janeiro
Setembro de 2018

Pontifícia Universidade Católica
do Rio de Janeiro

Renato Sayão Crystallino da Rocha

Um filtro para arcos em Árvores de Dependência

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Ruy Luiz Milidiú
Orientador
Departamento de Informática – PUC-Rio

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática – PUC-Rio

Prof. Leonardo Alfredo Forero Mendoza
UERJ

Prof. Márcio da Silveira Carvalho
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 26 de Setembro de 2018

Renato Sayão Crystallino da Rocha

Graduou-se em Engenharia de Computação na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Participou do desenvolvimento do sistema OCTOPUS junto ao Laboratório de Inteligência Computacional Aplicada da PUC–Rio, hoje em funcionamento na PETROBRAS. Atua como pesquisador e desenvolvedor na área de métodos de apoio a decisão.

Ficha Catalográfica

Sayão Crystallino da Rocha, Renato

Um filtro para arcos em Árvores de Dependência / Renato Sayão Crystallino da Rocha; orientador: Ruy Luiz Milidiú. – Rio de janeiro: PUC-Rio, Departamento de Informática , 2018.

v., 60 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática .

Inclui bibliografia

1. Informática – Teses. 2. Long Short-Term Memory;. 3. Redes Neurais Recorrentes;. 4. Árvores de Dependência;. 5. Classe Gramatical;. 6. Classificação;. I. Milidiú, Ruy Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática . III. Título.

CDD: 004

Agradecimentos

Agradeço ao meu orientador, Prof. Ruy Luiz Milidiú pela sua atenciosidade e paciência durante o decorrer deste trabalho.

Ao meu pai e a meus leais amigos, pois a fé de vocês em mim muitas vezes é a única fonte da minha força de vontade.

Aos meus fiéis companheiros ao redor do país, pelas inesquecíveis aventuras compartilhadas dia após dia.

Agradeço também a PUC-Rio pelo apoio e auxílios concedidos que possibilitaram este trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Resumo

Sayão Crystallino da Rocha, Renato; Milidiú, Ruy Luiz. **Um filtro para arcos em Árvores de Dependência**. Rio de Janeiro, 2018. 60p. Dissertação de Mestrado – Departamento de Informática , Pontifícia Universidade Católica do Rio de Janeiro.

A tarefa de Processamento de Linguagem Natural consiste em analisar linguagens naturais de forma computacional, facilitando o desenvolvimento de programas capazes de utilizar dados falados ou escritos. Uma das tarefas mais importantes deste campo é a Análise de Dependência. Tal tarefa consiste em analisar a estrutura gramatical de frases visando extrair aprender dados sobre suas relações de dependência. Em uma sentença, essas relações se apresentam em formato de árvore, onde todas as palavras são interdependentes. Devido ao seu uso em uma grande variedade de aplicações como Tradução Automática e Identificação de Papéis Semânticos, diversas pesquisas com diferentes abordagens são feitas nessa área visando melhorar a acurácia das árvores previstas. Uma das abordagens em questão consiste em encarar o problema como uma tarefa de classificação de *tokens* e dividi-la em três classificadores diferentes, um para cada sub-tarefa, e depois juntar seus resultados de forma incremental. As sub-tarefas consistem em classificar, para cada par de palavras que possuam relação pai-dependente, a classe gramatical do pai, a posição relativa entre os dois e a distância relativa entre as palavras. Porém, observando pesquisas anteriores nessa abordagem, notamos que o gargalo está na terceira sub-tarefa, a predição da distância entre os *tokens*. Redes Neurais Recorrentes são modelos que nos permitem trabalhar utilizando sequências de vetores, tornando viáveis problemas de classificação onde tanto a entrada quanto a saída do problema são sequenciais, fazendo delas uma escolha natural para o problema. Esse trabalho utiliza-se de Redes Neurais Recorrentes, em específico Long Short-Term Memory, para realizar a tarefa de predição da distância entre palavras que possuam relações de dependência como um problema de classificação *sequence-to-sequence*. Para sua avaliação empírica, este trabalho segue a linha de pesquisas anteriores e utiliza os dados do *corpus* em português disponibilizado pela Conference on Computational Natural Language Learning 2006 Shared Task. O modelo resultante alcança 95.27% de precisão, resultado que é melhor do que o obtido por pesquisas feitas anteriormente para o modelo incremental.

Palavras-chave

Long Short-Term Memory; Redes Neurais Recorrentes; Árvores de Dependência; Classe Gramatical; Classificação;

Abstract

Sayão Crystallino da Rocha, Renato; Milidiú, Ruy Luiz (Advisor). **A Dependency Tree arc filter**. Rio de Janeiro, 2018. 60p. Dissertação de Mestrado – Departamento de Informática , Pontifícia Universidade Católica do Rio de Janeiro.

The Natural Language Processing task consists of analyzing the grammatical structure of a sentence written in natural language aiming to learn, identify and extract information related to its dependency structure. This data can be structured like a tree, since every word in a sentence has a head-dependent relation to another word from the same sentence. Since Dependency Parsing is used in many applications like Machine Translation, Semantic Role Labeling and Part-Of-Speech Tagging, researchers aiming to improve the accuracy on their models are approaching this task in many different ways. One of the approaches consists in looking at this task as a token classification problem, using different classifiers for each sub-task and joining them in an incremental way. These sub-tasks consist in classifying, for each head-dependent pair, the Part-Of-Speech tag of the head, the relative position between the two words and the distance between them. However, previous researches using this approach show that the bottleneck lies in the distance classifier. Recurrent Neural Networks are a kind of Neural Network that allows us to work using sequences of vectors, allowing for classification problems where both our input and output are sequences, making them a great choice for the problem at hand. This work studies the use of Recurrent Neural Networks, in specific Long Short-Term Memory networks, for the head-dependent distance classifier sub-task as a sequence-to-sequence classification problem. To evaluate its efficiency, this work follows the line of previous researches and makes use of the Portuguese corpus of the Conference on Computational Natural Language Learning 2006 Shared Task. The resulting model attains 95.27% precision, which is better than the previous results obtained using incremental models.

Keywords

Long Short-Term Memory; Recurrent Neural Network; Dependency Trees; Part-Of-Speech; Classification;

Sumário

Lista de figuras

Lista de tabelas

Lista de Abreviaturas

ML – *Machine Learning*
DL – *Deep Learning*
RNN – *Recurrent Neural Network*
BRNN – *Bidirectional Recurrent Neural Network*
LSTM – *Long Short-Term Networks*
CoNLL – *Conference on Computational Natural Language Learning*
NLP – *Natural Language Processing*
DP – *Dependency Parsing*
DT – *Dependency Tree*
SRL – *Semantic Role Labeling*
UAS – *Unlabelled Attachment Score*
IFIS – *Incremental Feature Induction and Selection*
S-IFIS – *Incremental Feature Induction and Selection*
LEARN – Laboratório de Engenharia de Algoritmos e Redes Neurais

1
Introdução

Com o avanço da tecnologia nas últimas décadas, houve também um crescimento no quão facilmente informações de todas as formas podem ser encontradas e guardadas. Esses dados - que podem possuir diferentes formatos como texto, vídeos e áudios - no geral se encontram em uma forma não-estruturada (9), ou seja, não possuem nenhuma organização ou rótulos que ajudem em sua utilização. A falta de organização e os enormes tamanhos em tais bases de dados resultam em problemas na hora de buscar e extrair informações. Isso resultou no crescimento de áreas de pesquisa como Visão Computacional para imagens e Processamento de Linguagem Natural para textos e áudios.

Processamento de Linguagem Natural, ou *Natural Language Processing* (NLP), se refere ao conjunto de técnicas computacionais que, combinadas com informações linguísticas, permitem que computadores representem e utilizem conhecimentos expressados em frases de linguagem natural (3). Ou seja, o objetivo de NLP é entender e extrair informação de textos e áudios de forma computacional. Essa técnica pode ser fragmentada em tarefas mais simples como Tokenização e identificação de classes gramaticais, ou em mais complexas como *Semantic Role Labeling* e *Hedge Detection*. (9)

Devido a esses fatores, diversas técnicas de *Machine Learning* (ML) são aplicadas a diferentes tarefas da área. ML nos permite trabalhar em cima de *corpora* complexos independente da língua do texto original e, mais importante, de forma pouco dependente em conhecimentos linguísticos (9).

No entanto, os algoritmos de ML em questão funcionam melhor com grandes quantidades de dados e necessitam de diversas informações ou rótulos. Isso pode ser um problema, já que geralmente tais dados precisam ser extraídos manualmente. No entanto, graças a diversas iniciativas de comunidade NLP para a construção de *corpus* anotados, como a *Conference on Natural Language Learning* (CoNLL) *Shared Tasks*, esse problema pode ser contornado.

1.1 Dependency Parsing

Uma das aplicações de NLP consiste no aprendizado de estruturas complexas presentes em uma frase, como sequências ou grafos. Dentre os problemas abordados por NLP, há o de *Dependency Parsing*, ou Análise de Dependência, tarefa cujo foco é modelar relações de dependência intrínsecas a uma frase de linguagem natural através da predição de grafos ou árvores de dependência.

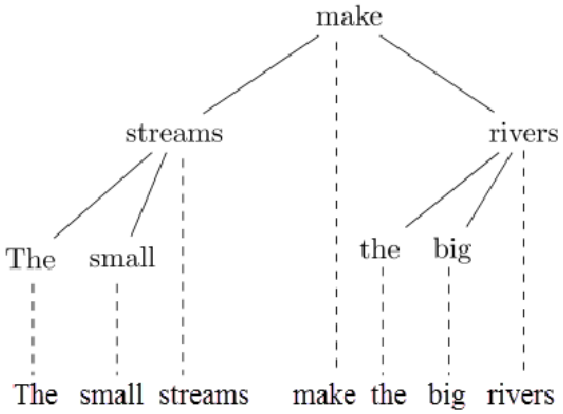

Figura 1.1: Exemplo de uma Árvore de Dependência.(18)

Na tarefa de *Dependency Parsing*, os nós de uma árvore são as palavras de uma frase e a presença de uma aresta as conectando representa uma relação de dependência. Em uma dessas relações, uma palavra representa um *token* dependente e está ligado a um outro *token*, do qual ele depende. Essas conexões são direcionadas por natureza, resultando então em um grafo direcionado e acíclico. Elas também são chamadas de *head-dependent*.

Tabela 1.1: Melhores *parsers* de dependência para o português.

Parser	**Ano**	**Algoritmo de Aprendizagem**	**UAS (%)**
Koo et al. (21)	2010	MIRA	93.03
Milanés (3)	2016	S-IFIS	92.96
Martins et al. (23)	2010	MIRA	92.71
Fernandes. (12)	2012	ESL	92.66
Motta (25)	2014	IFIS	92.01
Crestana (9)	2010	ETL	89.74

Existem diversas formas de encarar o problema de *Dependency Parsing*, como os métodos *Transition-Based* e *Graph-Based*. Porém, para este trabalho, estamos interessados na abordagem *Token-Based*. Neste método, ao invés do

parsing ser feito através de transições ou grafos, o modelo classifica cada palavra de uma sentença de forma a identificar o seu *head*, ou seja, a outra palavra da frase, com a qual a palavra em questão possui uma relação de dependência. Após isso, a tarefa é então tratada como um problema de classificação de *tokens*.

Essa classe ou *tag*, possui informações sobre a orientação, a classe gramatical e a distância entre duas palavras relacionadas. Ela é composta pela concatenação de um *contador de distância*, a *classe gramatical do head* e se o *head* vem antes (*Left*) ou depois (*Right*) na frase. Portanto, se um token é de classe *1_v_R*, isso significa que seu *head* é o "primeiro verbo para esquerda", *2_v_L* significaria que seu *head* seria o segundo verbo para direita e *1_prp_L* é a primeira preposição para esquerda. (9)

Id	Word	Part-of-Speech	Head	Special Tagset
1	É	adv	9	2_v_R
2	por	prp	9	2_v_R
3	isso	pron	2	1_prp_L
4	que	adv	9	2_v_R
5	,	punc	6	1_v_R
6	diz	v	0	root
7	,	punc	6	1_v_L
8	não	adv	9	1_v_R
9	tem	v	6	1_v_L
10	pena	n	9	1_v_L
11	de	prp	10	1_n_L
12	Bill	prop	11	1_prp_L
13	.	punc	6	2_v_L

Figura 1.2: Dependência na abordagem *Token Based*. (9)

O problema desta abordagem pode ser evidenciado ao analisarmos a base de dados. O *corpus* utilizado conta com 14 classes gramaticais, frases de até 125 palavras de comprimento e *heads* que podem estar posicionados para qualquer lado de uma palavra. Isso resulta em até 700 possíveis classes diferentes, que, somado ao fato de possuirmos mais de 26000 palavras, pode causar problemas na etapa de classificação.

