Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo de Jesus Coelho Reis

Anotação morfossintática a partir do contexto morfológico

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 2016

Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo de Jesus Coelho Reis

Anotação morfossintática a partir do contexto morfológico

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

Profª. Maria Cláudia de Freitas
Departamento de Letras – PUC-Rio

Prof. Leandro Guimarães Marques Alvim
Universidade Federal Rural do Rio de Janeiro – UFRRJ

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

Rio de Janeiro, 27 de Setembro de 2016

Eduardo de Jesus Coelho Reis

Graduou-se em Ciência da Computação pela Universidade Federal do Maranhão (São Luís-MA, Brasil). Foi comtemplado pelo Governo Federal para o programa Ciencia sem Fronteiras com duração de 1 ano, ingressando como aluno especial na Lakehead University (Thunder Bay-ON, Canadá). Teve seu Mestrado financiado pela CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior). Atualmente é membro do Grupo de Geofísica Computacional do Tecgraf (PUC-Rio).

Ficha Catalográfica

Coelho Reis, Eduardo de Jesus

Anotação morfossintática a partir do contexto morfológico / Eduardo de Jesus Coelho Reis; orientador: Ruy Luiz Milidiú. – 2016.

v., 91 f: il. ; 30 cm

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

Inclui bibliografia

1. Processamento de linguagem natural;. 2. Part-of-speech tagging;. 3. Redes neurais;. 4. Representação morfológica;. 5. N-Grams;. 6. Regularização esparsa.. I. Milidiú, Ruy Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 620.11

Agradecimentos

Agradeço, sobretudo, ao meu Senhor e Deus Jesus Cristo, pelo amor imerecido, pela minha vida e por tudo aquilo que me foi concedido. Pois sem a Sua Graça, aquilo que minhas mãos produzem e minha mente elabora nunca tornaria-se real, nem ao menos tangível.

Aos meus pais, Francisco e Nilcileia, à minha irmã, Bianca, e à minha avó Socorro, pelo amor, incentivo, companheirismo, educação e por todo sacrifício que fizeram em benefício da minha formação pessoal e profissional.

Aos professores e funcionários dos Departamentos de Informática e de Engenharia Elétrica da PUC-Rio pela contribuição inigualável para a consolidação de minha formação acadêmica e profissional. Em especial ao meu orientador Ruy Luiz Milidiú, pela orientação, apoio, confiança; ao professor Marcelo Gattass, por toda sua solicitude desde que ingressei na PUC-Rio; aos professores Waldemar Celes, Raul Feitosa, Raimundo Sampaio e Rodrigo de Lamare por todo ensinamento e incentivo que me passaram.

Aos professores do Departamento de Informática da UFMA, em especial aos professores Aristófanes Corrêa, Anselmo Paiva, Carlos de Salles e Geraldo Braz pelo incentivo e ajuda no ingresso à pós-graduação.

Aos professores Leandro Alvim e Cláudia Freitas, pela participação na banca examinadora.

Aos amigos que têm sido irmãos, Pedro Henrique, Ramon Coelho, Juliana Leite e Aline Nascimento, por toda a cordialidade, fraternidade e companheirismo durante todo esse tempo.

Agradeço ao meu amigo Rafael Rocha sempre foi muito solícito em ajudar e discutir ideias. Aos meus colegas da PUC-Rio pelos momentos de estudo, diversão e convivência. Em especial, Augusto Ícaro, Daniel Guimarães e Gabriel Ligneul. Aos amigos que me acompanham desde a graduação, Sasha Nícolas, e toda a galera do MA-Rio Bros por todos os momentos de descontração.

Ao pessoal do Tecgraf e à equipe do Suporte; a todas as pessoas do grupo de Geofísica Computacional, do qual faço parte; às pessoas que me ajudaram compartilhado o uso de suas máquinas para processar dado. Em especial, agradeço à Equipe 3, que sempre foi muito compreensível permitindo que eu me ausentasse nos períodos mais intensos do mestrado.

À PUC-Rio, à CAPES, e ao Tecgraf, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

Enfim, agradeço à todos aqueles que acreditaram em mim e que, de alguma forma, contribuíram para a realização deste trabalho. Obrigado!

Resumo

Coelho Reis, Eduardo de Jesus; Milidiú, Ruy Luiz. **Anotação morfossintática a partir do contexto morfológico** . Rio de Janeiro, 2016. 91p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Rotular as classes gramaticais ao longo de uma sentença—*part-of-speech tagging*—é uma das primeiras tarefas de processamento de linguagem natural, fornecendo atributos importantes para realizar tarefas de alta complexidade. A representação de texto a nível de palavra tem sido amplamente adotada, tanto através de uma codificação esparsa convencional, e.g. *bag-of-words*; quanto por uma representação distribuída, como os sofisticados modelos de *word-embedding* usados para descrever informações sintáticas e semânticas. Um problema importante desse tipo de codificação é a carência de aspectos morfológicos. Além disso, os sistemas atuais apresentam uma precisão por *token* em torno de 97%. Contudo, quando avaliados por sentença, apresentam um resultado mais modesto com uma taxa de acerto em torno de $55-57\%$. Neste trabalho, nós demonstramos como utilizar *n-grams* para derivar automaticamente atributos esparsos e morfológicos para processamento de texto. Essa representação permite que redes neurais realizem a tarefa de POS-*Tagging* a partir de uma representação a nível de caractere. Além disso, introduzimos uma estratégia de regularização capaz de selecionar atributos específicos para cada neurônio. A utilização de regularização embutida em nossos modelos produz duas variantes. A primeira compartilha os *n-grams* selecionados globalmente entre todos os neurônios de uma camada; enquanto que a segunda opera uma seleção individual para cada neurônio, de forma que cada neurônio é sensível apenas aos *n-grams* que mais o estimulam. Utilizando a abordagem apresentada, nós geramos uma alta quantidade de características que representam afeições morfossintáticas relevantes baseadas a nível de caractere. Nosso POS *tagger* atinge a acurácia de $96,67$ % no *corpus* Mac-Morpho para o Português.

Palavras-chave

Processamento de linguagem natural; Part-of-speech tagging; Redes neurais; Representação morfológica; N-Grams; Regularização esparsa.

Abstract

Coelho Reis, Eduardo de Jesus; Milidiú, Ruy Luiz (Advisor). **Morphosyntactic annotation based on morphological context** . Rio de Janeiro, 2016. 91p. MSc. Dissertation – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Part-of-speech tagging is one of the primary stages in natural language processing, providing useful features for performing higher complexity tasks. Word level representations have been largely adopted, either through a conventional sparse codification, such as bag-of-words, or through a distributed representation, like the sophisticated word embedded models used to describe syntactic and semantic information. A central issue on these codifications is the lack of morphological aspects. In addition, recent taggers present per-token accuracies around 97%. However, when using a per-sentence metric, the good taggers show modest accuracies, scoring around $55-57\%$. In this work, we demonstrate how to use n-grams to automatically derive morphological sparse features for text processing. This representation allows neural networks to perform POS tagging from a character-level input. Additionally, we introduce a regularization strategy capable of selecting specific features for each layer unit. As a result, regarding n-grams selection, using the embedded regularization in our models produces two variants. The first one shares globally selected features among all layer units, whereas the second operates individual selections for each layer unit, so that each unit is sensible only to the n-grams that better stimulate it. Using the proposed approach, we generate a high number of features which represent relevant morphosyntactic affection based on a character-level input. Our POS tagger achieves the accuracy of 96.67 % in the Mac-Morpho corpus for Portuguese.

Keywords

Natural language processing; Part-of-speech tagging; Neural networks; Morphological representation; N-Grams; Sparse regularization.

Sumário

1 Introdução **13**
1.1 Panorama 13
1.1.1 Aprendizado baseado em regras 14
1.1.2 Aprendizado probabilístico 15
1.1.3 Representação de domínio 16
1.1.4 Benchmark 17
1.2 O problema 18
1.3 Propósito do estudo 20
1.4 Contribuições 21
1.5 Organização do documento 21

2 Fundamentação Teórica **22**
2.1 Representação de Domínio 23
2.1.1 Tipos de Variáveis 24
2.1.2 Categorização de Atributos 24
2.1.3 Representação de Texto 27
2.1.3.1 Representação Baseada em Vocabulário 27
2.1.3.2 Templates baseados em Regras 28
2.1.3.3 Word-Embedding 29
2.2 Classificadores Linear Binários 29
2.2.1 Perceptron 30
2.2.1.1 Raciocínio: Mapeamento de Domínio 30
2.2.1.2 Generalização: Aquisição de Conhecimento 32
2.2.1.3 Representação do Modelo 33
2.2.2 Gradiente Descendente 34
2.2.2.1 Abordagens 35
2.2.2.2 Algoritmos de otimização 36
2.2.2.3 Considerações 38
2.2.3 Regressão Logística 39
2.2.3.1 Sigmóide 39
2.2.3.2 Erro Quadrático 40
2.2.3.3 Entropia Cruzada 40
2.2.3.4 Considerações 41
2.3 Classificadores Multiclasse 43
2.3.1 Estratégias Gerais 43
2.3.2 Perceptron Multiclasse 44
2.3.3 Softmax 45
2.4 Redes Neurais Artificiais 46
2.5 Sobreajuste 47
2.5.1 Balanceamento entre Bias e Variância 47
2.5.2 Dropout 48
2.5.3 DropConnect 49

3 Descritores Morfológicos **51**

3.1 Sistemas Baseados em N-grams 51
3.2 Representação da Palavra 53
3.2.1 Mapa de Índices 54
3.2.2 Palavra Morfológica 54
3.2.3 Contexto Morfológico 57
3.3 Considerações 57

4 Regularização de Domínio **60**
4.1 Definições 61
4.2 Perceptron Esparso 62
4.3 Regularização Compartilhada 63
4.3.1 Erro de Predição 64
4.3.2 Gradiente da Perda 64
4.3.3 Normalização do Gradiente 65
4.4 Regularização Distribuída 65
4.5 Considerações 66

5 Anotação Morfossintática **71**
5.1 A tarefa 71
5.1.1 Corpora 72
5.1.2 Erros Datilográficos 73
5.1.3 Estatísticas dos Corpora 73
5.2 Representação de Domínio 75
5.3 Anotador Morfossintático 76
5.3.1 Ambiente Computacional 76
5.3.2 Modelo de Aprendizagem 77
5.3.3 Experimentos 77
5.4 Regularização de Domínio 79
5.4.1 Regularização com Erro de Predição 80
5.4.2 Regularização com Gradiente da Perda 80
5.4.3 Regularização Distribuída 81
5.5 Experimentos Finais 83
5.5.1 Anotador Simples 83
5.5.2 Regularização Offline 84

6 Conclusão **85**

A Demonstrações Complementares **86**
A.1 Sigmóide 86
A.2 Entropia Cruzada 86

Referências bibliográficas **88**

Lista de figuras

2.1 Ilustração do processo de aprendizagem em que o modelo de aprendizagem é definido por um conjunto hipótese $\mathcal{H}$ e um algoritmo de aprendizagem $\mathcal{A}$. 23
2.2 Partição do domínio. Comparação entre modelo linear W e polinomial P. 25
2.3 Ilustração do modelo de classificação da árvore de decisão. 26
2.4 Partição de domínio pelo modelo da árvore de decisão e reprojeção do separador linear W^*. 26
2.5 Ilustração de indução de atributos em que cada região do domínio $\mathcal{X}$ é representado por uma dimensão propria. 27
2.6 Ilustração da utilização da representação distribuida através de word-embedding. 29
2.7 Modelo de Neurônio de McCulloch-Pits (17): A resposta de um neurônio aos seus estímulos elétricos pode ser modelada através uma função de ativação o total da soma ponderada destes estímulos. Imagem retirada de http://www.controlglobal.com/articles/2006/221/. 30
2.8 Ilustração de um classificador linear binário. Neste exemplo, o classificador é representado uma reta que passa pela origem. 31
2.9 Ilustração da solução de um problema de minimização através de graduente descendente. Neste exemplo temos uma função objetivo convexa. Imagem retirada de http://sebastianraschka.com/faq/docs/closed-form-vs-gd.html. 35
2.10 Comparação do SGD sem momentum, à esquerda, e com momentum, à direita. Imagem retirada de (19). 37
2.11 Ilustração do modelo de uma rede MLP 3-4-1 Imagem retirada de ftp://ftp.inf.puc-rio.br/pub/docs/theses/ 04_PhD_silva.pdf. 46
2.12 Ilustração do uso de *dropout* em uma rede neural. (a) representa uma rede neural comum com duas camadas intermediárias e (b) representa uma rede "afinada" resultante do dropout aplicado em (a). Imagem retirada de (35). 49

3.1 Ilustração de *binary drigrams*. Neste exemplo, temos um dicionário de palavras de tamanho três, e um alfabeto de tamanho quatro. No total, são geradas três matrizes de transições. Imagem adaptada de (24) 52
3.2 Ilustração da representação w da palavra através da ocorrência dos *trigrams* presentes na palavra original w_0. 55
3.3 Ilustração da abordagem convolucional para a extração de informações de contexto a partir de uma janela de palavras de tamanho três. 57