No entanto, a modelagem proposta traz também uma vantagem. Como cada *tag* é composta por três componentes, a tarefa de predição pode ser decomposta em tarefas menores. Ao invés de haver apenas uma etapa de classificação responsável por mapear um grande vocabulário em uma grande quantidade de classes, o laboratório LEARN provou (3) que ótimos resultados podem ser obtidos ao treinarmos diversos classificadores e os utilizar de forma

Tabela 1.2: Contagem de cada *tag* única.

Tag	**Únicos**
Part-Of-Speech	14
Side	2
Distance	25

incremental. Uma análise das classes também torna evidente uma divisão natural da tarefa, dado que cada *tag* é composta por três informações.

1.2 Objetivos

A tarefa de predição de uma árvore de dependência pode ser modelada como um problema de predição estruturado cujo domínio do *output* consiste na combinação de todas as possíveis árvores de dependência para uma dada sentença. Mesmo na abordagem *token-based*, cada palavra ainda pode ser classificada como qualquer *tag* existente derivada de uma combinação de diversas possíveis distâncias e classes gramaticais. Essa esparsidade do contradomínio, como demonstrada na Figura 1.3 (3), dificulta a etapa de prever corretamente tais árvores, principalmente para frases maiores.

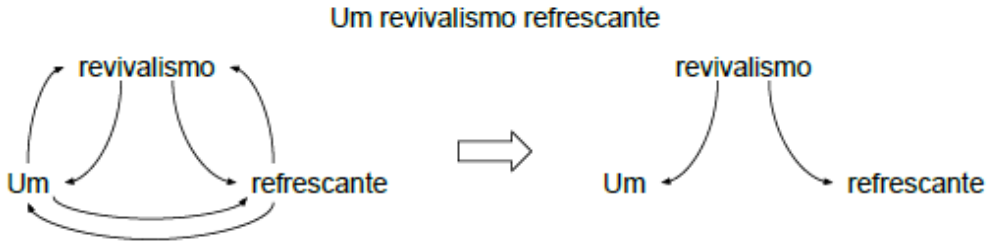

Figura 1.3: Representação visual do contradomínio.

Diversos trabalhos anteriores desenvolvidos pelo grupo de pesquisas do *Laboratório de Engenharia de Algoritmos e Redes Neurais* (LEARN) utilizaram o paradigma *token-based* com sucesso. No entanto identificou-se que um dos maiores gargalos consiste em encontrar a distância entre duas palavras que possuam uma relação de dependência, principalmente em frases grandes. Este trabalho propõe a construção de um filtro de arestas para o problema de *Dependency Parsing* em frases do português através da utilização de conhecimentos da linguagem.

1.2.1 Objetivo Principal

Este trabalho tem como objetivo a construção de um filtro de arestas responsável por reduzir o contradomínio no problema de predição de árvores de dependência em abordagens baseadas em *tokens*. Esse filtro é capaz de, dado informações sobre a classe gramatical da palavra e de seu *head*, dizer com alta acurácia a distância entre duas palavras que possuem uma relação de dependência.

1.2.2 Objetivos Parciais

Para alcançarmos o objetivo principal, os seguintes objetivos secundários devem ser obtidos:

1. propor uma nova *tag* composta por informações obtidas em trabalhos anteriores do laboratório LEARN[1];

2. treinar uma rede neural recorrente capaz de predizer a distância de um *token* para seu *head*, dado como entrada *arrays* contendo as *tags* propostas para qualquer dada frase;

3. utilizar a base de dados e métrica do Bosque, o *corpus* de Português disponibilizado pela CoNLL Shared Task (6, 1, 14) para avaliar e comparar o filtro proposto com os resultados obtidos em sistemas propostos anteriormente pelo LEARN.

1.3 Contribuições

As principais contribuições desta dissertação são:

– utilização de *machine learning* para treinar um filtro de alta acurácia com alta velocidade e desempenho;
– implementação de um modelo utilizando Redes Neurais Recorrentes capaz de realizar predições sobre a frase de forma sequencial;
– um filtro de arestas capaz de superar o estado-da-arte atual, obtendo uma acurácia de 94.92% utilizando a métrica UAS;
– o filtro criado pode ser utilizado incrementalmente com outras linhas de pesquisa do LEARN, de forma a trazer melhoras de desempenho e acurácia para o estado atual da pesquisa.

[1]http://www.inf.puc-rio.br/blog/laboratorio/learn/

1.4 Organização da Dissertação

Esta dissertação é estruturada da seguinte maneira. No capítulo 2, descrevemos com maior detalhes a tarefa Análise de Dependência, suas diferentes abordagens e a base de dados utilizada. No capítulo 3, explicamos a abordagem escolhida para o problema, também entrando em detalhes sobre a extração e o significado de cada atributo utilizado. O capítulo 4 é responsável por mostrar a estrutura da rede utilizada, descrevendo cada uma das múltiplas camadas do modelo ML construído neste trabalho. O capítulo 5 se encarrega de analisar o *corpus* utilizado, descrever a métrica aplicada e o mostrar classificador *baseline* e avaliar os resultados do modelo treinado. Por último, o capítulo 6 finaliza este trabalho, listando nossas conclusões assim como trabalhos futuros.

2
Dependency Parsing

Neste capítulo explicaremos alguns dos termos necessários para compreensão do problema em questão. Em específico é explicado o que são conceitos como Árvores e *Parsers* de Dependência, as diferentes abordagens nessa tarefa e o *corpus* utilizado.

2.1
Análise de Dependência

A tarefa de análise de dependência, do inglês *Dependency Parsing*, consiste em identificar e analisar estruturas gramáticas de uma sentença de forma a reconhecer suas relações de dependência.(8) Para uma frase de linguagem natural - no nosso caso o Português - cada palavra de uma sentença possui uma relação de dependência com alguma outra palavra. Caso haja uma conexão entre as duas palavras, damos o nome de "Dependente", ou *dependent*, para a palavra que possui uma dependência em outra e "Pai", ou *Head*, para a palavra que possui dependentes.

Cada palavra - ou *token* - poderá ter um e somente um pai, porém cada pai pode ter uma quantidade [0, N] de filhos onde N é o tamanho da sentença. A imagem abaixo representa visualmente as relações *head-dependent* para a frase do português *"É por isso que, diz, não tem pena de Bill."*. Cada seta conectando duas palavras significa uma relação *head-dependent*, apontando do *token* pai para o seu dependente. Caso uma palavra não possua nenhuma outra palavra que seja seu pai, dizemos que o seu *head* é *"ROOT"*, ou seja, a raiz da sentença.

Figura 2.1: Estrutura de dependência.

2.2 Árvores de Dependência

Uma das formas de representarmos estruturalmente as relações de dependência de uma frase é através de grafos cujos nós são as palavras de uma frase e as arestas são as relações de dependência.

Para uma representação completa de uma sentença, o grafo deve seguir três restrições. Primeiramente ele deve ser conexo (24), pois nenhuma palavra em uma frase não possui relações com as demais. A maioria das gramáticas de dependência seguem algo chamado de *single-head constraint*, onde cada nó é conectado a no máximo um *head*. Por último o grafo deve ser acíclico, ou seja, um nó não pode ter como *head* um nó que já possua um caminho até ele.

Devido à natureza *one-to-many* do problema em questão, o grafo que representa as relações pai-dependente de uma frase assume a forma de uma árvore. Ou seja, cada *head token* - ou *token* pai - pode se relacionar com um ou mais *tokens* dependentes mas cada dependente só poderá se relacionar com um *head token*.

Uma árvore é uma estrutura de dados composta por nós, arestas e uma raiz. Cada palavra de uma sentença em português estará conectada ao seu ancestral por uma aresta. Nesta representação em Árvore, as palavras são os nós e as arestas são as relações de dependência entre elas. O nó que não possuir conexão de dependência com nenhum outro nó é ligado na raiz da árvore. A estrutura em questão, como formalizada por (3), é representada abaixo.

Figura 2.2: Árvore de dependência.

Seja $x = (x_0, x_1, ..., x_n)$ uma sequência de entrada onde x_i é a i-ésima palavra do conjunto x, e x_0 é um *token* artificial para representar a raiz da sentença. A estrutura sintática associada à x é um grafo direcionado $G = (V, A)$ onde há um nó i para cada *token* em x, ou seja $V \subseteq \{0, 1, ..., n\}$; e diversas arestas $(i, j)\epsilon A$ que representam relações de dependência. Caso uma aresta $(m, n)\epsilon A$, existe uma relação *head-dependent* entre o *token* pai x_m e o dependente x_n.

Por relações de dependência possuírem uma interpretação próxima de relações semânticas (9), árvores de dependência possuem diversas aplicações em áreas como *Question Answering* (17, 35, 26, 10), Tradução Automática (28, 11, 32), Extração de Conhecimento (7, 30, 2) e traz grandes benefícios na tarefa de *Semantic Role Labeling* (15).

Sendo assim, cada sentença da língua portuguesa pode ter suas relações de dependência representadas por um grafo estruturado em forma de árvore. A imagem acima demonstra a estrutura de árvore formada pela frase *"É por isso que, diz, não tem pena de Bill."* abordada anteriormente .

2.3 Data-Driven Dependency Parsing

Dependency Parsing é a tarefa de achar um grafo de dependência de uma frase de entrada de acordo com um *dependency grammar* (9). Este trabalho foca no que é chamado de *data-driven dependency parsing*. Esse paradigma consiste em construir grafos de dependência para sentenças de um *corpus* com anotações ao invés de analisar estruturas gramaticais como em *Grammar-Driven Dependency Parsing*.

O uso de modelos *data-driven* para tarefas de NLP se torna cada vez mais viável com a maior disponibilidade de *corpora* com anotações de dependência, que por sua vez causa um aumento de atividade nessa área de pesquisa. Este trabalho usufrui disso e utiliza como base de dados diversas frases da língua portuguesa com anotações disponibilizada pela CoNLL (*Conference on Natural Language Learning Shared Task* (13).

2.3.1 Transition-Based Dependency Parsing

Modelos *transition-based* utilizam *parsers* determinísticos para construir as estruturas de dependência dos dados. O *parser* em questão é capaz de construir árvores de dependência ao ler as palavras de uma sentença da esquerda para direita. Esse método utiliza dois vetores. O primeiro é chamado de *buffer* e é inicializado com toda a sentença. O segundo é o vetor de *stack*,

inicializado sem nenhum elemento. O modelo então pode realizar três ações diferentes.

- Shift: Responsável por transferir a próxima palavra, em ordem de leitura, do *buffer* para o *stack*.
- Left-Arc: Adiciona uma relação de dependência entre as duas palavras mais recentes de *stack*, onde a palavra do topo é o *token* dependente. Após feito isso, remove a segunda palavra do topo de *stack*.
- Right-Arc: Adiciona uma relação de dependência entre as duas palavras mais recentes da *stack*, onde a palavra do topo é o *head*. Após feito isso, remove a palavra do topo de *stack*.

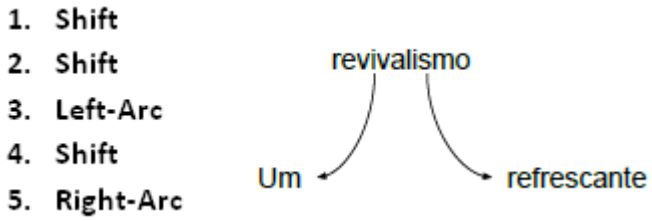

Figura 2.3: Passos do *parsing* baseado em transição.

2.3.2 Graph-Based Dependency Parsing

Algoritmos *graph-based* tem sua metodologia mais diretamente ligada com a forma estrutural de grafo que relações de dependência de uma frase apresentam, diferentemente da abordagem anterior. Modelos baseados em transição identificam relações de dependência de forma mais local, dado que realizam o *parser* da frase palavra por palavra. A abordagem baseada em grafo, no entanto, foca em achar o melhor grafo de dependência global para cada frase.

Neste método, o modelo começa com um grafo completamente conectado. Cada aresta deste grafo possui um peso. O modelo então tenta encontrar a *minimum spanning tree* do grafo resultante, ou seja, a árvore com menos aresta que conecte todos os pontos e, ao mesmo tempo, maximize a soma dos pesos das arestas nela contida.

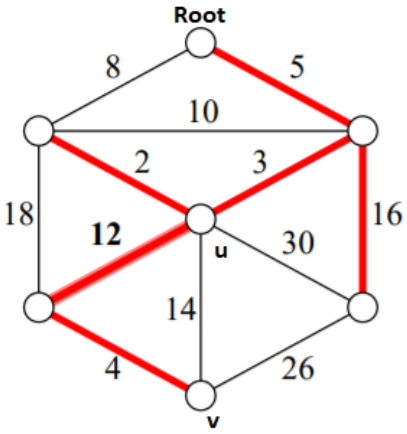

Figura 2.4: Exemplo de *MST*

2.3.3 Token-Based Dependency Parsing

De todas as abordagens que utilizam técnicas de ML em NLP, a classificação de *tokens* é definitivamente a mais comum (9). Abordagens baseadas em *tokens* se diferem bastante das opções listadas anteriormente. Nesta proposta, o problema de análise de dependência é tratado como um problema de classificação de *tokens*. Uma classe é atribuída para cada palavra de uma sentença e o objetivo do modelo é classificar corretamente cada palavra.

Essas classes podem ser baseadas em vários aspectos da linguagem e devem ser escolhidas de acordo com a natureza do problema a ser resolvido e é essencial que elas sejam selecionadas de forma a melhor retratar um problema. Para tarefas mais simples como *Part-Of-Speech Tagging*, basta que elas sejam a classe gramatical da palavra. No entanto, para representarmos árvores de dependência, é essencial que tais classes sejam capazes de caracterizar uma estrutura de dependência.

Uma das formas de representarmos árvores de dependência é dar para cada palavra uma classe capaz de indicar o seu *head*. Sendo assim, a junção do *token* e de sua classe são suficientes para formarmos uma relação de dependência. Tendo escolhido a melhor forma de representar as classes, o resto é um simples problema de classificação de *tokens*.

Não é necessário muita informação para referenciarmos ao *head* de cada palavra. Frases como ”primeiro substantivo para a direita” ou ”segunda preposição para a esquerda” possuem todas as informações necessárias, dado que se saiba a classe gramatical de cada palavra. Essas sentenças podem ser codificadas em classes através da concatenação de três informações, sendo esses um contador de distância, a classe gramatical do *head* e uma direção. Para

os dois exemplos dados, teríamos então *1_substantivo_direita* e *2_preposição_esquerda*.

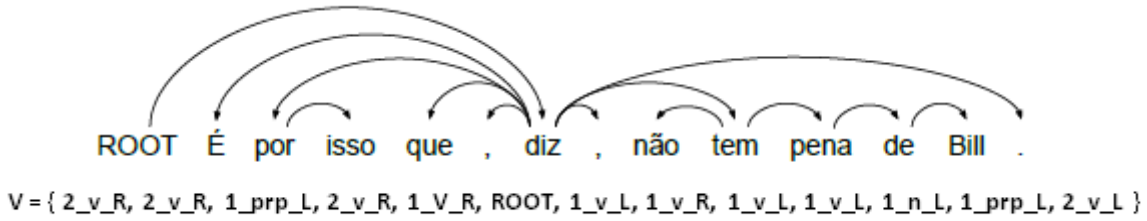

Figura 2.5: Exemplo de representação baseada em *tokens*

2.4 Conference on Natural Language Learning Shared Task

Diferentemente de modelos *grammar-driven* (27), que utilizam conhecimentos da gramática, modelos *data-driven* necessitam de *corpus* com dados anotados sobre relações de dependência para aprenderem a predizer grafos de dependência. (3) A base de dados utilizada neste trabalho é derivada de um dos muitos *corpora* disponibilizados pela *Conference on Natural Language Learning Shared Task* 2006.

A cada ano a CoNLL realiza uma forma de competição onde participantes treinam e testam sistemas utilizando o mesmo *dataset* em busca de melhores resultados. A tarefa do ano de 2006 teve como foco o desenvolvimento de *parsers* de dependência multilíngues visando avançar o estado-da-arte na criação de modelos capazes de representar dependências independente da linguagem. (9) Para os competidores realizarem seus testes foram disponibilizados 13 *corpora* para as 13 línguas abordadas no desafio. Neste trabalho utilizaremos o Bosque, que foi o *corpus* utilizado nesta competição.

```
<ext n=1 sec=clt sem=92b>
<t>
<s id="1" ref="CP1-1" text="Um revivalismo refrescante">
Um   [um] <arti> ART M S @PU #1->2
revivalismo  [revivalismo]  N M S @NPHR #2->0
refrescante  [refrescante]  ADJ M S @N< #3->2
</s>
</t>
<p>
```

Figura 2.6: Exemplo de uma entrada do Bosque.

Bosque é um dos quatro *datasets* do projeto *Floresta Sintáctica* e possui o foco em textos retirados de várias fontes e completamente revisado por

linguistas. Ele é composto por diversas frases, suas análises morfossintáticas e relações hierárquicas, possuindo mais de 186000 palavras organizadas em 9359 frases, sendo mais de 26000 palavras únicas. Desses dados podemos extrair informações importantes como a classe gramatical de cada palavra e a árvore de dependência de cada frase. A análise morfossintática de suas frases foi extraída automaticamente pelo parser Palavras, também utilizado em outros trabalhos de anotação morfossintática (4), e depois manualmente corrigidas por (14). Apesar de não ser tão grande quanto os outros *corpora*, o Bosque possui um foco maior em precisão, fazendo ele ser o *corpus* mais apropriado para utilização em pesquisas.

3
Modelagem

Nesta seção focamos na modelagem do problema assim como na seleção e extração das classes necessárias para a abordagem *Token-Based* utilizada neste trabalho. Este capítulo contém a analise do *corpus* com os dados em sua forma menos polida assim como as estratégias empregadas para transformar a tarefa de Análise de Dependência em um problema de classificação de *tokens*.

3.1
Feature Engineering

Feature Engineering, ou Engenharia de Atributos, é o nome dado para a tarefa que consiste na identificação ou criação de atributos ou classes - aqui chamados de "Etiquetas" ou *Tags* - em nossos dados com o intuito de gerar uma melhor representação do problema a ser resolvido. Este trabalho utiliza como base o *token-based approach* proposto por (9) e implementado com sucesso em outras pesquisas do laboratório LEARN(3) (25). Neste trabalho, o problema de Análise de Dependência de uma frase é tratado como um problema de classificação de *tokens* cujos atributos são informações referentes à classe gramatical de cada palavra e a orientação da sua relação de dependência. O objetivo desta seção é apresentar cada um dos atributos extraídos de relevância para a etapa de classificação. Para o resto desse texto, utilizaremos os termos "Etiqueta" ou "*Tag*" como referência às informações extraídas.

3.1.1
Classe Gramatical (POS)

Essa etiqueta é responsável por representar a classe gramatical, ou *Part-Of-Speech*, da palavra em questão. Palavras que possuem a mesma classe gramatical geralmente apresentam um comportamento similar em termos sintáticos e morfológicos, ou seja, elas tem papéis parecidos na estrutura gramatical de uma oração.

Além disso, essa abordagem ajuda a reduzir o domínio do conjunto de entrada do problema, dado que temos milhares de palavras contidas no vocabulário português mas apenas algumas dezenas de possíveis classes

Tabela 3.1: Exemplo da extração da *tag* PoS

ID	***Token***	**PoS**
1	É	adv
2	por	prp
3	isso	pron
4	que	adv
5	,	punc
6	diz	v
7	,	punc
8	não	adv
9	tem	v
10	pena	n
11	de	prp
12	Bill	prop
13	.	punc

gramaticais. Alguns exemplos de classes gramaticais na língua portuguesa são verbos, substantivos e adjetivos.

Tabela 3.2: Dados da *tag* PoS presentes no *corpus*

Nome	***Tag***	**Count**
Adjetivo	adj	10721
Advérbio	adv	9092
Artigo	art	29313
Conjunção	conj	7393
Prefixos	ec	5
Interjeição	in	36
Substantivo	n	40065
Numeral	num	4161
S. Preposicional	pp	397
Pronome	pron	10992
Nome Próprio	prop	11703
Preposição	prp	32390
Verbo	v	26302
S. Verbal	vp	1

3.1.2 Dados sobre o *Head*

Como este trabalho consiste em classificar relações pai-dependente em uma sentença, é necessário que nossa *tag* contenha informações sobre as relações de dependência de cada palavra. Inicialmente isso pode ser representado pelo índice absoluto do *head*, *hIndex*, e pela etiqueta POS do *head*, (*hPOS*). (Tabela 3.3)

Tabela 3.3: Etiquetas com informações do *head*.

ID	***Token***	**POS**	**hIndex**	**hPoS**
1	É	adv	9	v
2	por	prp	9	v
3	isso	pron	2	prp
4	que	adv	9	v
5	,	punc	6	v
6	diz	v	0	root
7	,	punc	6	v
8	não	adv	9	v
9	tem	v	6	v
10	pena	n	9	v
11	de	prp	10	n
12	Bill	prop	11	prp
13	.	punc	6	v

Porém, ao aplicarmos a representação proposta para as sentenças mais longas de nosso *corpus*, evidenciamos um problema. O *range* dos possíveis valores para o índice absoluto do *head* de uma palavra é muito grande, pertencendo ao intervalo de [1,N] sendo N o número de elementos da sequência. Essa variação torna difícil a tarefa de classificação, dado que não há nenhum padrão real contido nessa informação.

A forma proposta por (9) para contornar este problema é utilizar a subtração dos índices de palavras envolvidas em uma relação de dependência. Essa *distância* entre as palavras é uma representação mais estável pois é invariável quanto a localização das palavras na frase.

Esse *contador de distância* pode ainda, em teoria, assumir valores tão grandes quanto a maior frase de nosso *corpus*. Uma forma de contornar este problema é utilizar uma distância relativa ao invés de uma distância absoluta. Usamos então a quantidade de palavras entre um *token* dependente e seu *head* que compartilham a mesma etiqueta *PoS* do *head*. Em outras palavras, ao invés da classe de cada *token* dizer "arco 10 palavras para direita", ela diz "arco 3 verbos para a direita".

Pode-se notar que essa nova representação resulta em um número de distâncias relativas muito menor, assumindo, para o *corpus* em questão, valores apenas entre 1 e 25. Apesar de ainda ser uma quantidade aparentemente grande, valores muito altos - como por exemplo 5 e acima - se mostram extremamente incomuns. Isso se dá devido a estrutura da língua portuguesa, onde é muito raro que uma palavra seja dependente de alguma outra a, por exemplo, 5 ou mais verbos de distância.

Tabela 3.4: Distância relativa do *head*.

ID	***Token***	**POS**	**hIndex**	**hPoS**	**hDistance**
1	É	adv	9	v	2
2	por	prp	9	v	2
3	isso	pron	2	prp	1
4	que	adv	9	v	2
5	,	punc	6	v	1
6	diz	v	0	root	root
7	,	punc	6	v	1
8	não	adv	9	v	1
9	tem	v	6	v	1
10	pena	n	9	v	1
11	de	prp	10	n	1
12	Bill	prop	11	prp	1
13	.	punc	6	v	2

Tabela 3.5: Distribuição da Distância

Distância	***Count***
1	160435
2	8786
3	2372
4	935
>4	893
root	288

3.1.3 Lado relativo do *Head*

Além de representarmos a distância entre duas palavras que possuem uma relação, devemos também especificar qual a ordem relativa entre seus tokens. A terceira e última informação extraída é a orientação relativa do *token* pai em relação ao seu dependente. Ela consiste simplesmente em dizer, para cada palavra dependente, se o seu pai se localiza para esquerda ou para direita em relação à ordem de leitura da frase. Caso a palavra seja a raiz da árvore, dizemos que seu *hSide* é *root*.

Tabela 3.6: *Tags* de orientação

hSide	***Significado***
R	Direita
L	Esquerda

Assim, torna-se possível, para cada frase, representar sua estrutura de

dependência através de um *array* de *tags* compostas pela concatenação das etiquetas *hDist*, *hPoS* e *hSide*.

Tabela 3.7: Etiqueta relativa final.

Token	***hPoS***	**hDist**	**hSide**	**Tag**
É	v	2	R	2-v-R
por	v	2	R	2-v-R
isso	prp	1	L	1-prp-L
que	v	2	R	2-v-R
,	v	1	R	1-v-R
diz	root	root	root	root
,	v	1	L	1-v-L
não	v	1	R	1-v-R
tem	v	1	L	1-v-L
pena	v	1	L	1-v-L
de	n	1	L	1-n-L
Bill	prp	1	L	1-prp-L
.	v	2	L	2-v-L

Com isso temos todas as informações necessárias para representar a estrutura de árvore em uma sentença. Todas as etiquetas são concatenadas para formar a etiqueta utilizada em nossa classificação. O nosso problema de classificação é então reduzido de mais de 27000 classes, ou palavras únicas, para apenas 128 combinações diferentes dos três atributos extraídos.

3.2 Um filtro para distâncias

Na seção anterior entramos em detalhes da modelagem necessária para tornar a tarefa de *parsing* de Árvores de Dependência uma tarefa de classificação de *tokens*. A *tag* responsável por identificar um *head* é composta por três partes: um *contador de distância*, uma *classe gramatical* e uma *orientação*. O objetivo principal deste trabalho é criar um modelo especializado na sub-tarefa responsável por dizer o valor do *contador de distância*. Para alcançarmos esse objetivo, precisamos utilizar todas as informações possíveis.

Atualmente nosso problema é caracterizado por uma tarefa de classificação cujo dicionário de saída possui 25 classes, ou distâncias, e no dicionário de entrada constam 26 mil palavras diferentes. Cada uma dessas palavras tem uma classificação diferente dependendo das palavras que as cercam e também de seu uso na frase. Em busca de diminuirmos o dicionário de entrada, este trabalho propõe utilizarmos as demais informações disponíveis na base de dados para termos uma melhor representação da frase. Ao invés do nosso *input*

Tabela 3.8: Dicionário de dados para cada *tag*.