5.1 Histograma de distribuição do *tag set* do Mac-Morpho v1. Este histograma considera apenas as 21 das suas 41 *tags*, em que as *tags* de pontuação são desconsideradas na avaliação dos sistemas que utilizam essa versão. 74
5.2 Histograma de distribuição do *tag set* do Mac-Morpho v3. Nessa versão, além de apresentar um *tag set* de 26 categorias, as *tags* de pontuação são agrupadas econsideradas na avaliação do sistema. 74
5.3 Experimento com *Softmax* Esparso que ilustra o comportamento da seleção de atributos com base no erro de predição. Cada iteração de L utiliza um modelo treinado ao longo de 25 épocas. Neste experimento adotamos $P_{dropout} = 0$. 81
5.4 Experimento com *Softmax* Esparso que ilustra o comportamento da seleção de atributos com base gradiente da perda. Cada iteração de L utiliza um modelo treinado ao longo de 25 épocas. Neste experimento adotamos $P_{dropout} = 0$. 82
5.5 Experimento com *Softmax* Esparso que demonstra o comportamento da seleção de atributos com base na regularização distrbuida. Cada iteração de L utiliza um modelo treinado ao longo de 25 épocas. Neste experimento adotamos $P_{dropout} = 0$. 82

Lista de tabelas

1.1 Comparação entre os sistemas de POS-*Tagging* para o corpus Mac-Morpho v1. 18

3.1 Número de combinações de *n-grams* considerando apenas letras do alfabeto 58

5.1 Tamanho do Corpora 72
5.2 Estado-da-Arte das versões do Mac-Morpho 72
5.3 Estatísticas adicionais do corpora utilizado. Observamos como o tamanho do vocabulário e a quantidade de OOV variam dependendo da presença de sensibilidade à caixa da letra. 75
5.4 Dimensionalidade da representação de domíno do Mac-Morpho v3. 76
5.5 Experimento utilizando o MLP com *Dropout* em que combinamos a representação de domínio baseada em *trigrams* de caracteres. Este experimento utiliza o conjunto de desenvolvimento para a avaliação, onde fazemos uma busca do número de neurônios por camanda ao longo de 25 épocas de treinamento. 78
5.6 Experimento utilizando o MLP sem *Dropout* em que combinamos a representação de domínio proposta para aprender um anotador morfossintático. Utilizamos o conjunto de desenvolvimento para a avaliação os efeitos de diferentes tamanhos de *n-gram*. 79
5.7 Experimento utilizando o MLP com *Dropout* em que combinamos a representação de domínio proposta para aprender um anotador morfossintático. Utilizamos o conjunto de desenvolvimento para a avaliação os efeitos de diferentes tamanhos de *n-gram*. 79
5.8 Desempenho do anotador morfossintático simples avaliado no *corpora* do MacMorpho. 84
5.9 Desempenho do anotador morfossintático com regularização *offline* avaliado no *corpora* do MacMorpho. 84

Remember that all models are wrong; the practical question is how wrong do they have to be to not be useful.

George Box ,
Empirical Model-Building and Response Surfaces, p. 74. 1987 .

1
Introdução

O desenvolvimento tecnológico das últimas décadas tem ocasionado uma produção exponencial de dados não estruturados, tal como conteúdos de texto e fala. Assim, escrever programas para automaticamente processar dado bruto e gerar informação estruturada se tornou um recurso valioso.

Um dos grandes desafios em processamento de linguagem natural, do inglês Natural Language Processing—NLP, é fornecer informações linguísticas necessárias para um computador realizar tarefas de alto nível baseadas em linguagem. Técnicas de aprendizado de máquina vem sendo amplamente utilizas em NLP (1), permitindo a construção de sistemas que, se programados manualmente, consumiriam uma quantidade inviável de tempo e esforço humano. Como resultado, observamos o surgimento de ferramentas que tornaram-se indispensáveis em nosso cotidiano, como motores inteligentes de busca e assistentes virtuais que operam por comando de voz.

A fim de se realizar tarefas de NLP de alta complexidade, é necessário primeiramente quebrar o texto em seus elementos básicos, identificando suas estruturas linguísticas fundamentais, e a partir delas inferir incrementalmente o conhecimento ali representado. A anotação morfossintática, em inglês chamada de *Part-of-Speech*—POS—*Tagging*, é uma das primeiras tarefas de NLP e consiste em rotular as classes gramaticais das palavras ao longo de uma dada sentença. Esta tarefa é o principal objeto de estudo deste trabalho, sendo um estágio fundamental por fornecer atributos para tarefas de mais alta complexidade como *Dependency Parsing*, *Name Entity Recognition* e *Semantic Role Labeling*.

1.1
Panorama

A disponibilização de grandes acervos anotados, ou *corpora*, difundiu métodos empíricos voltados para a resolução de tarefas de NLP.

Os primeiros sistemas desenvolvidos apresentavam tanto abordagens descritivas, baseadas em regras, quanto probabilísticas (1). Sendo a última bem difundida entre os sistemas de NLP, pois geralmente apresentam resultados superiores às abordagens descritivas. Com isso, pesquisas recentes têm dado

enfoque na maneira de como estes sistemas representam texto, uma vez que os modelos probabilísticos necessitam de bons descritores de domínio, elevando cada vez mais o estado-da-arte.

Nesta seção, apresentamos um panorama dos sistemas de anotação morfossintática que retrata o uso de modelagens baseadas em regras. Em seguida, recapitulamos o avanço dos modelos através de abordagens probabilísticas, ainda que compreendam características baseadas em regras. Por último, mostramos como abordagens probabilísticas têm se estendido em busca de representações de contexto de representações mais automática e robusta.

1.1.1 Aprendizado baseado em regras

O aprendizado baseado em regras consiste em construir um sistema de NLP através da elaboração de regras, tais como regras:

- léxicas: classe gramatical da palavra;
- morfológicas: informação de estrutura da palavra, como prefixos, sufixos, lemas;
- sintáticas: relacionamento estrutural entre as palavras.

Estas regras podem ser formuladas manualmente a partir de análises estatísticas de fenômenos linguísticos específicos para uma determinada tarefa. Por exemplo, os verbos no infinitivo costumam apresentar as terminações "ar", "er" e "ir". Ao observar que este fenômeno linguístico é frequente, pode-se utilizá-lo na composição de uma regra que auxilia na identificação de verbos. Esta regra poderia ser utilizada na tarefa de anotação morfossintática.

Identificar e correlacionar padrões que auxiliem a construção de regras para uma determinada tarefa de NLP torna a construção de modelos um processo custoso por se tratar de um processo manual e por vezes limitado a um *corpus* em particular.

Brill (4) introduz o *Transformation Based Learning*—TBL. Esse é um *framework* de aprendizado que ameniza o gargalo das definições manuais utilizando *templates* para a aquisição automática de regras (3). Assim, tarefas de NLP podem ser solucionadas através de correções incrementais a partir de anotadores iniciais. Dos Santos (5) propõe o *Entropy Guided Transformation Learning*—ETL que generaliza o TBL. Seu sistema utiliza de Árvores de Decisões para gerar *templates* de forma automática por meio de *Information Gain*—IG. As abordagens baseadas em TBL apresentam um desempenho competitivo se comparadas aos métodos probabilísticos (6). Além disso, elas

permitem uma interpretabilidade das regras aprendidas fornecendo pistas das estruturas linguísticas capturadas no processo de aprendizagem.

Diversos sistemas de aprendizagem têm incorporado o uso de IG em suas técnicas, possibilitando a criação de *frameworks* mais robustos. O *Entropy Guided Structured Learning*—ESL, introduzido por Fernandes (7), apresenta uma generalização do ETL chamada *Entropy Guided Feature Generation*. Esse mecanismo produz atributos esparsos a partir da binarização de inúmeras características contextuais obtidas através de *templates* gerados automaticamente. Os atributos induzidos são então utilizados como entrada para um algoritmo de *Structured Learning* (16). De forma semelhante, o *Incremental Feature Induction and Selection*—IFIS, proposto por Motta (8), também utiliza de IG para realizar uma etapa de *Entropy Guided Feature Induction* que induz novas características ao longo de várias iterações. Para cada iteração deste *framework*, as características mais significativas são selecionadas e utilizadas na iteração seguinte, permitindo a indução de características cada vez mais complexas.

1.1.2 Aprendizado probabilístico

Em contrapartida, o aprendizado baseado em abordagens probabilísticas está presente em grande parte dos sistemas de NLP, visto que apresentam resultados competitivos se comparados sistemas elaborados através de regras artesanais. Ratnaparkhi (15) apresenta um modelo baseado em *Maximum Entropy*—ME—para a tarefa de POS–*Tagging*, em que o contexto morfossintático das palavras é descrito por atributos binários chamados *feature-vector representation*. O ME combina as vantagens das representações ricas de atributos baseados em regras, gerando uma distribuição de probabilidade de anotações para cada palavra. Dentre as abordagens probabilísticas, destacamos também o *hidden Markov model*—HMM (2). Esta é uma abordagem estocástica que analisa a tarefa de POS–*Tagging* como um problema de predição de sequência, utilizando o algoritmo de Viterbi que garante encontrar a sequência de eventos, i.e. classes gramaticais, mais provável para a anotação de todas as palavras da sentença.

Como alternativa aos modelos HMM, Collins (16) introduz o *Structured Perceptron*, uma generalização do algoritmo do *Perceptron* de Rosenblatt (14). Collins faz uso do *feature-vector representation* para estimar os parâmetros utilizados pelo algoritmo de Viterbi. Essa é uma abordagem notável por proporcionar que técnicas de aprendizado de máquina usualmente utilizadas em tarefas de classificação sejam adaptadas para tarefas de NLP.

Embora os processos de aprendizagem dos modelos mencionados acima

adotem uma abordagem probabilística, a representação de domínio ainda dispõe de um estilo baseado em regras, uma vez que os *feature-vector representation* são adquiridos através de *templates*.

Recentemente, redes neurais artificiais de múltiplas camadas—ANN—têm sido amplamente utilizadas para solucionar tarefas de NLP. Collobert (9) demonstra como redes neurais profundas—DNN—possibilitam a construção de sistemas genéricos. Ao invés de depender de estruturas elaboras manualmente, em que características específicas de uma tarefa são assimiladas, a própria arquitetura da ANN se encarrega de aprender estas representações ao longo de suas várias camadas. Tal arquitetura associa cada palavra a um vetor de características por meio de um procedimento chamado *word-embedding* e produz uma matriz. Para cada palavra da sentença, essa matriz contém a probabilidade associada com cada *tags* produzidos pelo anotador. Ao fim, o algoritmo de Viterbi utiliza dessa matriz de probabilidades para de computar a sequência de *tags* de maior probabilidade.

1.1.3 Representação de domínio

O desempenho de algoritmos de aprendizado de máquina depende diretamente de como o domínio do problema é representado, podendo-se adotar um esquema de representação local ou distribuída.

Representação Local O esquema de representação local utiliza de dimensões específicas para descrever cada elemento do domínio, resultando em uma representação esparsa. Este esquema supõe a utilização de um vocabulário, que pode ser formado a partir do conjunto de todas as palavras do conjunto de treinamento. O vocabulário, em geral, costuma ter dezenas e até centenas de milhares de palavras, dependendo do idioma e do corpus utilizado. Assim, cada palavra é atribuída a um vetor binário de tamanho igual ao número de palavras contidas em um dado vocabulário, estando ativo apenas no índice referente àquela palavra. Essa representação é adotada pela abordagem *bag-of-words*, que descreve um documento como a soma dos vetores de todas as palavras nele contidos, resultando em um único vetor que contém o número de ocorrências de cada palavra ao logo do texto. Apesar de possuir uma descrição simples, essa abordagem foi explorada por diferentes sistemas de NLP, por exemplo para solucionar a tarefa de categorização de texto (10).

Representação Distribuída Em contraste, a representação de palavras por um esquema distribuído utiliza uma quantidade menor de atributos para re-

presentar cada palavra, possuindo uma baixa dimensionalidade, em que todas as dimensões contém uma representatividade para cada elemento do domínio. Cada palavra é associada a um vetor, sendo um vocabulário de palavras representado inteiramente por uma matriz de características que é previamente computada. Deste modo, é possível selecionar a representação distribuída de cada palavra a partir do produto interno entre sua representação local e essa matriz. O *word-embedding* utiliza deste artifício para embutir tal representação no inicio da rede neural permitindo um refinamento desses vetores de característica específico para a tarefa. Isso possibilita que informações sintáticas e semânticas das palavras sejam capturadas por DNN.

Dificuldades Embora *word-embedding* seja uma representação poderosa, essa abordagem trata as palavras como entidades independentes, falhando em capturar relações explícitas de variações morfológicas entre palavras. Por exemplo, palavras derivadas de um mesmo radical, ou que possuam uma mesma terminação. Consequentemente, essa abordagem apresenta uma ausência de informações morfossintáticas de palavras fora-do-vocabulário ou out-of-vocabulary—OOV, uma vez que estas não possuem uma representação própria. Em geral, as abordagens baseadas em *word-embedding* tratam palavras OOV como sendo "raras", compartilhando o mesmo vetor de características. Portanto, essa representação apresenta uma carência de aspectos de forma, uma vez que estruturas internas da palavra são ignoradas. Collobert (9) atenua esse problema através de embeddings adicionais a partir da capitalização da palavra, como também através de seu sufixo; sendo o último exclusivo para o sistema POS–*Tag*. Através de uma abordagem similar, Fonseca (11) propõe um sistema que utiliza *neural language model*—NLM para pré-calcular os vetores de características, além do uso de *embeddings* adicionais de prefixos e sufixos de diferentes tamanhos. Ainda seguindo essa abordagem, Dos Santos (12) propõe o CharWNN que utiliza uma representação baseada na junção de *embeddings* tanto a nível de palavra, quando ao nível de caractere.