Tag	**Descrição**
adj	Adjetivo
adv	Advérbio
art	Artigo
conj	Conjunção
ec	Prefixos
in	Interjeição
n	Substantivo
num	Numeral
pp	S. Preposicional
pron	Pronome
prop	Nome Próprio
prp	Preposição
v	Verbo
vp	S. Verbal
R	Direita
L	Esquerda
[0,25]	Distância

ser baseado no vocabulário, utilizaremos uma *tag* composta pela *classe gramatical* (*PoS*), a *classe gramatical de seu head* (*hPoS*) e a orientação de seu *head* (*hSide*).

Tabela 3.9: *Tag* proposta

Token	***PoS***	***hPoS***	**hSide**	**Tag Proposta**	**hDist**
É	adv	v	R	adv-v-R	2
por	prp	v	R	prp-v-R	2
isso	pron	prp	L	pron-prp-L	1
que	adv	v	R	adv-v-R	2
,	punc	v	R	-	1
diz	v	root	root	root	root
,	punc	v	L	-	1
não	adv	v	R	adv-v-R	1
tem	v	v	L	v-v-L	1
pena	n	v	L	n-v-L	1
de	prp	n	L	prp-n-L	1
Bill	prop	prp	L	prop-prp-L	1
.	punc	v	L	-	2

4
Redes Neurais

Nesta seção focamos na metodologia aplicada para resolver o problema de classificação de *tokens* seguindo a formalização descrita anteriormente. A abordagem deste trabalho consiste em utilizar uma estrutura de diversas camadas e Redes Neurais Recorrentes (RNNs), mais em específico *Long Short-Term Networks*. Abaixo descrevemos e mostramos a estrutura da rede utilizada por este trabalho.

4.1
Embedding

Para a etapa de treino de nosso algoritmo, devemos alimentar uma rede neural com dados de nosso *dataset*. O *corpus* em questão é composto por palavras, ou símbolos, porém modelos de *Machine Learning* geralmente trabalham em cima de vetores de números inteiros. Para contornar este problema, devemos empregar uma maneira de representar o vocabulário - lista de palavras únicas do texto - de forma numérica.

Uma das possíveis conversões consiste em utilizarmos vetores *one-hot*. Nessa técnica, damos para cada palavra do vocabulário um valor $v \epsilon [0, VocabSize]$ e a representamos através de vetores de bits de comprimento *VocabSize*. Todos os *bits* de um vetor têm valor zero, com exceção do *bit* cujo índice corresponde ao valor dado para a *string* em questão.

Tabela 4.1: Exemplo de representação *one-hot*.

Palavra	***VSize***	**Vetor**
Menino	8	[0,0,0,0,0,0,0,1]
Homem	8	[0,0,0,0,0,0,1,0]
Príncipe	8	[0,0,0,0,0,1,0,0]
Rei	8	[0,0,0,0,1,0,0,0]
Menina	8	[0,0,0,1,0,0,0,0]
Mulher	8	[0,0,1,0,0,0,0,0]
Princesa	8	[0,1,0,0,0,0,0,0]
Rainha	8	[1,0,0,0,0,0,0,0]

Apesar de ser uma representação válida, a Tabela 4.1 evidencia o porquê dessa representação não ser desejável. Primeiramente, como o comprimento do vetor resultante deve ser igual ao tamanho do vocabulário da base de dados, a abordagem se torna pouco escalável pois pode resultar em vetores muito longos. Além disso, vetores *one-hot* não possuem informação alguma em relação ao significado, uso e contexto da palavra ou símbolo que representam. Para evitarmos estes problemas, utilizamos então *Word Embeddings*.

Word Embeddings têm como objetivo reduzir os longos vetores da representação *one-hot* em vetores menores e de forma a preservar o significado e contexto de cada palavra (37). Essa técnica consiste em criar um espaço vetorial de tamanho *EmbeddingSize*, onde cada palavra corresponde a um lugar dentro deste espaço. Isso resulta em uma matriz de tamanho $VocabSize * EmbeddingSize$, onde cada símbolo é um vetor de comprimento *VocabSize* contido dentro dessa matriz. Porém, simplesmente utilizar essa técnica ainda não garante que os vetores resultantes irão conter informações contextuais da palavra. Para isso utilizamos Word2Vec.

4.1.1 Word2Vec

Word2Vec são modelos que, através de uma etapa de treino, são capazes de realizar a tarefa de *Embedding* de forma a agrupar símbolos de significados semelhantes utilizando dados sobre sua proximidade em frases. Esse modelo recebe como *input* os vetores de *tags* extraídas da base de dados e com usos de técnicas como *Negative Sampling* e *skipgrams* nos retorna uma matriz de pesos como representação de cada símbolo. Com isso conseguimos representar nossos símbolos de uma forma compacta e sem perdermos informações preciosas sobre cada palavra.

Tabela 4.2: Exemplo da representação *Embedding* com valores *mock*.

Palavra	**Dimensão**	**Vetor**
Menino	3	[0,0,0]
Homem	3	[0,0,1]
Príncipe	3	[0,1,0]
Rei	3	[0,1,1]
Menina	3	[1,0,0]
Mulher	3	[1,0,1]
Princesa	3	[1,1,0]
Rainha	3	[1,1,1]

A Tabela 4.2 mostra um exemplo de representação das mesmas palavras utilizadas anteriormente mas agora com *word embeddings*. Neste exemplo pode-se notar que foram utilizados vetores menores para a representação dos mesmos símbolos, resultando em modelos que precisam de menos memória. Podemos observar também que a representação em um espaço vetorial de 3 dimensões consegue manter certas relações entre as palavras, dado que palavras semelhantes dividem o mesmo plano.

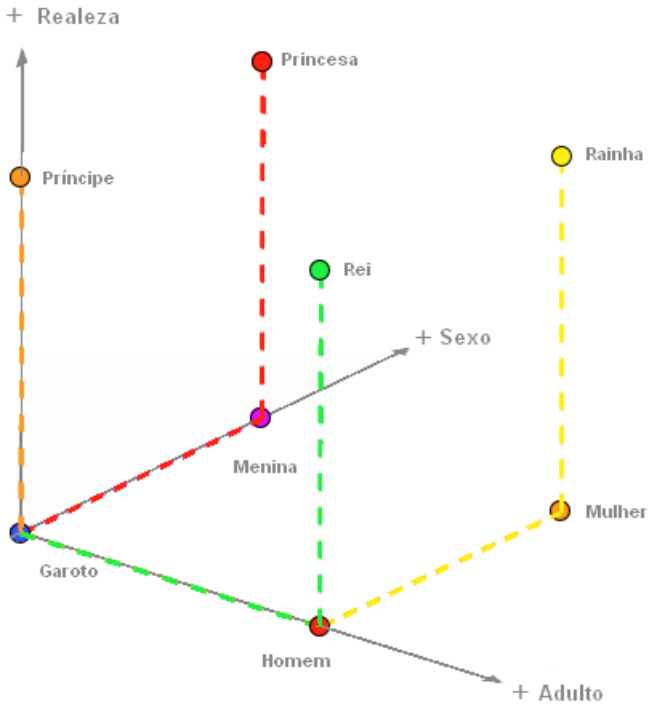

Figura 4.1: *Mock* da representação gráfica de *embeddings* com contexto.

4.2 Redes Neurais Recorrentes

Uma das limitações de Redes Neurais convencionais são suas restrições em termos de *output*. O uso de RNs geralmente consiste em receber como entrada um vetor de tamanho fixo - como por exemplo um *array* de características de um objeto ou até mesmo uma imagem - e produzir como saída um outro vetor, geralmente contendo a probabilidade de classes diferentes, também de tamanho fixo.(20)

RNNs, do inglês *Recurrent Neural Networks*, são modelos que nos permitem trabalhar utilizando sequências de vetores, tornando viáveis problemas de classificações de natureza *one-to-one*, como problemas tradicionais de classificação, *one-to-many*, como criação automática de legendas para imagens, *many-to-one*, como análise de sentimentos e *many-to-many*, como tradução de textos.

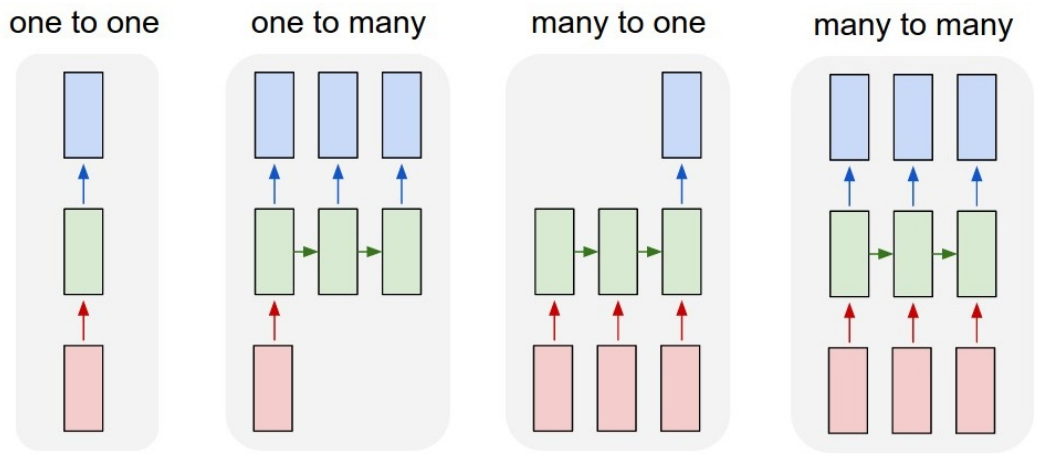

Figura 4.2: Diferentes configurações de RNNs (20)

Esses modelos consistem em redes contendo *loops* entre si, permitindo com que informações de *timesteps* anteriores persistam para a avaliação de *timesteps* posteriores (19), demonstrando o porquê de sua relação intrínseca com listas e sequências. Isso faz com que RNNs sejam o modelo de escolha para problemas de natureza sequencial, como *speech recognition*, *language modeling* e *neural machine translation*. Como a natureza deste trabalho consiste na predição de uma sequência de saída na forma de um vetor de distâncias pai-dependente dado uma sequência de entrada, RNNs são a escolha natural para a abordagem em questão.

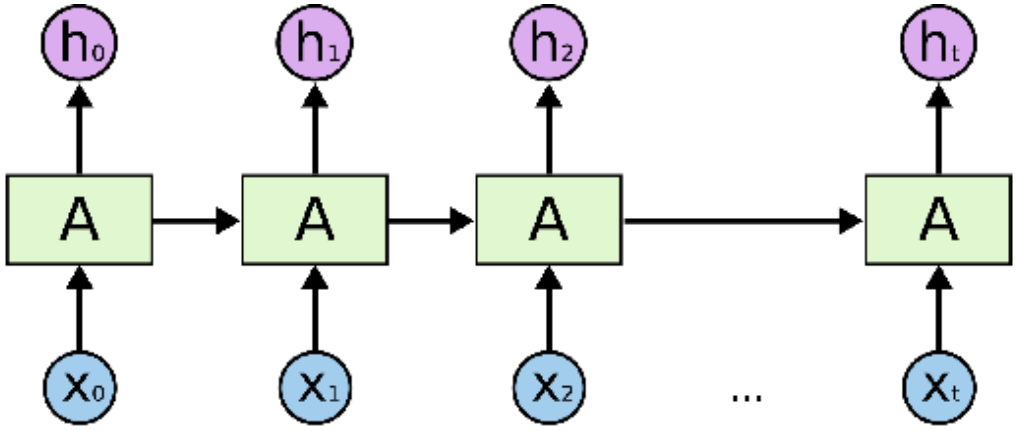

Figura 4.3: Redes Neurais Recorrentes (19)

Porém, embora seja possível que esses modelos consigam reter informações extraídas no começo da sequência para serem utilizados apenas no final, na prática isso não é tão simples. Conforme a distância aumenta, a capacidade de RNNs de reter informações e as conectar com sucesso é reduzida. Para resolver esse problema foram introduzidas (16) redes LSTM, descritas na seção seguinte.

4.2.1 LSTM

Redes RNN são uma forma de *loop* ou corrente de redes neurais, onde cada iteração utiliza o *output* da iteração anterior. O problema dessa estrutura é que ao aplicarmos *back-propagation*(36) convencional, o gradiente tende ou a sumir (*vanishing gradient*) ou explodir (exploding gradient) (16). Isso faz com que RNNs convencionais tenham problemas em retratar dependências de longo prazo em dados sequenciais.

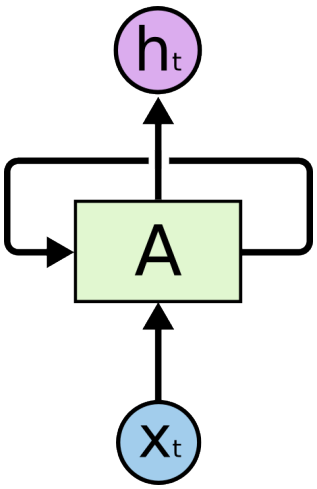

Figura 4.4: RNNs retratadas como *loops*.