1.1.4 Benchmark

A principal métrica de avaliação desses sistemas é através da contagem de palavras classificadas corretamente, ou *per-word accuracy*—PWA. Uma métrica secundária apresentada pelos sistemas mais recentes mede o erro de classificação contabilizando apenas as palavras OOV, que são aquelas que estão fora do vocabulário de palavras utilizado no processo de treinamento. Diversos sistemas para a tarefa de POS–*Tagging* tem sido propostos. A Tabela 1.1

Tabela 1.1: Comparação entre os sistemas de POS-*Tagging* para o corpus Mac-Morpho v1.

Ano	Sistema	PWA (%)	OOV (%)
2015	NLM	**97,57**	**93,38**
2014	CharWNN	97,47	92,49
2014	IFIS	97,13	–
2012	ELS	97,12	–
2009	ETL_{CMT}	96,94	–
2016	Este Trabalho	96,67	90,07
1995	TBL	96,60	–

apresenta o *benchmark* para essa tarefa utilizando o corpus Mac-Morpho v1 para o Português.

1.2 O problema

A anotação morfossintática, ou *part-of-speech*—POS *tagging*, é o processo de associar um POS ou marcador léxico para cada palavra em um texto (2), rotulando as palavras em categorias de classes gramaticais através do papel que cada uma desempenha em seu respectivo contexto. A seguir, temos um exemplo extraído do *corpus* Mac-Morpho, cujos marcadores possuem os seguintes significados: `ART`=artigo, `ADJ`=adjetivo, `N`=nome, `NUM`=numeral, `PREP`=preposição e `V`=verbo.

```
A/ART desertificação/N tornou/V crítica/ADJ a/ART
produtividade/N de/PREP 52/NUM mil/NUM km²/N em/PREP a/ART
região/N ./.
```

Ainda que exista um consenso entre as categorias básicas, as peculiaridades presentes em cada caso dificultam o estabelecimento de um único conjunto "correto" de marcadores, até mesmo dentro do mesmo *corpus*. O Mac-Morpho, por exemplo, apresenta diferentes *tagsets* em suas três versões.

Como mostrado na Seção 1.1, diversas abordagens já foram apresentadas para resolver esse problema. Os atuais sistemas estado-da-arte apresentarem uma precisão satisfatória por palavra, em torno de 97%. Contudo, ao serem avaliados por sentença apresentam resultados mais modestos com uma taxa de acerto em torno de $55 - 57\%$ (13). Portanto, ainda há espaço para melhorias no intuito de reduzir erros propagados para as tarefas posteriores.

Para elaborar sistemas de POS–*Tagging* robustos, é necessário tratar alguns fenômenos que dificultam essa tarefa, como palavras homógrafas, pala-

vras com um baixo número de ocorrência, palavras fora-do-vocabulário, e erros ortográficos.

Palavras homógrafas são aquelas que possuem a mesma escrita porém com significado diferente. Logo, é comum nos depararmos com palavras que assumem papeis diferentes dependendo do contexto em que se encontram, como na sentença abaixo.

```
Verdes/V !/! Vestes/V verdes/ADJ vestes/N ./.
```

Palavras que possuem um baixo número de ocorrência têm grandes chances de apresentarem descrições ineficazes, visto que o aprendizado baseado em modelos estatísticos requer uma quantidade de amostras suficiente alta para gerar descrições que possuem boa representatividade. Esse problema se agrava em abordagens tradicionais que tratam palavras como entidades totalmente independentes, ignorando suas estruturas internas, visto a frequência de ocorrência de cada palavra é menor do que a frequência das subsequências de caracteres nela contidos.

Uma forma de lidar com palavras de baixa frequência é utilizar as n palavras mais frequentes do *corpus* para definir um dicionário (9). Assim, palavras que ficam de fora desse dicionário são substituídas e agrupadas por uma única palavra especial "$RARE$".

Palavras fora-do-vocabulário, ou palavras "novas", são aquelas que não apresentam nenhuma participação no aprendizado do modelo por estarem disponíveis apenas no conjunto de teste. Como resultado, elas não possuem um representação própria em abordagens tradicionais, em geral também associadas a mesma representação da palavra especial"$RARE$".

Por fim, uma outra dificuldade para os sistemas de anotação morfossintática é a ocorrência de palavras que contenham erros ortográficos ou datilográficos. Pois, um erro pode produzir uma palavra fora-do-vocabulário, representando palavras frequentes como "$RARE$". Por outro lado, um erro pode produzir uma outra palavra do dicionário, porém com significado totalmente inesperado para o dado contexto. Apesar de ser relevante para aplicações reais, os sistemas atuais não tem abordado essa questão. Em geral, os *corpora* utilizados pela maioria das tarefas de anotação morfossintática não apresentam especificações de erros de escrita ou digitação.

1.3

Propósito do estudo

Na seção anterior, apresentamos algumas questões inerentes da tarefa de anotação morfossintática. Com exceção da ocorrência de palavras homógrafas, os demais problemas estão relacionados ao uso de representação de palavras de forma totalmente independente. A utilização de contexto morfológico nestas representações tem mostrado ser uma abordagem promissora, uma vez que demonstram agregar informações relevantes para a construção de anotadores morfossintáticos (9, 12). Contudo, ainda é necessário elaborar tratamentos para os problemas levantados anteriormente. Visto que, apesar dessa abordagem conter a adição de informações de morfologia importantes, a sentença ainda depende de uma representação independente a nível de palavra.

Diante das dificuldades apresentadas, este estudo tem como propósito explorar a tarefa de anotação morfossintática utilizando técnicas de aprendizado de máquina para capturar aspectos linguísticos de sentenças utilizando exclusivamente o contexto morfológico de suas palavras. Nosso estudo é descrito em quatro partes: representação, aprendizado, regularização e avaliação.

Primeiramente, propomos uma representação local que gera automaticamente atributos morfológicos, descrevendo a palavra através de sua forma. Especificamos cada palavra por meio dos *n-grams* a nível de caractere nela contidos. Desta maneira, produzimos uma representação esparsa do domínio através de seus elementos mais primitivos.

Em seguida, demostramos a relevância destes atributos optando por uma abordagem mais direta. Utilizamos estes atributos para alimentar uma rede neural, que encarrega-se de aprender aspectos morfossintáticos voltados nossa tarefa sem a necessidade de quaisquer especificação adicional. Dispensamos o uso do algoritmo de Viterbi, deste modo a classificação de classes gramaticais é realizada pela própria rede utilizando apenas o contexto local das palavras.

Posteriormente, exploramos estratégias de regularização de domínio. A regularização de domínio tem como objetivo selecionar os atributos mais relevantes para reduzir o tamanho do modelo gerado, além de reduzir o número de operações realizadas. Ainda, introduzimos uma variação desta estratégia de regularização capaz de selecionar atributos durante o aprendizado, além de permitir uma seleção fina de atributos, escolhendo os *n-grams* que mais sensibilizam cada neurônio.

Por fim, avaliamos nossa abordagem comparando-a com o *benchmark* apresentado na Tabela 1.1. Além das métricas PWA e OOV, apresentamos o desempenho do nosso sistema avaliando seu desempenho em um conjunto que contenha erros datilográficos.

1.4 Contribuições

Este trabalho tem três contribuições principais:

- Representação de palavras através de uma alta quantidade de características geradas automaticamente que representam suas feições morfossintáticas.
- Anotador de texto que não descrimina palavras raras e fora-do-vocabulário.
- Regularização de domínio com base em gradiente.

1.5 Organização do documento

Utilizamos a tarefa de anotação morfossintática como uma aplicação alvo para demonstrar os conceitos apresentados neste trabalho. O restante deste documento está organizado da seguinte maneira.

No Capítulo 2, fazemos uma exposição dos fundamentos das técnicas de aprendizado de máquinas e regularização de domínio empregadas neste trabalho.

No Capítulo 3, detalhamos como são gerados os descritores morfológicos que representam palavras a partir de *n-grams*.

No Capítulo 4, introduzimos uma extensão da regularização de dominio capazes de operar seleção de atributos online e por neurônio.

No Capítulo 5, relatamos os experimentos e apresentamos os resultados da anotação morfossintática a partir do contexto morfológico.

Por fim, no Capítulo 6, apresentamos nossas considerações finais.

2
Fundamentação Teórica

No capítulo anterior, apresentamos uma visão geral dos trabalhos mais relevantes para a tarefa de anotação morfossintática. Através dos sistemas apontados, pode-se observar como a transição de modelos baseados em especificações manuais para modelos mais genéricos, tem elevado cada vez mais o estado-da-arte, fazendo uso de representações distribuídas. Destacam-se ainda as dificuldades provenientes da tarefa de anotação morfossintática, enfatizando-se que tais dificuldades são agravadas pela ausência de aspectos morfológicos por parte das representações tradicionais ao nível de palavra. Ao fim, introduzimos a abordagem proposta por essa dissertação. Utilizamos uma representação que se baseia unicamente em *n-grams* a nível de caractere para representar as palavras e seus aspectos morfológicos.

Agora, apresentamos os conceitos básicos de aprendizado de máquina que fundamentam este trabalho, dando suporte para a discussão dos sistemas de anotação morfossintática mais atuais, bem como para a compreensão da metodologia proposta.

Em termos gerais, o processo de aprendizagem pode ser formalizado como a inferência de uma função $f : \mathcal{X} \mapsto \mathcal{Y}$, Figura 2.1. A descrição da função objetivo f é realizada a partir da análise das amostras do domínio $\mathcal{X}$. Para tal, o problema de aprendizado considera dois componentes: um conjunto de hipóteses $\mathcal{H} = \{h\}$, que descrevem f; e um algoritmo de aprendizagem $\mathcal{A}$ que infere um elemento $g \in \mathcal{H}$ que produza uma boa aproximação de f. Juntos, esses componentes definem um modelo de aprendizagem. Redes Neurais Artificiais, descritas posteriormente neste capítulo, são exemplos de conjunto de hipóteses, em que o *back propagation* é utilizado como algoritmo de aprendizagem.

Neste capítulo, discorremos de forma geral sobre o processo de aprendizagem computacional, enfatizando três etapas: representação de domínio, modelos de aprendizagem, e controle de sobre-ajuste.

Na Seção 2.1, extendemos os conceitos representação de domínio apresentados na Seção 1.1.3. Fazemos uma breve descrição sobre tipos de variáveis de domínio e enfatizamos como variáveis categóricas permitem adaptar métodos de classificação para solucionar tarefas de processamento de texto. Na

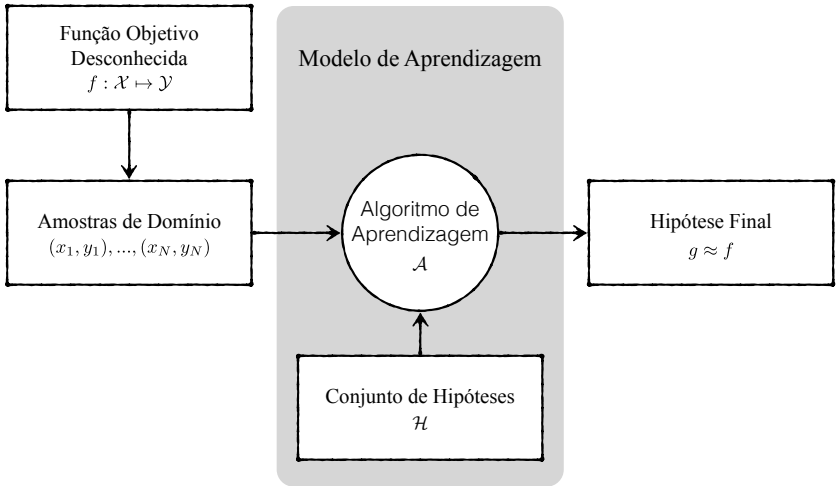

Figura 2.1: Ilustração do processo de aprendizagem em que o modelo de aprendizagem é definido por um conjunto hipótese $\mathcal{H}$ e um algoritmo de aprendizagem $\mathcal{A}$.

Seção 2.2, detalhamos os classificadores lineares elementares e como estes solucionam problemas de tomada de decisão binária. Na Seção 2.3, apresentamos como classificadores lineares são utilizados em problemas com múltiplas classes. Na Seção 2.4, mostramos como estes classificadores são utilizados para compor Redes Neurais Artificiais, permitindo a modelagem de sistemas não-lineares cada vez mais complexos ao longo de múltiplas camadas. Na Seção 2.5, discutimos o sobre-ajuste enfrentado pelos modelos estatísticos e como a técnica de *Dropout* atenua esse problema.

2.1 Representação de Domínio

Em inteligência computacional, a complexidade de um problema está diretamente relacionada com a dificuldade em obter-se o mapeamento que minimize os erros de projeção do domínio no contradomínio.

Em tempos em que o poder de processamento era algo escasso, o esforço humano era voltado para a elaboração de bons descritores que simplificassem o trabalho dos algoritmos de aprendizagem. Assim, parte da complexidade do modelo de aprendizagem era desviada para a representação do domínio. Nesta seção, discutimos como texto é representado categoricamente em que fazemos um contraste com forma de representação numérica.

Primeiramente, introduzimos os tipos de variáveis utilizados na representação de elementos de domínio e contradomínio. Em seguida, mostramos como a indução de variáveis categóricas podem proporcionar um maior grau de liberdade para os algoritmos de aprendizagem. Por fim, detalhamos como

estas variáveis contemplam representações de texto, permitindo a utilização de classificadores numéricos na solução de tarefas de NLP.