Redes *Long Short Term Memory* (LSTM)(16) são uma forma de RNN capaz de resolver o problema de aprendizagem de *long-term dependencies*. De forma semelhante a RNNs convencionais, redes LSTM também são formadas por redes sequenciais. A diferença é que ao invés do módulo que se repete ser apenas uma camada, em redes LSTM esse módulo é composto por quatro camadas de redes neurais interagindo de uma forma específica. Cada uma dessas redes são chamadas de *gates* e possuem funcionalidades específicas. Além disso, cada uma delas possui uma matriz de pesos própria, o que permite que sejam atualizadas evitando os problemas de gradiente que temos em redes convencionais.

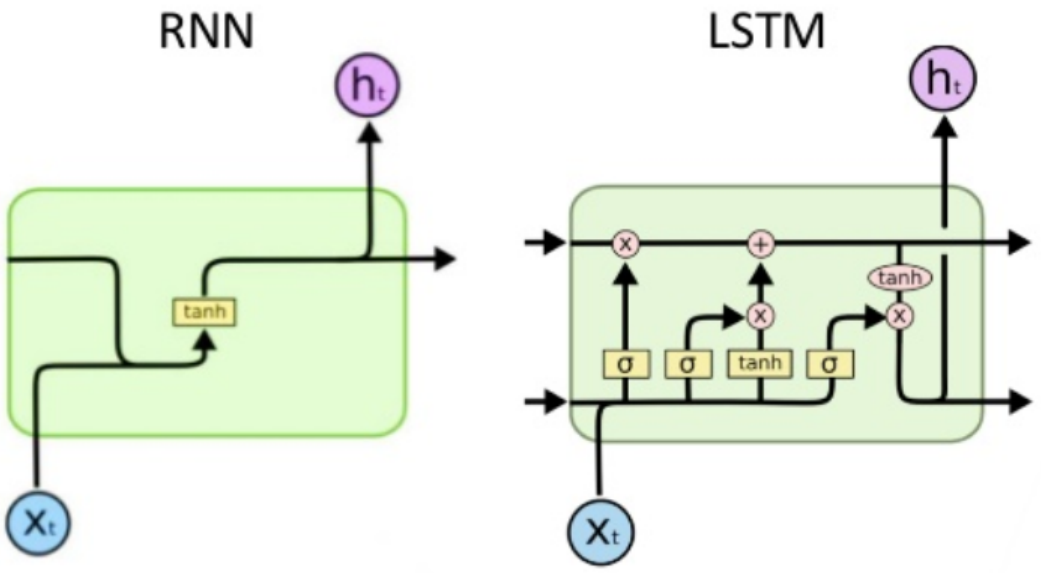

Figura 4.5: Diferença de uma cédula RNN para LSTM. (19)

O motivo de redes LSTM serem tão boas em memórias de longo prazo é o *cell state* (c), que realiza a função de *long-term memory*. A cada novo *timestep* de uma LSTM, essa memória pode ser alterada. Dela são removidas informações pouco importantes de passos anteriores e então são adicionadas informações do passo atual que a rede julgue importante se lembrar.

Cada um dos *gates* de uma LSTM é responsável por uma operação específica. A Figura 4.6 mostra em detalhes a estrutura de uma célula LSTM. Cada uma de suas camadas e funcionalidades é descrita abaixo.

1. **Forget** (f): Camada sigmoide responsável por olhar para o novo *input* e decidir quais informações de *timesteps* anteriores devem ser removidas do *cell state*. Em outras palavras, essa camada decide quais palavras a rede deve esquecer.

2. **Input** (i): Camada sigmoide que decide quais valores do *timestep* atual de importantes de se lembrar.

3. **Candidate** ($\tilde{c}$): Decide quais valores do *input* (X) são candidatos válidos para serem adicionados na memória de longo prazo.

4. **Output** (o): Determina qual parte da entrada (*timestep* atual) é passada para o *output* junto com o *cell-state* atualizado.

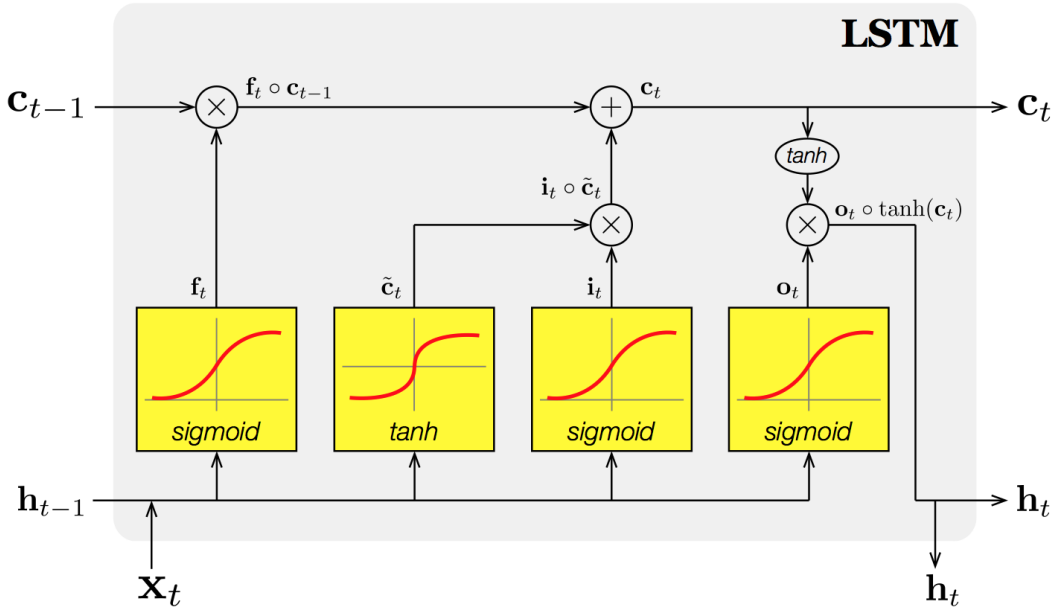

Figura 4.6: Visão detalhada de célula LSTM.

Cada um dos *gates* é uma rede neural tradicional com diferentes funções de ativação e uma matriz de peso própria. A fórmula de cada uma delas é listada abaixo, junto com as operações que utilizam a saída de cada *gate* representadas na imagem anterior.

Gating variables

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t\right)$$
$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

Candidate (memory) cell state

$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right)$$

Cell & Hidden state

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$$

Figura 4.7: Equações de uma célula LSTM.

4.2.2 Bidirectional LSTM

Redes Neurais Recorrentes e LSTMs são uma ótima forma de lidar com dados sequenciais onde existem correlações entre dados próximos. Esses

modelos focam em utilizar informações relevantes de dados passados para melhor classificar dados atuais. Para alguns casos isso é o suficiente.

No entanto em tarefas NLP, devido a forma que uma linguagem natural pode ser estruturada, informações relevantes para processarmos a significância de uma palavra podem se localizar muito a frente em uma sentença. Em tarefas de *Dependency Parsing* é especialmente fácil visualizar este problema, dado que existem diversas arestas de dependência com várias palavras de distância entre o *head* e seu dependente.

Figura 4.8: Frase com relações de dependência de longa distância.

Informações futuras da entrada podem ser úteis para a predição. RNNs tradicionais são capazes de tratar esses casos se atrasarmos sua saída por alguns *timesteps*. (29) Em teoria esse atraso, ou *delay*, poderia ser grande o suficiente para conter toda informação futura. Por exemplo: se temos o tamanho máximo de uma frase, sabemos o quão grande esse *delay* deve ser para nossa saída levar em conta toda a nossa entrada. Porém, na prática essa técnica nos dá resultados piores para atrasos muito altos. Isso ocorre pois, para *delays* grandes, o foco da modelagem de uma RNN acaba concentrado em se lembrar dos dados de entrada relevantes para a predição do próximo *timestep*, deixando menos foco no que foi aprendido em *inputs* passados. (31)

Redes LSTM Bidirecionais são uma forma de contornarmos esse problema. Essa técnica consiste em utilizar duas redes neurais separadas de forma a utilizar todas as informações do conjunto de entrada. Nesse método, cada rede LSTM é responsável por interpretar a informação em uma direção de tempo. Ou seja, teremos uma rede LSTM responsável pelo processamento de uma frase na ordem que ela é lida e uma segunda rede responsável por processar a mesma frase no sentido contrário. Após isso, a saída de cada uma das redes é combinada para obtermos o resultado

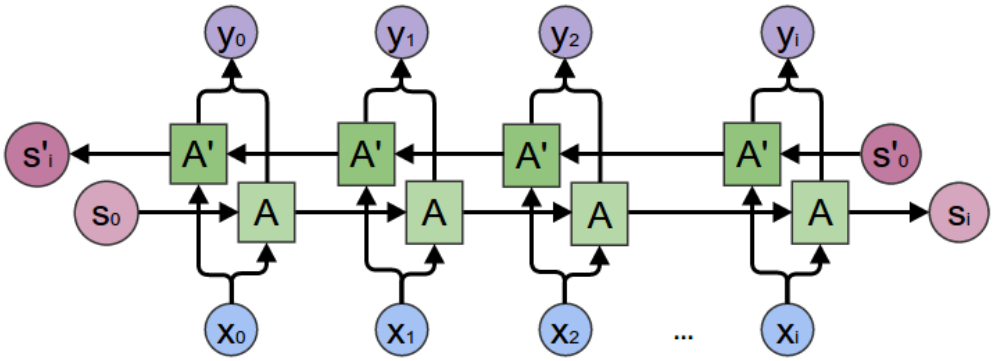

Figura 4.9: Estrutura de redes LSTM Bidirecionais.
[1]

4.3 Dropout

Dropout é uma camada responsável por aleatoriamente ignorar neurônios durante a fase de treino com uma dada probabilidade. Para cada amostra do conjunto de treino, e em cada iteração, uma fração p dos nós da rede é zerada. Isso aumenta a quantidade de iterações necessárias para haver convergência, mas reduz o tempo de treino de cada época e também o *overfitting*, pois força a rede neural a aprender características mais robustas ao invés de focar no que dá mais sucesso no conjunto de testes. (33)

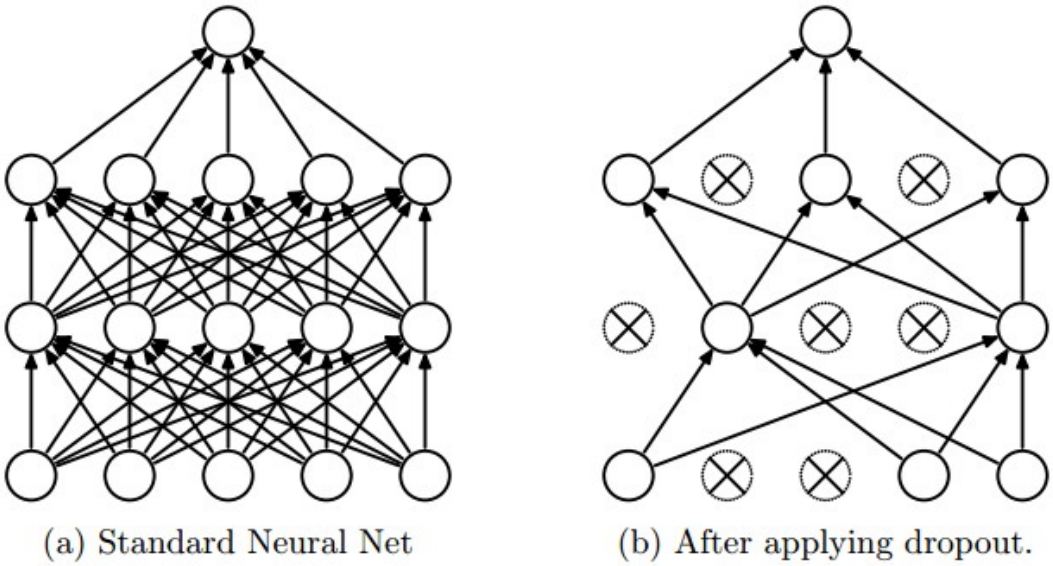

Figura 4.10: Exemplo de *dropout* (33).

[1] http://colah.github.io/posts/2015-09-NN-Types-FP/

4.4 Dense

Uma rede *dense* é uma simples camada *fully-connected* tradicional conectando cada nó da camada anterior em cada nó da camada atual. Em nosso caso, redes *dense* são utilizadas como a camada de *output* da rede. Seu uso, no entanto, é um pouco diferente em problemas de natureza *sequence-to-sequence*. Para entender melhor o problema, precisamos explicar o *shape* de algumas entradas e saídas em nosso modelo.