2.1.1 Tipos de Variáveis

Pode-se definir variáveis como toda e qualquer medida ou informação que os elementos de um determinado conjunto podem assumir, seja através de valores numéricos ou não numéricos. Desta forma, temos os seguintes conceitos:

- **Variáveis Numéricas**: Também chamadas de variáveis quantitativas, são características que podem ser medidas através de valores numéricos dentro de um intervalo finito ou infinito, podendo ser ainda subdividas em variáveis contínuas e discretas.
- **Variáveis Categóricas**: Diferente das variáveis numéricas, não possuem medidas quantitativas; ao contrário, representam os indivíduos de forma qualitativa através de um conjunto finito de categorias, podendo ainda ser subdividas em nominais e ordinais dependendo da existência de uma relação de ordem entre seus valores.

O tipo das variáveis está relacionado diretamente com a natureza do problema. Por exemplo, em tarefas de aprendizado supervisionado, um contradomínio descrito por variáveis numéricas definindo um problema de regressão, enquanto que quando representado por variáveis categóricas define um problema de classificação.

2.1.2 Categorização de Atributos

A despeito do processo de aprendizagem, Figura 2.1, os algoritmos de aprendizado probabilísticos necessitam de uma representação numérica do domínio para a construção do modelo de predição. Nessa seção, observamos os efeitos do uso de atributos categóricos na partição do espaço amostral em problemas de classificação.

Considere amostras de duas classes dispostas conforme a Figura 2.2. Essa ilustração descreve dois modelos que exemplificam possíveis divisões do espaço amostral: Modelos descritos por um polinômio P são mais flexíveis. Porém, devido a sua complexidade, seu processamento exige um grande esforço computacional. Modelos lineares W são vantajosos por serem mais rápidos e simples para computar. Contudo, a divisão do espaço amostral é feita de forma "grosseira", não capturando nuances do domínio.

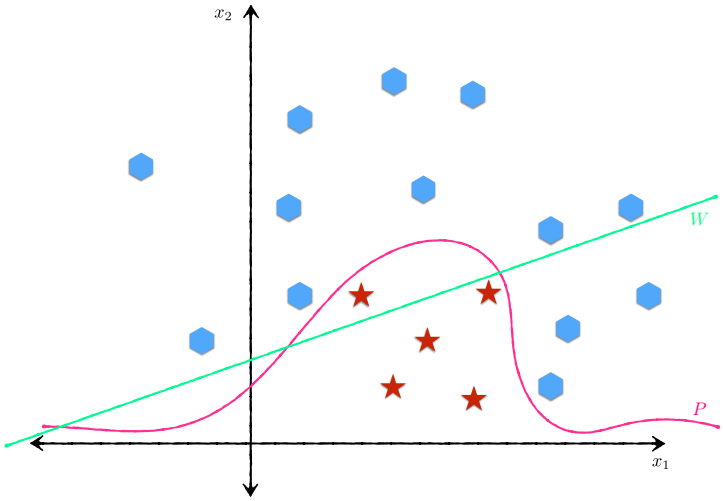

Figura 2.2: Partição do domínio. Comparação entre modelo linear W e polinomial P.

Uma alternativa para aprender modelos que descrevam o domínio de forma mais refinada, porém com as vantagens dos modelos lineares, é descrever o domínio de forma categórica, através de atributos binários. Uma forma ingênua de binarização atributos é dividir o intervalo em que um dado atributo x_i está definido igualmente em n partes. Dessa maneira, o atributo x_i passa a ser descrito por n dimensões, em que apenas uma delas está ativa, valor igual a 1.

De forma mais esperta, binarização de atributos pode ser realizada a partir dos nós dos modelos aprendidos por classificadores baseados em árvores de decisão—DT—binária, Figura 2.3. Os algoritmos de aprendizados tradicionais de árvores de decisões utilizam de ganho de informação para melhor dividir os intervalos do domínio com base em entropia. Na ilustração da Figura 2.4, os dois atributos numéricos (x_1, x_2) seriam substituídos por três atributos binários $(x_1 > a, x_1 > b, x_2 > c)$. Assim, a categorização de atributos numéricos possibilita um remapeamento do domínio $\mathcal{X} \mapsto \mathcal{X}^*$, em que um melhor separador linear pode ser inferido. A projeção desse separador em $\mathcal{X}$ é ilustrada por W^*.

Neste ponto, pode-se observar duas características presentes na binarização de atributos: aumento de dimensionalidade e esparsidade. A correlação entre esses dois fatores dependente da abordagem utilizada na binarização. Ao utilizar árvores de decisões binárias, o número de dimensões ativas é no máximo o tamanho do caminho da raiz até uma folha, esta quantidade é limitada pela altura da árvore h. Por outro lado, o número nós da árvore equivale ao número de total de dimensões obtidas, em torno de 2^n dependendo do balanceamento da árvores.

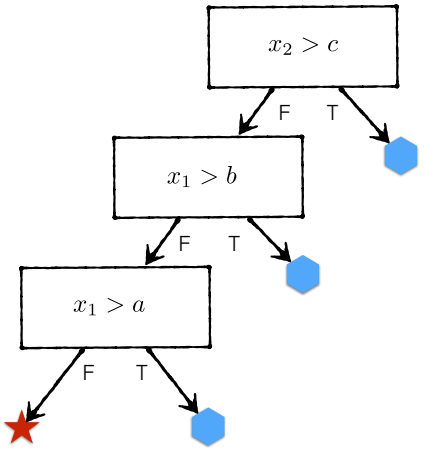

Figura 2.3: Ilustração do modelo de classificação da árvore de decisão.

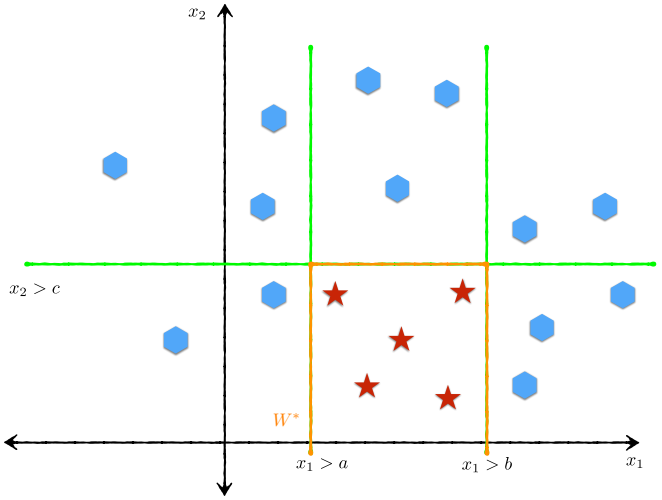

Figura 2.4: Partição de domínio pelo modelo da árvore de decisão e reprojeção do separador linear W^*.

Uma abordagem ainda mais elaborada é a indução de atributos binários a partir do modelo da DT, ilustrada na Figura 2.5. Aqui, os nós da árvore são combinados para gerar *templates*. Ilustramos a indução utilizando o *template* de tamanho três formado pelos três nós descritos na Figura 2.3. O número de dimensões geradas por um *template* é de até $2^n - 1$, em que apenas uma está ativa por *template*. Para um template de tamanho $n = 3$ teríamos sete dimensões geradas, porém em nosso exemplo temos apenas um total de cinco

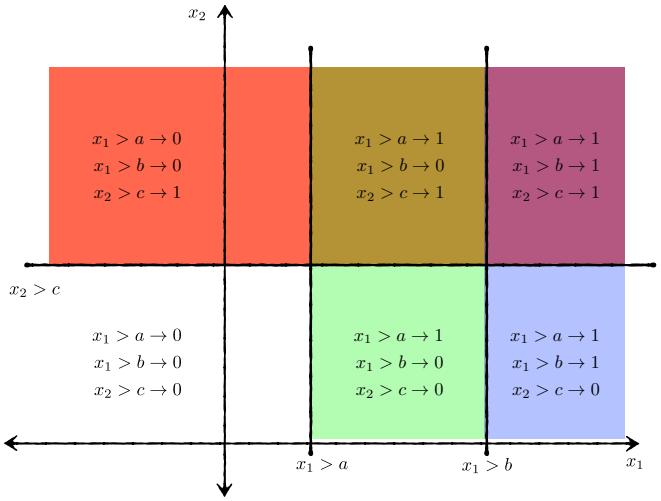

Figura 2.5: Ilustração de indução de atributos em que cada região do domínio $\mathcal{X}$ é representado por uma dimensão propria.

pois duas não ocorrem por ser inválidas:

$$x_1 > a \to 0 \qquad x_1 > a \to 0$$
$$x_1 > b \to 1 \qquad x_1 > b \to 1$$
$$x_2 > c \to 0 \qquad x_2 > c \to 1$$

2.1.3 Representação de Texto

A seção anterior introduz algumas implicações do uso de atributos categóricos, dentre elas destaca-se a esparsidade das representações obtidas. A seguir, detalhamos as formas tradicionais de representação de texto, tais como a representação baseada em vocabulário, representações baseadas em regras e *word-embedding*.

2.1.3.1 Representação Baseada em Vocabulário

A representação baseada em vocabulário está presente na grande maioria dos sistemas de processamento de texto. Esta é uma representação local, em que os elementos de domínio são representados por dimensões específicas. Cada palavra é descrita por um vetor esparso w de tamanho igual ao tamanho do vocabulário V^{wrd} das palavras de um conjunto de treinamento. Nessa representação, apenas a dimensão cujo índice referência a palavra em questão está ativa.

Tal abordagem trata as palavras como elementos atômicos, adotando a premissa de que estas constituem categorias nominais totalmente independentes. Apesar de equivoco, assumir que as palavras são elementos independentes, facilita a elaboração ágil de sistemas de NLP a partir de uma representação simples. Neste caso, o próprio modelo encarrega-se de refinar as relações sintáticas e semânticas entre as palavras dado uma quantidade suficientemente grande de amostras.

Ainda que uma representação baseada em vocabulário simplifique a elaboração de sistemas, essa representação utiliza apenas a posição em que uma palavra ocorre no vocabulário. Quando presente no vocabulário, a informação obtida sobre a palavra basear-se unicamente na dimensão que é ativa. Assim, palavras fora-do-vocabulário, que ocorrem apenas no conjunto de teste, apresentam uma carência de representação por não possuírem um índice próprio ativo, uma vez que não participaram da etapa de treinamento. Em geral, as palavras fora-do-vocabulário são tratadas como "raras", compartilhando o mesmo índice.

2.1.3.2 Templates baseados em Regras

Uma outra representação domínio utilizado em tarefas de processamento de texto tem com base o uso de *templates* baseado em regras. Um exemplo disso é o anotador morfossintático TBL (4), que usa um classificador ingênuo para gerar uma anotação inicial. A partir dessa anotação, é possível representar o domínio através de templates que mapeiem o contexto morfológico. Abaixo exemplificamos três templates:

- `pos[0]=ART pos[1]=ART` Esse *template* avalia a anotação inicial da palavra corrente e da palavra seguinte. Então um atributo é ativo dependendo do valor do POS.
- `pos[0]=N pos[-1]=N pos[-2]=ART` Neste, são avaliadas as anotações iniciais de POS da palavra corrente e das duas anteriores, em um índice do vetor de representação é ativo dependendo da combinação de POS satisfeita.
- `pos[0]=ART pos[1]=V word[0]=a` Semelhante aos *templates* anteriores, este exemplo considera o contexto inicial de POS, além de considerar a própria palavra na posição corrente.

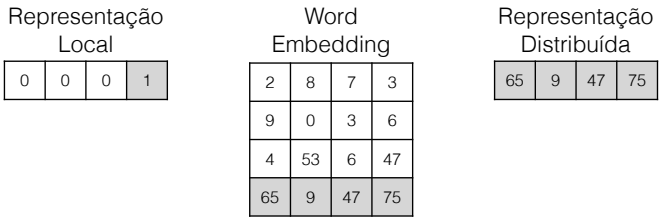

Figura 2.6: Ilustração da utilização da representação distribuida através de word-embedding.

2.1.3.3 Word-Embedding

A técnica de *word-embedding* é uma forma esperta de se associar uma representação distribuída a uma variável categórica que representa a palavra, dando assim mais liberdade para os algoritmos de aprendizagem.

Observamos na Figura 2.6 que isso pode ser feito através de uma simples multiplicação de matrizes. Essa matriz é calculada em uma etapa de treinamento não-supervisionado que busca aprender uma quantidade de atributos, inicialmente estipulada como parâmetro fornecido, relacionem as palavras conforme o contexto em que elas aparecem. Conforme Collobert (9) demonstra, esses atributos aprendidos correlacionam as palavras através de propriedades sintáticas e semânticas. Uma vez aprendida, essa matriz pode ser refinada durante o aprendizado da tarefa em questão através de *backpropagation*, descrito posteriormente neste trabalho.

2.2 Classificadores Linear Binários

Para desenvolver sistemas inteligentes capazes de aprender e tomar de decisões, é importante definir modelos que possuam as habilidades de *percepção*, *generalização* e *raciocínio* necessárias para exercer tarefas complexas. Uma vez que dispomos de dados amostrais armazenados em hardware para representação do domínio—*percepção*, precisamos ainda abstrair e manter o conhecimento ali representado–*generalização*, para então definir como a informação adquirida é aplicada para desempenhar uma tarefa—*raciocínio*, (14).

Antes de compreender o funcionamento de algoritmos mais complexos, é necessário buscar por *insights* a partir das estruturas elementares utilizadas no processos de aprendizagem computacional, i.e. *generalização* e *raciocínio*, analisando as técnicas mais fundamentais que os compõem.

Nesta seção, discutimos os classificadores mais básicos, descrevendo como eles se propõem a resolver problemas de classificação binária. Inicialmente,

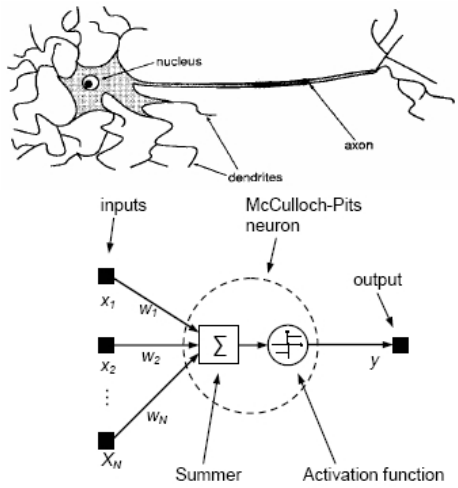

Figura 2.7: Modelo de Neurônio de McCulloch-Pits (17): A resposta de um neurônio aos seus estímulos elétricos pode ser modelada através uma função de ativação o total da soma ponderada destes estímulos. Imagem retirada de http://www.controlglobal.com/articles/2006/221/.

apresentamos o algoritmo do *Perceptron* que encontra um separador linear que classifica as amostras em duas classes. Em seguida, discutimos como a tarefa de classificação por de ser resolvida como um problema de otimização através gradiente descendente. Finalmente, apresentamos a regressão logística que informa o grau de confiabilidade da classificação.

2.2.1
Perceptron

Muitos algoritmos de otimização buscam inspiração em comportamentos da natureza para a construção de sistemas inteligentes. O *Perceptron* de Rosenblatt (14) é um modelo de aprendizado probabilístico análogo aos sistemas biológicos e tem como base o modelo de células nervosas de McCulloch-Pits (17) exemplificado pela Figura 2.7. A seguir, veremos como este modelo nos permite resolver problemas de classificação binária.

2.2.1.1
Raciocínio: Mapeamento de Domínio

Agora, vamos discutir o *raciocínio* de como cada elemento é mapeado. Consideremos elementos bidimensionais dispostos em duas classes conforme ilustrado na Figura 2.8. Nesse exemplo, a classificação das amostras pode ser realizada de maneira simples, atribuindo a cada amostra uma pontuação que é utilizada para dividi-las em dois grupos, pontuações altas e baixas.

Uma maneira simples de se pontuar amostras de domínio é através de

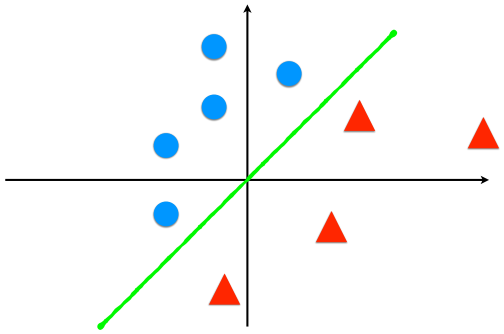

Figura 2.8: Ilustração de um classificador linear binário. Neste exemplo, o classificador é representado uma reta que passa pela origem.

uma combinação linear, associando um peso para cada atributo. Desse modo, podemos utilizar uma reta para repartir o domínio em duas partes, de forma que a avaliação ocorre de maneira bem direta identificando de que lado do espaço amostral um determinado elemento se encontra. Geometricamente, a classificação pode ser interpretada como uma comparação do ângulo entre o vetor normal à reta $\vec{n}$ e o vetor $\vec{x}$ que representa uma dada amostra. Ou ainda, mais formalmente pela Equação 2-1, em que temos $f(x) = \text{sign}(x)$ como função de ativação, associando pontuações altas e baixas respectivamente a valores positivos e negativos.

$$y = \text{sign}\left(\sum_i x_i n_i\right) \tag{2-1}$$

Do ponto de vista estatístico, destacamos que:

(i) A ativação gerada por um elemento está associada à correlação entre seus atributos e o modelo do classificador, adotando como métrica de similaridade o cosseno do ângulo entre suas representações, Equação 2-2.

$$similaridade = \cos(\theta) = \frac{\vec{x} \cdot \vec{n}}{\|\vec{x}\| \, \|\vec{n}\|} \tag{2-2}$$

(ii) A normalização dessas características pela centralização da média[1]em zero, i.e. $x_i - \bar{x_i}$, associando-as respectivamente às variáveis aleatórias X e N, demonstra que a métrica anterior é um caso específico do coeficiente de correlação de Pearson, Equação 2-3.

$$
\begin{aligned}
corr(X,N) &= \frac{cov(X,N)}{\sigma_X \sigma_N} \\
&= \frac{E[(X-\mu_X)(N-\mu_N)]}{\sigma_X \sigma_N} \\
&= \frac{\sum_{i=1}^{n}(x_i-\bar{x})(n_i-\bar{n})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2 \sum_{i=1}^{n}(n_i-\bar{n})^2}}
\end{aligned}
\tag{2-3}
$$

Assim, temos que o *Perceptron* é um neurônio artificial cuja a função de ativação indica se um dado elemento de domínio é similar ou dissimilar ao seu modelo.

2.2.1.2 Generalização: Aquisição de Conhecimento

Vimos que em seu processo de classificação, o *Perceptron* faz uso da correlação entre os atributos de entrada e seu modelo interno. Agora vamos especificar a etapa de aprendizagem que tem como finalidade produzir um modelo que generalize o conhecimento implícito nas amostras do domínio.

De forma abstrata, podemos compreender a aprendizagem como um processo iterativo que alterna entre duas fases: *identificação de erro* e *correção do modelo*.

Logo, podemos ver a regra de aprendizagem do algoritmo do *Perceptron*, Equação 2-4, em que $w = \vec{n}$, como um processo intuitivo que corrige o modelo através do aumento da correlação direta com os elementos da classe de interesse, i.e. $\vec{x}_A \cdot \vec{n} \rightarrow +1$. Analogamente, aumentando a correlação indireta com relação aos elementos da classe oposta, i.e. $\vec{x}_{\bar{A}} \cdot \vec{n} \rightarrow -1$.

$$
w_{t+1} = w_t + (y - \hat{y})x \tag{2-4}
$$

No Algoritmo 1, descrevemos o pseudocódigo do *Perceptron* para problema de classificação binária, onde $y \in \{0, 1\}$ e a função de ativação $sign(z)$ é definida pela Equação 2-5.

$$
f(z) = \text{sign}(z) = \begin{cases} -1 \text{ if } z < 0 \\ +1 \text{ if } z > 0 \end{cases} \tag{2-5}
$$

[1] Além da centralização da média, existem outras normalizações que são comumente utilizadas, como a normalização do intervalo de x_i para $[-1, 1]$ e a normalização de $\vec{x}$, obtendo um vetor de módulo unitário, $\frac{x}{\|x\|}$. Contudo, essas técnicas não são abordadas por este trabalho visto que utilizamos não são adequadas para uma representação de domínio esparsa utilizada pela nossa modelagem.

Algoritmo 1 Perceptron Binário

Input:
$\mathcal{D}$ ▷ Dataset de treino
$EPOCHS$ ▷ Número de épocas
Output:
w ▷ Modelo representado pelo vetor de pesos

1: $\mathbf{w} \leftarrow \mathbf{0}$
2: $t \leftarrow 0$

3: **while** $t < EPOCHS$ **do**
4: **for each** $(x, y) \in \mathcal{D}$ **do**
5: $\hat{y} \leftarrow [\text{sign}(wx) + 1]/2$ ▷ Avaliação
6: $error \leftarrow y - \hat{y}$ ▷ Indentificação de erro
7: $\mathbf{w} \leftarrow \mathbf{w} + [error]\mathbf{x}$ ▷ Correção do modelo
8: **end for**
9: $t \leftarrow t + 1$
10: **end while**

11: **return w**

2.2.1.3 Representação do Modelo

Acima, nós apresentamos o classificador linear através de um exemplo bem simples. A ilustração de dados amostrais da Figura 2.8, tem duas características em particular: o separador linear passa pela origem; e as amostras possuem uma perfeita separabilidade linear. Infelizmente, as aplicações práticas mais interessantes não se comportam dessa forma.

A primeira característica é generalizada com a introdução de um variável que representa o *bias* do domínio. Essa variável corrige a predição e seu valor depende do posicionamento global dos elementos no espaço amostral, logo deve ser aprendido junto ao modelo. No caso da Figura 2.8, essa correção pode ser interpretada como um limiar que ajusta a classificação de um valor entre pontuação alta e baixa. Ou ainda, analiticamente, como um deslocamento da reta no eixo y; e geometricamente, como uma translação das amostras no espaço amostral. Mais adiante, na Seção 2.5.1 discutiremos de forma mais abrangente sobre *bias* do modelo e como isso se relaciona com ruídos presentes na amostragem do dado, em que consideramos o ruído como tendo média zero justamente por ele ser compensado pela variável *bias*.

É comum a representação do *bias* através da adição de uma dimensão extra w_0 no modelo e um atributo constante $x_0 = 1$ nas amostras. Dessa forma, a adição da variável *bias* não desqualifica a linearidade do *Perceptron*, uma vez

que passa a ser descrito por uma função linear homogênea, em que o modelo é representado por um plano, ou um hiperplano **w** em domínios de dimensão superior, Equação 2-6.

$$\sum_i \mathbf{w_i} x_i = 0 \tag{2-6}$$

A segunda particularidade mencionada, a não separabilidade linear, é abordada por extensões dos classificadores linear. Uma delas é a adição de atributos resultantes de uma combinação não-linear dos atributos originais, como x_0^2, $x_0 x_1$, x_1^2, etc. Essa abordagem permite ao classificador aprender os coeficientes de um polinômio para separar as classes. Um outro exemplo é a suavização de margens por meio de variáveis de folga. Existe ainda o uso de "truques" de *kernel*, que mapeiam o produto interno como medida de utilizado na correlação, para um espaço de maior ordem. Estas extensões não serão discutidas pois estão além do escopo deste trabalho.

2.2.2 Gradiente Descendente

Como vimos, o algoritmo do *Perceptron* ajusta os parâmetros do modelo aumentando a sua correlação com as informações de domínio, o que é feito de amostra em amostra.

Encontrar os parâmetros que melhor satisfaçam um modelo ou sistema é uma tarefa difícil. Na maioria dos casos não existe uma solução ideal, portanto a busca passa a ser pelos parâmetros que mais se adequam, i.e. aqueles que produzem o menor erro. Uma abordagem comum para problemas de otimização é fazer uso de soluções analíticas. Por exemplo, modelos de regressão linear no formato $\hat{Y} = WX$, em que X é a matriz do *dataset*, o modelo pode ser estimado através do método de mínimos quadrados, Equação 2-7, tem assim uma solução dada pela Equação 2-8.

$$\hat{W} = arg_{min} \left\| Y - WX \right\|_2 \tag{2-7}$$

$$\hat{W} = (X^T X)^{-1} X^T Y \tag{2-8}$$

Porém, abordagens analíticas nem sempre são viáveis. Parte dos problemas de regressão não-linear nem mesmo possuem uma solução analítica. Em outros casos, $X^T X$ pode ser uma matriz singular, não admitindo inversa. Ademais, para os casos em que o *dataset* é muito grande, as operações de produto e inversão de matrizes requereriam recursos computacionais exorbitantes.

Agora, discutimos uma abordagem iterativa que ajusta os parâmetros do modelo com base em sua função objetivo. Seja uma função objetivo $J(\theta)$ derivável com relação a θ, em que $\theta \in \mathbb{R}^d$. O gradiente descendente—GD—

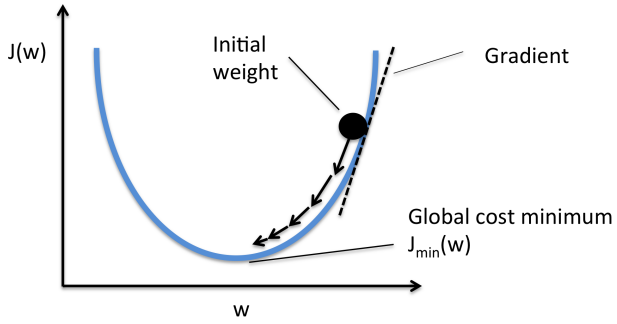

Figura 2.9: Ilustração da solução de um problema de minimização através de graduente descendente. Neste exemplo temos uma função objetivo convexa. Imagem retirada de http://sebastianraschka.com/faq/docs/closed-form-vs-gd.html.

consiste em uma busca iterativa por um θ que minimize a função objetivo $J(\theta)$, (18). Essa função pode ser vista como uma superfície definida no espaço de busca, em que o algoritmo de otimização busca um ponto de mínimo ajustando os parâmetros θ através de passos ao longo dessa superfície na direção oposta à sua inclinação na posição corrente, Figura 2.9.

2.2.2.1 Abordagens

As técnicas de GD possuem três abordagens que variam de acordo com a quantidade de dado utilizado para dar cada passo na otimização.

Gradiente descendente em lote Também chamada de *batch gradient descent*—BGD, a atualização por lote é a abordagem tradicional desse algoritmo, e consiste em utilizar todo o *dataset* para calcular um único passo de atualização. A sua principal vantagem é a garantia de convergência para o mínimo global em superfícies convexas, ou para um mínimo local em superfícies não-convexas. O tratamento global da minimização da função objetivo produz passos de atualização mais "suaves", sendo menos sensíveis às pequenas *flutuações* do conjunto de treino. Por outro lado, essa abordagem pode ser muito devagar ou até intratável para *datasets* que não cabem na memória, além de não contemplar uma atualização *online* dos parâmetros a partir de novas amostras.