O *output* de uma rede *LSTM* sequencial possui o formato (*Bs*, *SeqLen*, *Hs*), onde *Bs* é o tamanho de um *batch*, *SeqLen* é o tamanho máximo das frases e *Hs* é a dimensão do *hidden state* da LSTM. Ou seja, cada sentença está sendo representada em uma matriz de dimensões $SeqLen \times Hs$. Porém, queremos que a saída final de nossa rede seja uma matriz de tamanho $SeqLen \times outputSize$ onde *outputSize* é o tamanho de nosso vocabulário de saída, ou seja, a quantidade de classes ou distâncias diferentes presentes no *corpus*.

Em outras palavras, a saída de uma camada LSTM representa cada *token* de uma sequência de entrada como um vetor de tamanho *hiddenSize*. No entanto, desejamos classificar nossos *tokens* como um vetor representando a probabilidade de cada distância. O papel desta camada *Dense* em nossa aplicação é realizar a conversão de vetores de grandes dimensões para a dimensão desejada.

Em outros tipos de redes, bastaria utilizar uma camada *fully-connected* ligando cada nó da rede anterior com *outputSize* nós na saída. Em redes LSTM no entanto, nossa saída é uma matriz onde cada coluna é responsável por representar uma palavra e, portanto, deve ser conectado à uma camada *fully-connected*. Sendo assim teremos *SeqLen* cópias de uma camada *Dense*, cada uma responsável pela conversão de um *timestep* da rede LSTM.

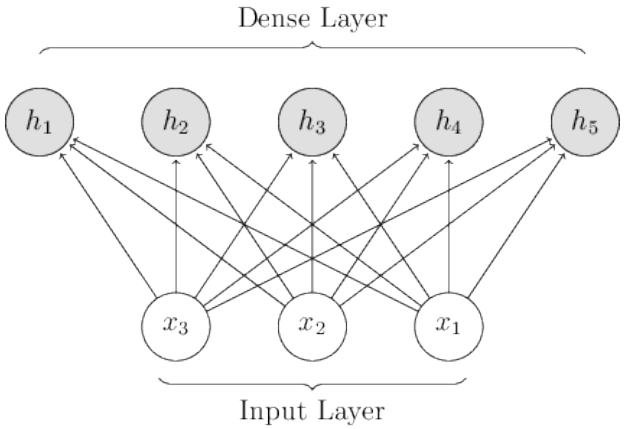

Figura 4.11: Visualização de uma camada *fully-connected*

O *output* da camada de *Dense* é então ativado por uma função *softmax* para obtermos a probabilidade de cada classe. A predição final da distância de cada *token* é o índice do elemento de seu vetor de saída com a maior probabilidade.

4.5 Modelo Completo

Com isso, todas as camadas da rede foram definidas. Os vetores de saída de cada camada da rede possuem o seguinte *shape*:

1. *Embedding*: (*batchSize*, *inputLen*, *embedSize*)
2. *Bi-LSTM*: (*batchSize*, *inputLen*, $2 \times lstmSize$)
3. *Dropout*: (*batchSize*, *inputLen*, $2 \times lstmSize$)
4. *Dense*: (*batchSize*, *inputLen*, *outputSize*)

Onde:

1. *inputLen*: Tamanho máximo das sentenças para o treino. Após a definição desse valor, sequências de tamanho superior são truncadas e sequências de tamanho inferior são preenchidas com 0.
2. *batchSize*: Quantidade de sentenças processadas a cada *batch*.
3. *embedSize*: Dimensão da camada de Embedding
4. *lstmSize*: Dimensão do *hidden-state* e do *output* da camada LSTM. Para redes bidirecionais, esse valor é dobrado dado que seu *merge* é feito através de concatenação.
5. *outputSize*: Quantidade de possíveis classes em nosso problema. Em nosso caso essa variável é a quantidade dos diferentes valores de distância que a rede deve ser capaz de classificar.

Uma representação visual do modelo completo pode ser encontrada abaixo.

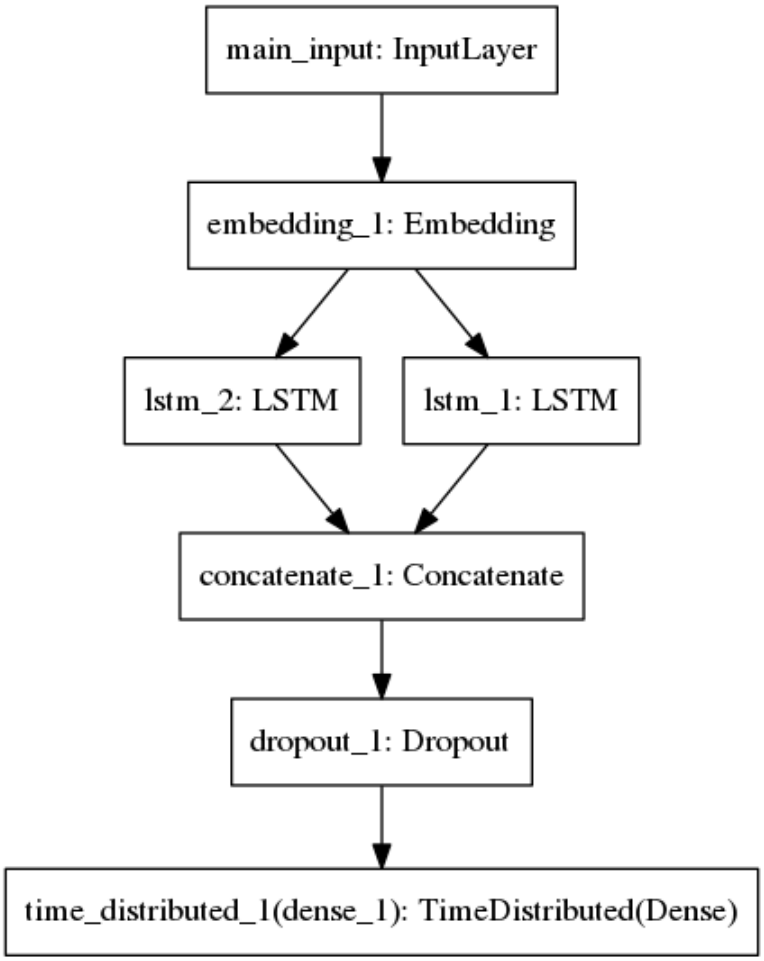

Figura 4.12: Visualização da rede completa.

5
Avaliação

Nesta seção são avaliados dados do *corpus* utilizado para obter nossos resultados, assim como os resultados experimentais e a comparação deles com os demais trabalhos nessa área desenvolvidos com a mesma abordagem. Primeiramente o corpus é descrito e analisado. Após isso discutimos a métrica utilizada para avaliação. Posteriormente, um modelo *baseline* é proposto e então o resultado deste trabalho é comparado com o mesmo e com os demais trabalhos nessa área.

5.1
Corpus

O projecto Floresta Sintá(c)tica é uma colaboração entre a Linguateca(22) e o projeto VISL(34). Contém textos em português brasileiro e europeu analisados automaticamente pelo analisador sintáctico PALAVRAS(5) e revistos por linguistas. Atualmente o projeto se divide em quatro partes, diferindo-se quanto ao modo (escrito ou falado) e quanto ao grau de revisão linguística. Para nós apenas é de interesse o *dataset* Bosque.

O Bosque é o corpus mais correto do Projeto Floresta Sintá©tica, sendo totalmente revisto por linguistas. Ele é composto por diversas frases, suas análises morfossintáticas e relações hierárquicas, possuindo mais de 186000 palavras organizadas em 9359 frases, sendo mais de 26000 palavras únicas.

Cada entrada do *corpus* é composta de uma frase em linguagem natural, no caso o Português, e dados referentes à sua análise, nem todos de interesse para este trabalho. As frases possuem tamanhos variados e passarão por etapas de parsing para extração de certos atributos descritos mais a frente.

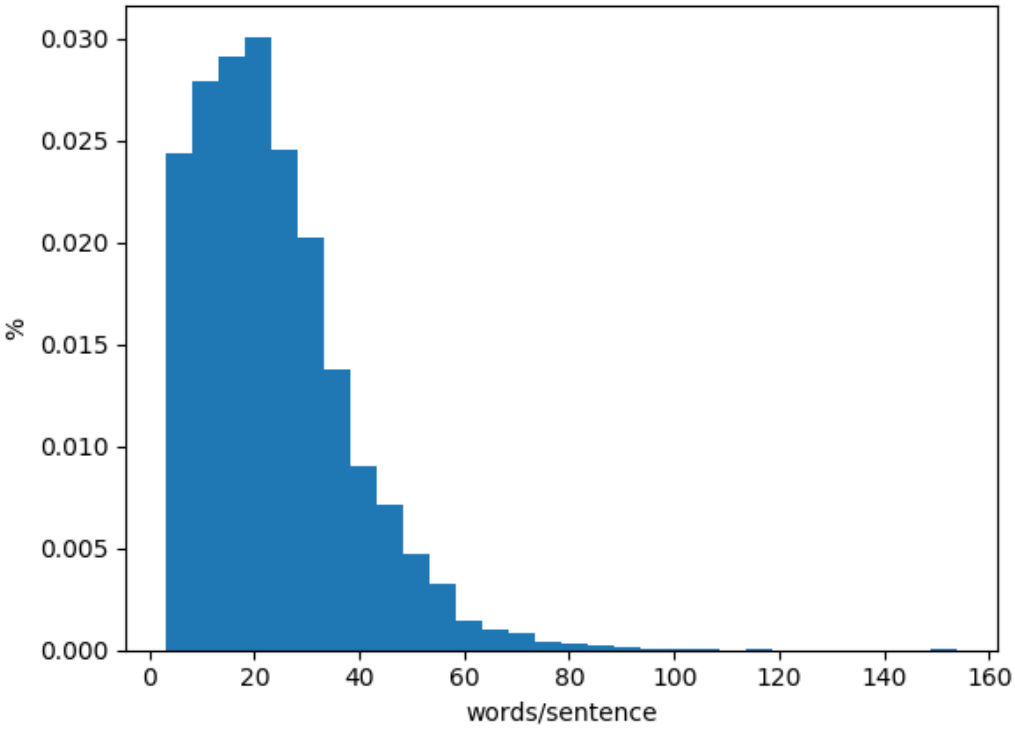

Figura 5.1: Histograma de palavras por frase no *corpus*.

Tabela 5.1: Dados sobre o *corpus*.

Atributos	**Quantidade**
Tokens	212.545
Sentenças	9.359
Tokens por sentença	22.8
Tamanho máximo de sentença	150
Tamanho médio de sentença	19.5
Etiquetas PoS	15
Etiquetas hPoS	15
Etiquetas hSide	2
Etiquetas hDistance	25

Para a CoNLL *Shared Tasks* 2006, o *corpus* foi particionado entre conjuntos de treino e teste. Das 9359 entradas do *corpus*, 288 frases foram selecionadas para compor a base de teste que é utilizada para medirmos a eficiência do modelo e o demais é utilizado para treino do mesmo. Como este trabalho segue a linha do proposto pela CoNLL, assim como pelos trabalhos de (3) (9) (25), seguiremos a mesma divisão.

Tabela 5.2: Splits de Treino e Teste

Dataset	**Sentenças**	**Tokens**
Treino	9.071	206.678
Teste	288	5.867

5.2 Métrica

Existem diversas métricas que podem ser empregadas para avaliar uma Árvore de Dependência. A CoNLL 2006 destacou três delas:

- *label accuracy* (LA) corresponde ao percentual de palavras com relações corretas;
- *unlabelled attachment score*(UAS) representa o percentual de palavras que o sistema conseguiu corretamente indicar seu *head*;
- *labeled attachment score* indica o percentual de *tokens* que o sistema foi capaz de corretamente classificar tanto o seu *head* quanto o tipo de relação dos dois.

Para avaliar os resultados deste trabalho é utilizada a métrica *Unlabelled Attachment Score* (UAS). Esse método de avaliação foi escolhido por se alinhar melhor ao propósito deste trabalho – a predição correta do *head* de cada *token* ao aprender sobre árvores de dependência. É importante notar que assim como na CoNLL, *tokens* que indicam pontuação não são utilizados na avaliação.

5.3 Avaliando o modelo

Este trabalho tem como objetivo fazer a predição da distância entre um *token* dependente e o seu *head*. Para avaliarmos a eficiência deste trabalho, o modelo proposto é aplicado no *corpus* da língua portuguesa utilizado na CoNLL 2006 *Shared Tasks*, o Bosque, e é avaliado utilizando a mesma métrica UAS utilizada na competição.

Os resultados são então comparados com o classificador *baseline* proposto e com os avanços feitos na mesma sub-tarefa por (3), (9) e (25).

5.3.1 Classificador *Baseline*

Para nosso *Baseline* vamos assumir que a distância entre o pai de cada *token* e seu dependente é de sempre 1. Esse valor foi escolhido por ser o valor mais frequentemente encontrado em nossa base de dados. A tabela abaixo mostra o valor de obtido com a métrica UAS para tal classificador.