Gradiente descendente estocástico Em sua versão estocástica, do inglês *stochastic gradient descent*—SGD, a atualização dos parâmetros é feita para cada amostra do conjunto de treinamento. Essa é uma vantagem em relação à versão em lote em que o BGD realizava cálculos redundantes computando

o gradiente para exemplo similares antes da atualização dos parâmetros. Além de ser uma alternativa bem rápida, essa abordagem possibilita o uso de aprendizado *online*, semelhante ao *Perceptron*. Entretanto, apesar da alta frequência das atualizações, o uso de uma amostra por atualização resulta em passos de atualização mais sensíveis à variância do *dataset*, propagando *flutuações* do conjunto de treino para a função objetivo. Essas *flutuações* fazem com que as atualizações extrapolem o ponto de mínimo, dificultando a convergência para um mínimo exato. Portanto, não neste caso não há garantia de convergência sem um tratamento especial.

Gradiente descendente em mini-lote Esse algoritmo é uma variação do SGD, em que combina o melhor das abordagens em lote e estocástica. As atualizações são computadas utilizando mini-batches, ou mini-lotes, compostos por n amostras. Assim, obtemos tanto uma redução da variância da função objetivo com passos de atualização "suaves", podendo resultar em uma convergência mais estável; como também um processo de minimização mais rápido. Por utilizar um conjunto pequeno de amostras para cada atualização, a abordagem em mini-lotes ainda torna viável o uso de algoritmos sofisticados de optimização que utilizam uma representação matricial, tornando o cálculo do gradiente com relação ao mini-lote mais eficiente.

2.2.2.2 Algoritmos de otimização

Acima, mostramos as variantes do gradiente descendente com relação ao número de exemplos utilizados no cálculo da atualização. Agora, destacamos alguns dos algoritmos de otimização mais utilizados para computar o passo de atualização, (19).

Convencional Em sua forma convencional, o gradiente descendente aplica uma atualização dos parâmetros proporcional ao gradiente da função objetivo $\nabla_\theta J(\theta)$ em relação a θ, Equação 2-9. Nesta equação, η determina o tamanho do passo dado em direção a um ponto de mínimo. Em tarefas de aprendizado, essa constante é referenciada como taxa de aprendizado.

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_{\theta_t} J(\theta_t) \tag{2-9}$$

Momentum No SGD convencional, o gradiente $\nabla_{\theta_t} J(\theta_t)$ pode variar bruscamente já que utilizamos um conjunto de amostras diferentes em cada passo de atualização. Tal comportamento é observado em regiões do espaço de busca

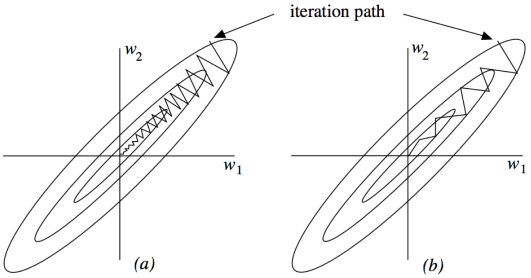

Figura 2.10: Comparação do SGD sem momentum, à esquerda, e com momentum, à direita. Imagem retirada de (19).

em que o declive em uma das dimensões da superfície é bem maior se comparado ao das demais dimensões, conforme ilustrado na Figura 2.10(a). Essas oscilações retardam a busca por um ponto de mínimo uma vez que o passo de atualização do algoritmo clássico do GD é proporcional ao declive da superfície, porém os declives mais ingrimes não necessariamente apontam na direção do ponto de mínimo. O gradiente descendente com *momentum* ameniza esse problema reutilizando valores de gradiente das iterações anteriores. Ao invés de seguir a direção indicada apenas pelo gradiente corrente, esse algoritmo introduz um hiperparâmetro de *momentum* que é uma média ponderada do passo anterior e do gradiente corrente. A regra de atualização é descrita pela Equação 2-10, em que $\mu \in [0, 1]$ é o *momentum*. Dessa forma, há um acumulo de deslocamento nas dimensões com pequenos passos mas que apontam em uma mesma direção, enquanto que os deslocamentos nas dimensões que mudam bruscamente de direção são atenuados, Figura 2.10(b). A utilização de *momentum* assemelha-se a introdução de inércia ou velocidade no algoritmo de busca, reduzindo a oscilação e aumentando sua velocidade de convergência.

$$\begin{aligned} v_{t+1} &= \mu v_t + \eta \cdot \nabla_{\theta_t} J(\theta_t) \\ \theta_{t+1} &= \theta_t - v_{t+1} \end{aligned} \tag{2-10}$$

Nesterov's accelerated gradient Vimos como o acúmulo de *momentum* ao longo das iterações favorece o deslocamento em dimensões com baixo deslocamento se comparado às dimensões que se comportam de maneira brusca. Contudo, existem casos em que o uso de *momentum* ocasiona um comportamento indesejado, aumentando a oscilação ao redor de um ponto de mínimo, ou até mesmo escapando de um ponto de mínimo. Quando as oscilações não são frequentes o suficiente para se cancelarem em atualizações

sucessivas, elas tendem a serem ampliadas pelo acúmulo de *momentum*. O *Nesterov's accelerated gradient*—NAG—é uma variação do *momentum* que aborda esse problema, sendo definida pela Equação 2-11. A diferença da regra de atualização do NAG em relação à regra de atualização do *momentum* está apenas na atualização do vetor de velocidade v. Ao invés de utilizar o gradiente na posição atual, o NAG calcula o gradiente com base em uma atualização parcial $\theta_t + \mu v_t$, essa atualização seria uma aproximação da próxima posição no espaço de busca. Dessa forma, a antecipação da atualização previne a ocorrência de oscilações acentuadas. Assim, o NAG atualiza o vetor v de forma mais responsiva e estável.

$$\begin{aligned} v_{t+1} &= \mu v_t + \eta \cdot \nabla_{\theta_t} J(\theta_t + \mu v_t) \\ \theta_{t+1} &= \theta_t - v_{t+1} \end{aligned} \tag{2-11}$$

2.2.2.3 Considerações

Vimos como gradiente descendente pode é utilizado para resolver problemas de otimização como uma alternativa eficaz para métodos analíticos. Existem alguns fatores que devem ser considerados:

- **Convexidade**: A não ser que a função objetivo defina uma superfície convexa, não há garantia de convergência dos parâmetros para um ponto de ótimo global. Pois, é possível que o modelo fique preso em mínimos locais resultando em soluções sub-ótimas. Assim, é comum o uso de SGD no intuito de impedir uma convergência prematura.
- **Convergência**: É necessário ainda definir um critério de convergência. Uma prática comum é utilizar um pequeno limiar. Assim, se a atualização entre iterações ao longo da função objetivo não foi significante o suficiente, o deslocamento no espaço de busca é tido com desprezado, caracterizando que houver convergência.
- **Desempenho**: A taxa de aprendizado o parâmetro central para o desempenho da otimização por gradiente descendente. A escolha de um valor adequado pode ser difícil. Se escolhido um valor muito baixo, o processo de convergência pode ser muito demorado. Por outro lado, uma taxa muito alta pode impedir a convergência, causando *flutuação* ou até mesmo divergência.
- **Taxa de aprendizado**: Um outro aspecto a ser observado é que por se tratar de um valor escalar, a mesma taxa de aprendizado é aplicado

para todos os parâmetros, o que pode ser inapropriado dependendo da aplicação. Por exemplo, em atributos esparsos em que os atributos possuem frequências diferentes, não é desejável que eles sejam atualizados da mesma forma, mas que atributos menos frequentes possuam uma taxa de aprendizado maior.

2.2.3 Regressão Logística

Vimos como *Perceptron* opera tarefas de classificação correlacionando o modelos com os dados amostrais através de uma combinação linear e uma função de ativação não-linear. Então, apresentamos como problemas de otimização podem ser resolvido através de gradiente descendente, produzindo um modelo minimizando uma função objetivo de modo iterativa. Agora, introduziremos uma abordagem que trata a tarefa de classificação como um problema de otimização.

A regressão logística é um modelo análogo à regressão linear, que infere parâmetros de uma combinação linear, mapeando a distribuição dos elementos do domínio. Assim, a regressão logística pode ser vista como um caso especial da forma geral de um modelo linear $\hat{y} = f(z)$, em que $f(z)$ é uma função de ativação não-linear e z é a pontuação da amostra x dada por $z = wx$. Isso possibilita que o contradomínio seja descrito através de representações categóricas ao invés do conjunto dos números reais $\mathbb{R}$.

Diferentemente do *Perceptron*, a regressão logística faz uso do gradiente descendente para aprender os parâmetros de seu modelo através da minimização de uma função objetivo ou função de perda.

A seguir, detalhamos a função sigmóide, utilizada como a função de ativação da regressão logística. Depois, apresentamos duas funções de perda: o erro quadrático e a entropia cruzada. Por fim, discutimos a escolha da entropia cruzada como função de perda mostrando como esta previne algumas problemas associadas ao uso do erro quadrático.

2.2.3.1 Sigmóide

A sigmóide, Equação 2-12, é uma função em formato de S que normaliza os valores de z para o intervalo $(0, 1)$, onde o parâmetro k que permite o ajuste de seu declive.

$$\sigma(z) = \frac{1}{1 + e^{-kz}} \tag{2-12}$$

Ao contrário da função sign(z) utilizada na ativação do *Perceptron*, a sigmóide é derivável em todos os pontos tornando possível a otimização por gradiente descendente. Na função sign(z), mesmo que fosse desconsiderado o ponto onde a derivada não é definida, $x = 0$, não seria possível utilizá-la como função de ativação em conjunto com o GD, visto que sua derivada é zero no intervalo em que é definida, anulando o passo de atualização do GD.

A derivada da sigmóide é computada de maneira bem direta através da Equação 2-13. O desenvolvimento dessa equação é demonstrado no Apêndice A.1.

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))k \; dz \tag{2-13}$$

2.2.3.2 Erro Quadrático

Uma vez definida uma função de ativação diferenciável, agora resta adotar uma função de perda que mede o erro de classificação do modelo, para então utilizar o método de minimização de erro por gradiente descendente.

O erro quadrático é uma das funções de perda mais utilizadas e descreve o erro através da Equação 2-14. Dessa forma associamos a perda do modelo ao quadrado da distância euclidiana entre y e $\hat{y}$.

$$L = (y - \hat{y})^2 \tag{2-14}$$

Assim, calculando a derivada da função de perda, Equação 2-15, e diferenciando com relação a w pela regra da cadeia, obtemos a regra de atualização do modelo utilizada pelo algoritmo do GD, Equação 2-16.

$$\frac{\partial}{\partial \hat{y}} L = 2(y - \hat{y}) \; \partial \hat{y} \tag{2-15}$$

$$w_{t+1} \; = w_t - \eta \; \underbrace{[y - \sigma(w_{t-1}x)]}_{L'} \; \underbrace{\sigma(w_{t-1}x)[1 - \sigma(w_{t-1}x)]}_{\sigma'} \; x \tag{2-16}$$

2.2.3.3 Entropia Cruzada

O resultado da sigmóide pode ser interpretado estatisticamente como a probabilidade de ocorrência, $p(y|x)$, da categoria y observado o evento x. Dessa forma, associamos o *dataset* a um conjunto de eventos X gerados através de uma distribuição $p(x)$ que relaciona os atributos de x à categoria y. Essa relação, $p(y|x)$, é também chamada de *probabilidade a posteriori*.

Em geral, a *probabilidade a posteriori* informa a chances de um objeto pertencer a uma categoria específica após observar a informação relacionada à essa associação, (21).

Assim, a regressão logística pode ser visto como um modelo probabilístico que aproxima a distribuição $p(x)$ por uma distribuição $q(x)$, maximizando o valor esperado de $q(y|x)$. Essa aproximação pode ser avaliada utilizando a entropia cruzada como métrica de correlação entre as distribuições $p(x)$ e $q(x)$.

A entropia cruzada é uma métrica de teoria de informação que mede o valor esperado E_p da quantidade de informação I_q necessária para representar um evento aleatório a partir de duas distribuições de probabilidades $p(x)$ e $q(x)$ sobre o mesmo conjunto de eventos X, Equação 2-17.

$$H(p,q) = E_p(I_q(X)) = -\sum_x p(x) \ln q(x) \tag{2-17}$$

Na Equação 2-18 temos a função de perda descrita em sua geral para problemas multiclasse.

$$L = -\sum_i y_i \ln \hat{y}_i \tag{2-18}$$

Para problemas de classificação binário, a função de perda é rescrita conforme a Equação 2-19.

$$L = -[y \ln \hat{y} + (1-y) \ln(1-\hat{y})] \tag{2-19}$$

Deste modo, calculando a derivada da função de perda e diferenciando com relação a w pela regra da cadeia, obtemos a regra de atualização do modelo utilizada pelo algoritmo do GD, Equação 2-20, que nos remete à regra de atualização do *Perceptron*, Equação 2-4. A derivada que compõe a regra de atualização é demonstrada no Apêndice A.2.

$$w_{t+1} \ = w_t - \eta \ (\sigma(w_t x) - y) \ x \tag{2-20}$$

2.2.3.4 Considerações

A escolha entre erro quadrático e entropia cruzada como métrica de erro é um questão já bem abordada, (21) e (20). Agora, discutimos sobre as duas métricas apresentadas, comparando interpretabilidade, inconveniências no calculo de erro e aprendizado.

Para isso, adotaremos uma representação vetorial onde y contém uma representação binária da categoria de uma amostra, e $\hat{y}$ é de distribuição de probabilidade inferida pelo modelo que aproxima y. Essa é uma representação utilizada por classificadores multiclasses, em que y_i e $\hat{y}_i$ contém um valor

associado a uma categoria i. Por exemplo, $y = 0$ e $\hat{y} = 0.7$ passam a ser representados por:

$$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \hat{y} = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}$$

Interpretabilidade Adotando a representação introduzida acima, temos que o erro quadrático é uma métrica de fácil compreensão, estando associada à distância euclidiana entre y e $\hat{y}$. Essa métrica assume que ambos tenham um comportamento Gaussiano. Entretanto, isso é violado dado que *targets* binários são utilizados para treinar o classificador. Em contraste, a entropia cruzada tem uma interpretação mais sofisticada, nas distribuições $p(x)$ e $q(x)$ são correlacionadas através da minimização da quantidade de informação $I_p(y)$ e aumento a probabilidade do valor esperado de y.

Medição do Erro A função de perda do erro quadrático produz um *erro absoluto* uma vez que essa métrica acumula o erro com base em cada dimensão dos vetores y e $\hat{y}$, em que $y_i - \hat{y}_i \in (0, 1)$. Na entropia cruzada, entretanto, uma vez que y é um vetor diferente de 0, i.e. $y_i = 1$, apenas na posição referente à real categoria da amostra x, temos uma descrição de *erro relativo*. Sendo assim, o erro é medido pela surpresa ou quantidade de informação. Quanto menor a probabilidade associada à classe real maior o erro, $-\ln \hat{y} \in [0, \infty)$.

Aprendizado Um comportamento esperado no processo de aprendizagem é que a correção do modelo seja proporcional ao erro por ele cometido. Entretanto é esse comportamento nem sempre é mantido pela combinação do erro quadrático com a sigmóide, (20). Suponhamos o caso em que $\hat{y} \approx 0$ quando $y = 1$, em que $wx \to \infty$ à medida em que $\hat{y} \to 0$. Consequentemente, quanto maior a distância entre y e $\hat{y}$, menor o passo de atualização do modelo, Equação 2-16, visto que $\sigma'(wx) \to 0$. Por outro lado, quando utilizado a entropia cruzada a derivada da sigmóide é cancelada conforme demostrado no Apêndice A.2, resultando em uma atualização do modelo proporcional ao erro, Equação 2-19. Esse é exatamente o comportamento esperado para erros muito grandes em que o erro quadrático produziria um passo pequeno de atualização. Logo, por ser mais adequado, a entropia cruzada é a métrica de perda adotada neste trabalho.

2.3

Classificadores Multiclasse

Na seção anterior, mostramos como classificadores lineares são utilizados para resolver problemas de predição binária. Apresentamos o algoritmo do *Perceptron* e Regressão Logística que utiliza a função sigmóide como ativação, realizando o aprendizado por gradiente descendente.

Contudo, a predição binária representa apenas uma pequena parcela dos problemas de classificação. A tarefa de anotação morfossintática, por exemplo, requer uma distinção bem mais refinada entre os elementos do domínio.

Nesta seção, nos voltamos para a solução de problemas de classificação multinomial, ou multiclasse, adotando as técnicas já discutidas como base. Inicialmente, discutimos estratégias que se baseiam na combinação de classificadores binários. Em seguida, apresentamos o algoritmo do perceptron multiclasse.

2.3.1 Estratégias Gerais

Os classificadores binários podem ser combinados para resolver problemas mutliclasses envolvendo K classes através de duas abordagens: *one-vs-one* e *one-vs-all*, (22).

One-vs-one Na abordagem *one-vs-one*, as classes são combinadas duas a duas. Um classificador binário é treinado para cada combinação utilizando apenas as amostras pertencentes àquelas classes. Essa abordagem aprende um total de $K(K-1)/2$ classificadores. Na resolução da tarefa, cada amostra é avaliada por todos os classificadores, em que cada classificador escolherá uma dentre duas classes. Cada classificação resulta em um voto que será creditado a classe predita. Assim, ao fim do processo, a classe que recebeu o maior número de votos é escolhida como resultado da predição multiclasse.

One-vs-all Essa abordagem consiste em treinar K classificadores binários, $f_k(x)$. Cada classificador utiliza todas as amostras do conjunto de treino, distinguindo uma classe específica de sua contra-classe, resultado da união das demais categorias. Para utilizar dessa abordagem, é necessário que o método de classificação escolhido, $f(x)$, informe um valor associado à confiabilidade da predição. Dessa forma, a predição do problema multiclasse é dado pela Equação 2-21.

$$\hat{y} = arg \max_{k \in 1...K} f_k(x) \tag{2-21}$$

Considerações Em ambas as estratégias apresentadas acima provêm uma solução para problemas multiclasses. Porém temos algumas questões a ser consideradas:

(i) Ambiguidade: Ambas as abordagens podem apresentar ambiguidade na categorização do espaço de amostra. A estratégia *one-vs-one* pode apresentar duas classes com o mesmo número de votos; enquanto que a estratégia *one-vs-all* pode ter uma escala de confiança que varie ao longo dos classificadores binários.

(ii) Desbalanceamento: É importante fazer uso de um *dataset* balanceado para se obter um modelo que gere uma boa generalização da relação entre as classes. Assim, no caso da abordagem *one-vs-all*, seria necessário um tratamento especial para manter o balanceando das classes utilizadas por cada um de seus K classificadores, visto que os conjuntos de classes positivas são em geral bem menor que os conjuntos resultante das negativas.

(iii) Desempenho: O número de classificadores cresce junto ao número de classes. Durante a etapa de treinamento, em ambas as abordagens, K destes classificadores compartilham as mesmas amostras. Essa redundância pode aumentar consideravelmente o tempo de treinamento do modelo.

2.3.2 Perceptron Multiclasse

Vimos como classificadores binários podem ser combinados para resolver problemas mutliclasses através abordagens que fazem uma composição de classificadores binários: *one-vs-one* e *one-vs-all*. Agora, apresentamos uma generalização do *Perceptron* para problemas multiclasse.

O *Perceptron* multiclasse opera de forma similar à sua versão binária, porém generalizando seu modelo para uma matriz de pesos w, cujo número de linhas é igual ao número de classes K. Cada linha w_k está associada ao modelo da classe k.

Essa descrição nos remente a abordagem *one-vs-all*, discutida anteriormente. Porém, os modelos do *Perceptron* multiclasse são interdependentes, uma vez que são aprendidos e avaliados simultaneamente. Isso torna o processo de classificação robusto com relação à ambiguidade que sujeita a abordagem *one-vs-all*. Não havendo uma variação de escala do valor produzido por cada modelo, pois os modelos são treinados de forma conjunta.

A predição do Perceptron Multiclasse pode ser feito conforme a Equação 2-22, em que é comum o uso de uma função de ativação linear, $f(z) = z$.

$$\hat{y} = arg \max_{k \in 1...K} f(w_k x) \tag{2-22}$$

Essa é uma técnica que se popularizou recentemente, especificamente no campo de NLP na resolução das tarefas de anotação morfossintática e análise sintática, (16). O Algoritmo 2 descreve seu processo de treinamento. Semelhantemente ao Perceptron binário, o Perceptron multiclasse atualiza seus modelos internos através do aumentando da correlação: direta, entre os atributos de entrada com modelo referente à classe correta; e indireta, entre os atributos de entrada e o modelo referente à classe predita.

Algoritmo 2 Perceptron Multiclasse

Input:
$\mathcal{D}$ ▷ Dataset de treino
$EPOCHS$ ▷ Número de épocas
Output:
$\mathbf{w}$ ▷ Modelo representa pela matriz de pesos

$\mathbf{w} \leftarrow \mathbf{0}$
$t \leftarrow 0$

while $t < EPOCHS$ **do**
  for each $(x, y) \in \mathcal{D}$ **do**
    $\hat{y} \leftarrow arg \max_{k \in 1...K} \{w_k x\}$ ▷ Avaliação
    $\mathbf{w}_y \leftarrow \mathbf{w}_y + \mathbf{x}$ ▷ Correção do modelo
    $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - \mathbf{x}$ ▷ As correções se cancelam quando $y = \hat{y}$
  end for
  $t \leftarrow t + 1$
end while

return w

2.3.3 Softmax

O *Softmax* é uma função não linear que utilizada para normalizar as saídas das ativações dos neurônios para o intervalo $(0, 1)$, sendo definido pela Equação 2-23. O *Softmax* funciona como uma generalização da função sigmóide para problemas multiclasses, sendo da mesma forma utilizado em conjunto com entropia cruzada como função de perda. Semelhantemente à sigmóide, o *Softmax* é combinado com a técnica de minimização por gradiente descendente para o aprendizado de um classificador multiclasse

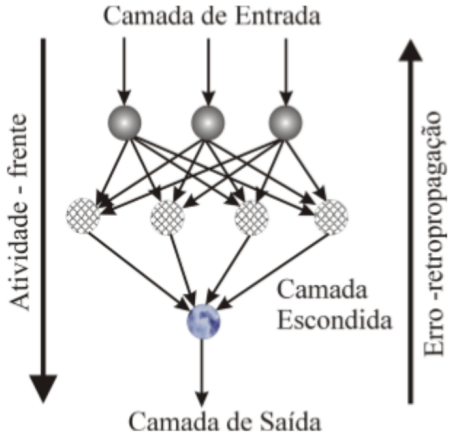

Figura 2.11: Ilustração do modelo de uma rede MLP 3-4-1 Imagem retirada de ftp://ftp.inf.puc-rio.br/pub/docs/theses/ 04_PhD_silva.pdf.

$$\sigma z_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{para} j = 1...K \tag{2-23}$$

2.4 Redes Neurais Artificiais

Nesta seção introduzimos os conceitos básicos de redes neurais artificiais em que abordamos o *Multilayer Perceptron*.

Multilayer Perceptron—MLP—é uma arquitetura de redes neurais muito utilizada. Nela, os neurônios são dispostos em duas ou mais camadas de processamento, de forma que sempre haverá uma camada de entrada, uma de saída e zero ou mais camadas intermediárias escondidas. A Figura 2.11 exemplifica uma rede neural com uma camada escondida. Esta arquitetura é geralmente referida como 3-4-1, ou seja, 3 neurônios de entrada, 4 neurônios escondidos e 1 neurônio de saída. Para generalizar, podemos dizer que uma rede com p entradas, h_1 neurônios na primeira camada escondida, h_2 na segunda camada escondida e q neurônios na camada de saída é descrita por p-h_1-h_2-q.

Para "aprender", a rede neural do MLP passa por um processo interativo de ajustes de pesos nas conexões e unidades das camadas da rede. O ajuste começa após a rede ser estimulada pela informação de entrada e ocorre de forma minimizar uma função de erro, e é comandado por algum algoritmo de treinamento. O algoritmo de aprendizado utilizado pelo MLP é a retropropagação, do inglês *backpropagation*.

No fundo, o MLP pode ser visto como uma versão mais robusta do *Softmax*, em que temos diversas camadas de abstração sobrepostas. Quanto mais interna for uma camada, mais complexas são os atributos elaborados

por elas. Semelhante ao *Softmax*, o MLP tem uma fase de ativação, também chamada de *feed-forward* e a fase de correção *feed-backward*, que é o já mencionado, *backpropagation*, que corrige os pesos internos da rede com base no gradiente descendente.

2.5 Sobreajuste

Redes neurais profundas, com um grande número de parâmetros, são modelos de aprendizado de máquina muito poderosos. Elas conseguem aprender relações complicadas entre suas entradas e saídas. No entanto, *overfitting* é um sério problema em tais redes. Quando a quantidade de dados treinamento é limitada, essas relações complicadas podem ser geradas como consequência de ruídos de amostragem. Essas relações, então, existirão no conjunto de treino, mas não no conjunto de teste, o que ocasiona o *overfitting*.

Muitos métodos têm sido desenvolvidos para reduzir o *overfitting*. Estes incluem parar o treinamento assim que o desempenho em um conjunto de validação começa a ficar pior e introdução de penalizações nos pesos, tais como nas regularizações de modelo L1 e L2.

A seguintes subseções explicarão o que é o balanceamento entre *bias* e variação e como ele é importante no controle do *overfitting*, e também os métodos de regularização de modelo, usados para evitar o overfitting, que foram utilizados nessa dissertação.

2.5.1 Balanceamento entre Bias e Variância

Para entender o que é um balanceamento entre *bias* e variância, é preciso entender primeiro o que são as fontes de erro no cálculo da predição. São três fontes de erro: ruído, *bias* e variância.

Primeiramente, é natural um *dataset* ser contaminado com ruídos. Existe um relacionamento entre uma amostra x_i e sua categoria y_i. Assim, existe uma função f que mapeia esse relacionamento. Sabe-se que existem outros fatores que influenciam y_i além de x_i que não são capturados na amostragem do domínio. Dessa forma, podem existir fatores informativos não são capturados pela função f.

Esses fatores são tratados como ruídos inerentes dos dados ε_i e não se há controle sobre eles, de forma que eles não podem ser preditos no modelo. Dessa forma podemos dizer que os ruídos também são chamados de erros irredutíveis porque não a nada que se possa fazer para reduzi-los. Por outro lado, é possível controlar os dois outros tipos de erro: *bias* e variância.