Tabela 5.3: *Unlabeled attachment score* para o classificador baseline.

Modelo	**UAS (%)**
Baseline	87.522

5.3.2 Curta-Distância

Nossa forma de representar o *contador de distância* nos trouxe a grande vantagem de regularizarmos as distâncias quando comparado a utilizar os índices das palavras. No entanto, como visto antes, ao analisarmos a frequência de cada distância podemos ver que há uma ocorrência muito grande de distâncias menores e muito poucas ocorrências de grandes distâncias *head-dependent*. Isso se dá pois é estatisticamente mais provável que uma palavra esteja se referindo a outras palavras nos seus arredores do que em outras no final da frase.

Tabela 5.4: Distribuição da Distância

Distância	***Count***
1	160435
2	8786
3	2372
4	935
>4	893
root	288

Esse desbalanceamento da base de dados funciona contra nós de algumas maneiras. Além de nosso modelo se tornar mais pesado, ele também tenta ser capaz de assimilar casos de baixa utilidade e frequência. Para contornar este problema, resolvemos então treinar a rede apenas para valores de distância abaixo de 4. Isso sacrifica a acurácia para relações de grande distância, mas em compensação torna a rede mais especializada nos valores mais frequentes.

5.3.3 Resultados

Para o treino deste modelo baseado em LSTMs, a base de dados de treino teve de ser dividida mais uma vez. O objetivo dessa nova divisão é criarmos um modelo de validação, com o objetivo de constantemente testarmos o efeito dos ajustes dos hiper-parâmetros sem precisarmos olhar para a base de testes. Para esse *split*, novamente seguimos o que foi proposto pelos outros trabalhos estudados, selecionando 10% da base aleatoriamente para servir como nosso conjunto de avaliação.

Tabela 5.5: *Splits* de Treino, Validação e Teste

Split	**Sentenças**	***Tokens***
Treino	8.164	185.974
Validação	907	20.704
Teste	288	5.867

Outro recurso utilizado no treino foi o truncamento e *padding* das sequências. Durante a etapa de ajustes de parâmetros, um tamanho máximo *sequenceLenth* de valor positivo foi estabelecido. Todas as sequências que possuíam uma quantidade menor de palavras do que este número foram preenchidas com um *token* arbitrário para representar nosso *padding*, e todas as sequências com mais palavras do que este valor foram truncadas. Este método foi empregado para aumentar a performance através da redução do tempo de treino, tendo em vista que mais de 99% das frases do *corpus* possuem menos de 63 palavras, apesar de encontrarmos *outliers* de até 150 palavras de comprimento. Este tratamento é feito apenas para o treino e não possui qualquer relevância para a etapa de teste.

Os experimentos foram rodados em duas máquinas. A primeira possui um processador Intel i7-3960X CPU @ 3.30 GHz, 16GB de RAM e GeForce GTX TITAN. A segunda dispõe de dois processadores Intel Xeon Platinum 8160 CPU @ 2.10GHz e 192 GB de RAM.

É de interesse deste trabalho avaliar duas novas formas de tratarmos nosso problema de classificação. A primeira é a abordagem por Redes Neurais Recorrentes junto com a etiqueta proposta composta por *PoS-hPoS-hSide* para a predição da distância entre um pai e seu *token* dependente em nossa representação. A segunda é uma modificação deste modelo com um foco maior para valores baixos de nosso *contador de distância*. Como queremos testar o desempenho de nossa *tag*, também é listado o desempenho do mesmo modelo

quando utilizamos como entrada apenas a classe gramatical (*PoS*) das palavras de uma frase.

Tabela 5.6: Predição da distância entre *tokens* pai-dependente.

Parser	**Ano**	**Algoritmo de Aprendizagem**	**UAS**
Milanes (3)	2016	S-IFIS	94.67
Crestana (9)	2010	ETL	93.77
Motta (25)	2014	IFIS	92.01
Este Trabalho	2018	LSTM (PoS)	91.59
Este Trabalho	2018	LSTM (*Tag* Proposta)	94.82
Este Trabalho	2018	LSTM (*Tag* Proposta); hDist<4	**95.27**

O modelo baseado em RNNs em conjunto com a etiqueta *PoS-hPoS-hSide* obteve uma eficiência melhor na tarefa de classificação da distância entre *tokens* dependentes e seus pais do que os métodos abordados nas tarefas anteriores, com 95.25% de precisão na métrica utilizada. Fora isso, este método também possui uma performance melhor, visto que o melhor resultado obtido até então por (3) foi relatado com um tempo de execução superior a seis horas, enquanto que o modelo proposto por este trabalho foi treinado com o tempo médio de uma hora e dez minutos.

Os resultados relatados foram obtidos com 128 e 256 dimensões nas camadas de *embedding* e Bi-LSTM respectivamente. Como a camada Bi-LSTM é composta por duas LSTMs, temos o total de 512 dimensões. A dimensão da camada das redes *Dense* foi a quantidade de classes presentes em cada experimento, ou seja, 25 para o modelo que engloba todas as distâncias e 5 para o modelo de curta distância. Utilizamos também um *dropout* de valor 0.8 e um *recurrent dropout* - responsável por atuar entre as unidades de uma rede recorrente - de 0.5. Esses valores foram escolhidos por terem sido os experimentos responsáveis pelo melhor resultado em nosso *split* de validação. Como condição de parada nós utilizamos uma *callback* responsável por suspender o treino depois de 20 gerações sem progresso na acurácia do conjunto de validação.

5.3.4 Análise dos Erros

É possível observar que o aumento do vocabulário de entrada foi responsável por um ganho de precisão na hora de medir a distância pai-dependente, mostrando que a *tag* proposta neste trabalho é uma abordagem válida. No entanto, isso não retira a maior fonte de erro na hora da predição. Valores

pequenos de distância, como 1 e 2, correspondem a grande parte da base de dados. Isso faz com que a rede seja muito eficiente em reconhecer os padrões que predizem corretamente esses valores e muito fraca na predição de distâncias maiores do que 3. Isso é facilmente evidenciado ao analisarmos a matriz de confusão resultante do modelo que classifica todas as distâncias.

Todas as distâncias											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**root**	**support**
1	**4336**	40	6	2	0	0	0	0	0	0	4384
2	145	**89**	1	0	0	0	0	0	0	0	235
3	26	10	**25**	3	0	0	0	0	0	0	64
4	8	1	0	**10**	0	0	1	0	0	0	20
5	4	1	1	1	**0**	1	0	0	0	0	8
6	0	0	0	2	0	**0**	0	0	0	0	2
7	0	0	0	0	0	0	**0**	0	0	0	0
8	0	2	1	0	1	0	0	**0**	0	0	4
9	1	1	0	0	1	0	0	0	**0**	0	4
root	0	0	0	0	0	0	0	0	0	**288**	288

Figura 5.2: Matriz de Confusão para o modelo com todas as distâncias.

Podemos notar que apesar da rede ter sido treinada para classificar distâncias de tamanho até 25, ela não foi capaz de acertar nenhum valor acima de 4. Também é importante destacar que apesar da distância máxima no Bosque ser 25, o conjunto de testes apenas possui distâncias somente até 9 e com exceção do 7. Isso prova que estamos desperdiçando memória, tempo de treino e desempenho nesse modelo.

Outro ponto negativo da forma em que o *corpus* foi selecionado é que há um grande desbalanceio das classes, mesmo entre as distâncias mais frequentes. Embora tenhamos uma alta precisão para predição de distâncias 1, a precisão para distâncias 2 é de apenas 61%. Apesar de valores similares ou piores a esse serem encontrados em outras classes, a ocorrência de grandes distâncias é bem menor.

Curta Distância											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**root**	**support**
1	**4338**	39	5	2	0	0	0	0	0	0	4384
2	131	**102**	1	1	0	0	0	0	0	0	235
3	20	11	**32**	1	0	0	0	0	0	0	64
4	7	0	1	**12**	0	0	0	0	0	0	20
5	4	1	0	3	**0**	0	0	0	0	0	8
6	1	0	0	1	0	**0**	0	0	0	0	2
7	0	0	0	0	0	0	**0**	0	0	0	0
8	2	0	0	2	0	0	0	**0**	0	0	4
9	2	0	0	2	0	0	0	0	**0**	0	4
root	0	0	0	0	0	0	0	0	0	**288**	288

Figura 5.3: Matriz de Confusão para dependências de curta distância.

Na figura acima podemos ver o impacto do foco em arcos menores. Ao sacrificarmos completamente relações de distância 5 ou acima, conseguimos um resultado melhor no geral. Essa rede tem um desempenho pior para sentenças com arcos de dependência longos, mas esses casos são estatisticamente raros em nossa representação e, portanto, possuem importância negligenciável. Esse ideal também se encaixa bem com a dificuldade que RNNs possuem em retratar *long-term dependencies*.

	Todas as Distâncias		*Curta Distância*	
	Precision	**Recall**	**Precision**	**Recall**
1	0,9592	0,989	0,9629	0,9895
2	0,618	0,3787	0,6667	0,434
3	0,7352	0,3906	0,8205	0,5
4	0,5555	0,5	0,5	0,6
5	0	0	-	0
6	0	0	-	0
7	0	-	-	-
8	-	0	-	0
9	0	0	-	0
root	1	1	1	1

Figura 5.4: Precisão e *recall*.

Apesar dessas insuficiências o modelo proposto possui uma precisão muito boa para os casos de relações pai-dependente com distância 1 e raiz, obtendo precisões satisfatórios de 96% e 100%. Além de aperfeiçoar a métrica UAS, focar a rede em valores pequenos de distância também trouxe melhorias na precisão e no *recall* de todas as classes.

6
Conclusões

Neste trabalho construímos um filtro de arestas para a tarefa de *dependency parsing* utilizando uma abordagem baseada em tokens através de redes *Long Short Term Memory* bidirecionais.

Foi explicado o que é *Dependency Parsing*, suas ligações *head-dependent* e como o conjunto de relações de dependência de uma frase forma um grafo acíclico e enraizado em forma de árvore. Nós destacamos que a tarefa de *parsing* neste trabalho é focada em dados e não em regras gramaticais do português. Após isso mostramos a diferença entre *parsers* baseados em transição e em grafos, duas das abordagens mais comuns. Então entramos em maiores detalhes em modelos *token based*, a abordagem utilizada para este trabalho.

Seguindo a linha de pesquisa do Laboratório de Engenharia de Algoritmos e Redes Neurais (LEARN), fomos a fundo na abordagem de análise de dependência através de classificação de *tokens*. Vimos como o *approach* proposto por (9) e (25) possibilita a divisão da tarefa em sub-etapas menos complexas relacionadas três informações do *head*. Em suas obras, eles constroem classificadores *token-based* separados para cada uma das informações do *head*, sendo elas: *part-of-speech*, sua orientação em relação ao seu *token* dependente e a quantidade de palavras entre o *head* e seu dependente com a mesma *PoS* do *head*.

Demonstramos em seguida os pormenores da abordagem por *tokens*. Listamos como cada *tag* pode ser extraída da base de dados e evidenciamos a sua vantagem quando comparado a métodos que utilizam somente a distância para caracterizar relações de dependência. Vimos que se utilizarmos as informações do *PoS* e da orientação do *head* na etapa de predição, conseguimos reduzir o contradomínio do nosso problema de até 125 possíveis distâncias *head-dependent* para apenas 25, com uma ocorrência baixíssima de valores acima de 4.

No domínio deste trabalho, a tarefa de *dependency parsing* é tratada como um problema de aprendizagem supervisionada utilizando dados estruturados. O foco desta tarefa é encontrar, para cada *token* de uma sentença, a *tag* capaz de identificar o seu *head*. Através de uma sequência de entrada composta pelo *PoS* da palavra junto com *PoS* e a orientação de seu *head*, nosso modelo

prediz uma das 25 possíveis distâncias que a palavra com a qual ela possui uma relação de dependência pode estar. Esse filtro de arcos pode ser utilizado em conjunto com outros preditores parciais desenvolvidos por diferentes linhas de pesquisa (3, 9, 25) de forma a melhorar um preditor final.

Em seguida listamos o modelo ML utilizado na construção de nosso filtro. Devido a natureza sequencial dos dados, nosso modelo de escolha são as Redes Neurais Recorrentes, que por natureza são capazes lidar com classificações *many-to-many*. Como estamos tratando de sequências grandes, RNNs tradicionais não seriam indicadas devido ao seu trabalho em modelarem dependências de longo prazo.

Explicamos então as *Long Short Term Networks*, uma forma especial de construir RNNs que, através de diversas camadas, consegue modelar melhor a memória de longo prazo necessária para nossa tarefa. Além disso explanamos como em NLP é necessário saber informações sobre a frase inteira e não apenas sobre as palavras lidas até o momento, nos levando então para LSTM bidirecionais por serem capazes de interpretar sentenças nos dois sentidos simultaneamente.

Para o treino utilizamos o *corpus* Bosque, disponibilizado pelo projeto Floresta Sintática. Esse dataset é composto por 9359 frases da língua portuguesa, contendo 18600 palavras e mais de 26000 palavras únicas. Além das frases em linguagem natural, o *dataset* também possui informações sobre as classes gramaticais e relações de dependência de cada palavra de uma sentença. Foi construído então um *parser* secundário para extração das *features* necessárias para etapa de treino, como o *PoS*, o *PoS* do *head* e a orientação.

Tendo definido a tarefa, a base de dados, a abordagem tomada e o modelo proposto, listamos nossa avaliação empírica. Nela observamos que a rede proposta foi capaz de obter uma acurácia de 94.92%, resultado que é maior que o estado-da-arte na tarefa de predição de distância obtido por (3) e em tempos até 6 vezes menores.

6.1 Trabalhos Futuros

Neste trabalho nós construímos um filtro de arestas rápido e de bom desempenho para a classificação da distância entre *tokens* que possuem uma relação de dependência utilizando redes LSTM. Isso mais uma vez prova que o poder de previsão *sequence-to-sequence* de RNNs se traduz bem para tarefas NLP. Sendo assim, é intuitivo propor como trabalhos futuros a aplicação de modelos Bi-LSTM para as demais sub-tarefas conhecidas. A construção de preditores LSTM para *head PoS-tags* ou *side* parece promissora e pode trazer

melhorias de acurácia e performance da tarefa completa.

Outro trabalho futuro é juntar os resultados obtidos através da metodologia proposta neste trabalho com os preditores estado-da-arte para as outras sub-tarefas. Acreditamos que o modelo criado neste trabalho seja capaz de auxiliar na acurácia do preditor incremental proposto por (3) e (25), uma vez que eles utilizam resultados inferiores aos obtidos aqui.

Referências bibliográficas

[1] AFONSO, S. ET AL.. **Floresta sintá(c)tica: A treebank for portuguese.** LREC, 2002.

[2] ALPHONSE, E.; AUBIN, S.; BISSON, G.; HAMON, T.; LAGARRIGUE, R.; NAZARENKO, A.; PIERRE MANINE, A.; NÉDELLEC, C.; OULD, M.; VETAH, A.; POIBEAU, T. ; WEISSENBACHER, D.. **Event-based information extraction for the biomedical domain: The caderige project**. In: ROCEEDINGS OF THE INTERNATIONAL JOINT WORKSHOP ON NATURAL LANGUAGE PROCESSING IN BIOMEDICINE AND ITS APPLICATIONS 2004, 2004.

[3] BARROSO, Y. M.. **Structured learning with incremental feature induction and selection for portuguese dependency parsing**. Dissertação de mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2016.

[4] BICK, E.; MELO, H.; PANUNZI, A.; RASO, T.. **The annotation of the c-oral-brasil spoken corpus using an adaptation of the palavras parser**. 06 2012.

[5] BICK, E.. **The Parsing System "Palavras": Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework**. Tese de doutorado, Aarhus University, Denmark, 2000.

[6] BUCHHOLZ, S.; MARSI, E.. **Proceedings of the tenth conference on computational natural language learning**. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, p. 149–164, 2006.

[7] CIRAVEGNA, F.; LAVELLI, A.. **Full text parsing using cascades of rules: an information extraction perspective**. In: PROC. OF THE 9TH CONF. OF THE EUROPEAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 1999, 1999.

[8] CONFERENCE ON NATURAL LANGUAGE LEARNING. **Conll 2018**. The SIGNLL Conference on Computational Natural Language Learning, 2018. Acesso em: Junho de 2018.

[9] CRESTANA, C. E. M.. **A token classification approach to dependency parsing**. Dissertação de mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2010.

[10] CUI, H.; SUN, R.; LI, K.; YEN KAN, M. ; SENG CHUA, T... **Information extraction for question answering: Improving recall through syntactic patterns**. SIGIR 2005, p. 400–407, 2005.

[11] DING, Y.; PALMER, M... **Machine translation using probabilistic synchronous dependency insertion grammars**. 2005.

[12] FERNANDES, ERALDO; DOS SANTOS, CICERO; MILIDIÚ, RUY. **A machine learning approach to portuguese clause identification**. 6001:55–64, 04 2010.

[13] FREITAS, C.; ROCHA, P.; BICK, E.. **Computational processing of the portuguese language**. Lecture Notes in Computer Science, 5190:216–219, 2008.

[14] FREITAS, C.; ROCHA, P.; BICK, E.. **Floresta sintá(c)tica: Bigger, thicker and easier**. p. 216–219, 09 2008.

[15] HACIOGLU, K... **Full text parsing using cascades of rules: an information extraction perspective**. In: COLING '04: PROCEEDINGS OF THE 20TH INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS,, p. 1273, Association for Computational Linguistics, Moristown, NJ, USA., 2004.

[16] HOCHREITER, S.; SCHMIDHUBER, J.. **Long short-term memory**. Neural Comput., 9(8):1735–1780, Nov. 1997.

[17] JIJKOUN, V.; RIJKE, M. D. ; MUR, J.. **Information extraction for question answering: Improving recall through syntactic patterns**. PROCEEDINGS OF THE 13 TH TREC, 2005.

[18] JING, YINGQI; LIU, HAITAO. **Mean hierarchical distance: Augmenting mean dependency distance**. 08 2015.

[19] KARPATHY, A.. **Understanding lstm networks**. Understanding LSTM Networks, 2015. Acesso em: Junho de 2018.

[20] KARPATHY, A.. **The unreasonable effectiveness of recurrent neural networks**. The Unreasonable Effectiveness of Recurrent Neural Networks, 2015. Acesso em: Junho de 2018.

[21] KOO, TERRY; MATTHEW RUSH, ALEXANDER; COLLINS, MICHAEL; S. JAAKKOLA, TOMMI; ALEXANDER SONTAG, DAVID. **Dual decomposition for parsing with non-projective head automata.** p. 1288–1298, 09 2010.

[22] LINGUATECA. **Linguateca**. Linguateca: um centro de recursos distribuído para o processamento computacional da língua portuguesa, 2015. Acesso em: Junho de 2018.

[23] MARTINS, A.; ALMEIDA, M. S. N.. **Turning on the turbo: Fast third-order non-projective turbo parsers**. 2:617–622, 08 2013.

[24] MEL'CUK,I. A.. **Dependency syntax: theory and practice.** SUNY press, 2.1, 2.2, 1988.

[25] MOTTA, E. N.. **Indução e seleção incrementais de atributos no aprendizado supervisionado**. Dissertação de mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2014.

[26] OUMA, G.; MUR, J.; NOORD, G. V.; PLAS, L. V. D.; TIEDEMANN, J.; GRONINGEN, R.. **Question answering for dutch using dependency relations.** WORKING NOTES FOR THE CLEF 2005 WORKSHOP, 2005.

[27] PLANK, B; VAN NOORD, G.. **Grammar-driven versus data-driven: Which parsing system is more affected by domain shifts?** 07 2010.

[28] QUIRK, C.; ET AL.. **The impact of parse quality on syntactically-informed statistical machine translation**. 2006.

[29] ROBINSON, TONY. **An application of recurrent nets to phone probability estimation**. 02 1997.

[30] SAETRE, RUNE; SAGAE, KENJI; TSUJII, JUN'ICHI. **Syntactic features for protein-protein interaction extraction.** 6, 01 2007.

[31] SCHUSTER, MIKE; K. PALIWAL, KULDIP. **Bidirectional recurrent neural networks**. 45:2673 – 2681, 12 1997.

[32] SHEN, L.; XU, J. ; WEISCHEDEL, R.. **A new string-to-dependency machine translation algorithm with a target dependency language model.** In: PROCEEDINGS OF ACL-08: HLT, p. 577–585, Columbus, Ohio, 2008.

[33] SRIVASTAVA, NITISH; HINTON, GEOFFREY; KRIZHEVSKY, ALEX AND SUTSKEVER, ILYA; SALAKHUTDINOV, RUSLAN. **Dropout: A simple way to prevent neural networks from overfitting**. 15:1929–1958, 06 2014.

[34] SYDDANSK UNIVERSITET. **Visl - world of visl**. Visual Interactive Syntax Learning, 2018. Acesso em: Junho de 2018.

[35] UI, H.; LI, K.; SUN, R.; SENG CHUA, T. ; YEN KAN, M.. **National university of singapore at the trec 13 question answering main task**. COLING 2004, p. 1284–1290, 2004.

[36] WERBOS, P.. **Backpropagation through time: what it does and how to do it**. 78:1550 – 1560, 11 1990.

[37] ZHANG, YE; RAHMAN, MD MUSTAFIZUR;. **Neural information retrieval: A literature review**. 11 2016.

A
Conjunto de Etiquetas

Cada elemento das *tags* utilizadas neste trabalho possui um significado e todos podem ser encontrados nesta tabela. Eles estão divididos em classes gramaticais, orientação do *head* em relação ao *token* em questão e a distância entre os dois.

Tabela A.1: Dicionário de dados para cada *tag*.

Tag	**Descrição**
adj	Adjetivo
adv	Advérbio
art	Artigo
conj	Conjunção
ec	Prefixos
in	Interjeição
n	Substantivo
num	Numeral
pp	S. Preposicional
pron	Pronome
prop	Nome Próprio
prp	Preposição
v	Verbo
vp	S. Verbal
R	Direita
L	Esquerda
[0,25]	Distância

B
Resumo do modelo

Os vetores de saída de cada camada da rede possuem o seguinte *shape*:

1. *Embedding*: (*batchSize*, *inputLen*, *embedSize*)
2. *Bi-LSTM*: (*batchSize*, *inputLen*, $2 \times lstmSize$)
3. *Dropout*: (*batchSize*, *inputLen*, $2 \times lstmSize$)
4. *Dense*: (*batchSize*, *inputLen*, *outputSize*)

Onde:

1. *inputLen*: Tamanho máximo das sentenças para o treino. Após a definição desse valor, sequências de tamanho superior são truncadas e sequências de tamanho inferior são preenchidas com 0.
2. *batchSize*: Quantidade de sentenças processadas a cada *batch*.
3. *embedSize*: Dimensão da camada de Embedding
4. *lstmSize*: Dimensão do *hidden-state* e do *output* da camada LSTM. Para redes bidirecionais, esse valor é dobrado dado que seu *merge* é feito através de concatenação.
5. *outputSize*: Quantidade de possíveis classes em nosso problema. Em nosso caso essa variável é a quantidade dos diferentes valores de distância que a rede deve ser capaz de classificar.

Uma representação visual do modelo completo pode ser encontrada abaixo.

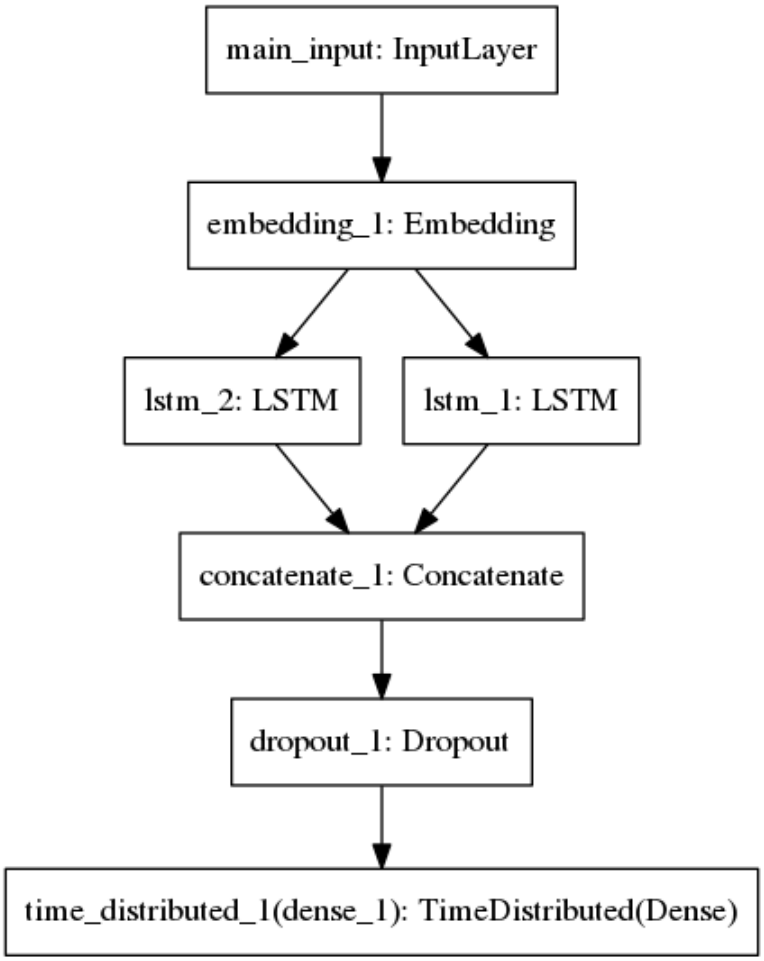

Figura B.1: Visualização da rede completa.