Bias é uma avaliação de quão bem o modelo g se aproxima do relacionamento real f entre x e y. Assim o *bias* pode ser descrito como um fator que indica se o modelo é flexível o suficiente para capturar o comportamento real. Geralmente, modelos muito simplistas tendem a não ser flexível o suficiente para se aproximar do relacionamento real e consequentemente tem um alto *bias*. Um alto *bias* por sua vez ocasiona mais erros na predição.

A variância representa o quão diferente são os modelos gerados para diferentes amostragens do domínio. Quando um modelo é bem complexo, capaz de se adaptar bem aos dados de entrada, percebe-se que, para cada nova amostragem do domínio, ele gera parametrizações completamente distintas. Se calcularmos a média de todas essas parametrizações do modelo, percebe-se que a variância entre elas é muito alta. No entanto, essa parametrização média tende a se aproximar muito bem do relacionamento real f. Modelos simples, por outro lado, tendem a ter variância baixa. Modelos de alta variância são indicativos fortes de *overfitting*, ou seja, comportam-se bem na etapa de treinamento, mas erram muito no teste.

Encontrar um balanceamento ideal entre *bias* e variância significa encontrar o ponto ideal onde o modelo tenha *bias* alto o suficiente para se aproximar do relacionamento real f, mas com variância baixa o suficiente para evitar *overfitting* nos dados de treino.

2.5.2 Dropout

O *Dropout* é uma regularização aplicada a redes neurais grandes, onde cada unidade de uma camada se liga a todas as unidades da camada seguinte. Essa técnica simples mostrou-se eficiente para reduzir o *overfitting* em redes neurais, (35).

Uma maneira de regularizar um modelo e prevenir o *overfitting* seria calcular a média das previsões de todas as configurações possíveis de parâmetros, pesando cada configuração pela sua probabilidade posterior com base nos dados de treinamento. No entanto, isso é geralmente inviável. Com grandes redes neurais, calcular a média dos resultados de muitas redes treinadas separadamente é muito caro porque seria necessária uma quantidade muito grande de computação.

Para que uma combinação de modelos seja efetiva, os modelos individuais devem ser diferentes um do outro. Eles devem ter arquiteturas diferentes e/ou serem treinados com conjuntos de dados diferentes. Treinar muitas arquiteturas diferentes é difícil porque encontrar parâmetros ideais para cada arquitetura é uma tarefa difícil e treinar cada rede grande requer muita computação. Além

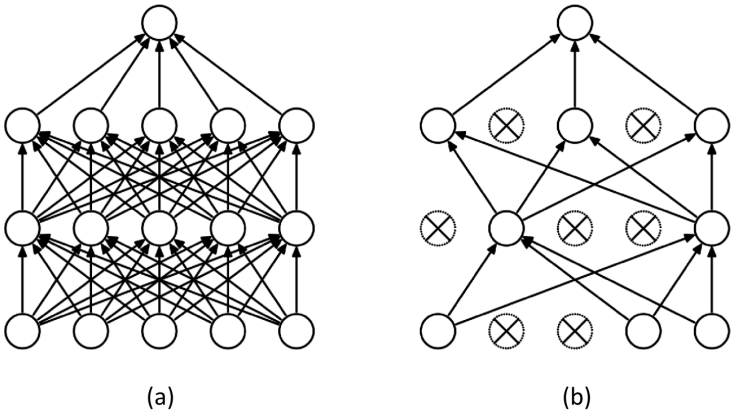

Figura 2.12: Ilustração do uso de *dropout* em uma rede neural. (a) representa uma rede neural comum com duas camadas intermediárias e (b) representa uma rede "afinada" resultante do dropout aplicado em (a). Imagem retirada de (35).

disso, grandes redes normalmente exigem grandes quantidades de dados de treinamento. Como haverá várias redes, será necessário ainda mais dados e pode não ser possível dividir esses dados em subconjuntos suficientemente grandes para cada uma dessas redes. Ainda assim, mesmo que seja possível treinar muitas redes grandes diferentes, usar todas no teste é inviável em aplicações onde é importante uma resposta rápida.

O *Dropout* é uma técnica que aborda estes problemas. Ele evita o *overfitting* e fornece uma maneira de, aproximadamente, combinar muitas arquiteturas de redes neurais diferentes de forma eficiente. A ideia é remover temporariamente unidades escondidas e visíveis em uma rede neural, juntamente com todas as suas ligações de entrada e saída. Esse processo é representado na Figura 2.12. A escolha de quais as unidades serão removidas é aleatória. Uma forma simples seria manter cada unidade com uma probabilidade fixa p independente de outras unidades.

2.5.3 DropConnect

O *DropConnect*, (36), assim como o *Dropout*, introduz uma esparsidade dinâmica dentro do modelo. A diferença é que cada conexão, em vez de cada unidade, pode ser descartada. A escolha de quais as conexões serão removidas é aleatória, com probabilidade p. Essas conexões são removidas temporariamente da rede neural em cada caso de treino.

Para cada caso de treino, uma nova rede com conexões removidas aleato-

riamente é criada e treinada. Após a aplicação do *DropConnect*, cada camada completamente conectada torna-se uma camada esparsamente conectada cujas as conexões são escolhidas ao acaso durante a fase de treinamento.

O *DropConnect* é pode ser visto como uma generalização da *Dropout*. No *Dropout*, quando uma unidade é removida, as suas conexões são removidas como consequência. De forma similar, no *DropConnect*, se todas as conexões de uma unidade são removidas, consequentemente essa unidade pode ser considerada removida também. No entanto, a remoção de uma conexão não implica necessariamente que sua unidade será removida, pois pode haver outras conexões ligadas a ela. Dessa forma, o *DropConnect* pode ser considerado mais flexível, capaz de gerar uma maior possibilidade de modelos em cada caso de treino do que o *Dropout*.

3
Descritores Morfológicos

No capítulo anterior, examinamos como esses sistemas utilizam uma representação distribuída baseada em *word-embedding* para capturar as informações sintática e semântica das palavras. Com a finalidade de suprir a ausência de aspectos morfológicos proveniente da descrição a nível de palavras, duas abordagens são adotadas por esses sistemas: a adição de atributos elaborados manualmente e a adição de *embeddings* a nível de caractere.

Neste capítulo, introduzimos os descritores de domínio adotados por nosso sistema de anotação morfossintática. Utilizamos *n-grams* para obter uma representação local da palavra que contenha suas informações morfológicas. Esses descritores são gerados automaticamente e se baseiam nos caracteres que compõem o sistema de escrita utilizado. Propomos a utilização de *n-grams* para descrever o contexto morfológico das palavras. A partir dessa representação, sistemas de NLP podem capturar informações morfossintáticas para operar análise de texto.

Na Seção 3.1, mostramos como o uso de *n-grams* tem sido abordado em tarefas de processamento de texto. Na Seção 3.2, detalhamos como nossa abordagem descreve cada palavra a partir com base em *n-grams*. Na Seção 3.3, discutimos sobre a representação proposta, destacando seus aspectos importantes.

3.1
Sistemas Baseados em N-grams

Nos primeiros anos de processamento de texto, uma das tarefas iniciais a ser exploradas foi a substituição de caracteres ilegíveis em documentos digitalizados, em que alguns desses sistemas baseiam-se em *n-grams*.

Carlson (23) propõe um sistema baseado na probabilidade de sequências de caracteres. Em seu sistema, é adotado uma representação de *trigrams* que resulta em uma relação razoável entre boa taxa de correção e tempo de processamento hábil. Carlson indica que, em alguns casos, sequências de tamanho quatro podem ser desejadas para obter-se melhores resultados. Contudo, longas sequências devem ser evitadas, visto que o número de combinações cresce exponencialmente, aumentando drasticamente o tempo de processamento. Carlson

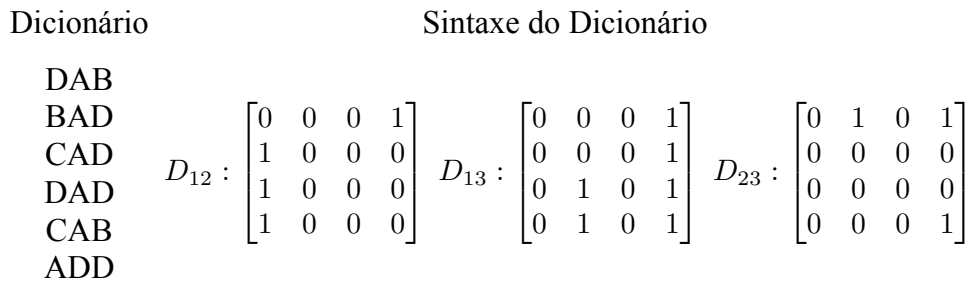

Figura 3.1: Ilustração de *binary drigrams*. Neste exemplo, temos um dicionário de palavras de tamanho três, e um alfabeto de tamanho quatro. No total, são geradas três matrizes de transições. Imagem adaptada de (24)

destaca ainda que o uso da probabilidade da ocorrência de caracteres individuais ou até de sequências de tamanho dois, *bigrams*, não produzem uma boa descrição, gerando substituições ruins.

Em uma tarefa similar, Riseman (24) faz uso de *bigrams* para processar um dicionário, vocabulário, formado a partir das palavras contidas em um documento digitalizado. Riseman restringe sua tarefa em produzir um conjunto de caracteres alternativos para cada caractere da palavra de entrada. Dessa forma, erros ortográficos na palavra digitalizada poderiam ser solucionados encontrando a sequência de transições de caracteres mais provável, analisando apenas um conjunto reduzido de estados.

Riseman demonstra sua representação através de *positional binary digrams*, em que a probabilidade de transição é truncada para um valor binário. Na Figura 3.1, temos uma ilustração da obtenção dessas representações com base em um dicionário de palavras de tamanho três. Em que o pode relacionar o caracteres através de três transições de posição. A primeira matriz $D_{1,2}$, por exemplo, descreve o número de ocorrências em que houve uma transição de caracteres entre as posições 1 e 2 nas palavras do vocabulário.

Generalizando, a Figura 3.1 é uma representação global que descreve o contexto sintático de um dicionário de palavras D de tamanho n através de $n(n-1)/2$ matrizes $D_{ij}(k,l)$ que mapeiam a possibilidade de transição entre dois caracteres (k,l) em função de suas respectivas posições i e j. Sinais de acentuação e pontuação são desconsiderados, produzindo *digrams* de tamanho 26×26, referente apenas às 26 letras do alfabeto.

Posteriormente, Riseman (25) estende o conceito do *positional binary*

digrams para *n-grams*, apresentando um único sistema de detecção e correção de erros ortográficos.

Além da detecção de erros e correção ortográfica, representações baseadas em sequência de caracteres estão presentes em sistemas de NLP de mais alto, como *Machine Translation* (27). No campo de *Document Retrieval*, por exemplo, *n-grams* são utilizados para representar atributos globais de um documento. Esse é o caso de sistemas de *Text Categorization* (26), *Authorship Identification* (28), *Genre Identification* (29) e *Anti-Spam Filtering* (30). Nesses sistemas, a representação de documento tem com base o número de ocorrências de *n-grams* de caracteres em todo o texto.

Portanto, esses trabalhos são fortes evidencias de que o uso de *n-grams* produz excelente descritores de aspectos morfológicos, sintáticos, de conteúdo e até de estilo literário.

Na tarefa de anotação morfossintática, *n-grams* são também utilizados. Contudo, observamos seu uso apenas a nível de palavras para captura de contexto. Brants (31) introduz o *Trigram'n'Tags—TnT*, uma variante do *HMM*, que computa a probabilidade de estados utilizando uma interpolação linear de *unigrams*, *bigrams* e *trigrams* de palavras.

3.2 Representação da Palavra

Na seção anterior, exemplificamos como *n-grams* tem sido explorados na representação de palavras, contexto e documentos. Os sistemas mencionados demonstram o uso da frenquência dos *n-grams* a nível de caracteres, seja através de um valor numérico associado à quantidade de ocorrências, quanto um valor truncado que informa apenas se houve ou não alguma ocorrência.

Nesta seção, detalhamos como descrevemos as palavras através de suas características morfológicas. Nossa representação pode ser comparada à utilizada por Riseman (24), em que desconsideramos a posição da transição dos caracteres nas palavra. Ou ainda, pode ser relacionada a uma versão binarizada das representações utilizadas em sistemas de categorização de texto, (26, 28, 29, 30). Porém, ao invés de estar associada à descrição global a nível de documento ou de sentença, fazemos descrições locais por palavra.

Definimos nossa representação de palavra como o conjunto formado pelas sequências de caracteres que a compõem. Neste contexto, nos referimos a "palavra" como sendo qualquer *token*, seja de pontuação ou uma palavra propriamente dita, com pelo menos um caractere. Assim, a palavra é descrita realizando os seguintes passos:

(i) Deslocamos uma janela de *n-gram* pela palavra, um caractere por vez, convertendo cada *n-gram* para um índice único.

(ii) Descrevemos a palavra utilizando a ocorrência de cada índice, atribuindo o valor 1 para os índices dos *n-grams* presentes na palavra, e 0 para os índices ausentes.

(iii) Incorporamos informações de contexto adicionando as representações de palavras vizinhas.

3.2.1 Mapa de Índices

Nossa representação requer a construção de uma tabela que enumera os *n-grams*, associando-os com índices únicos. No Algoritmo 3, temos uma descrição detalhada desse processo.

A tabela $\mathcal{M}$ é calculada a partir do vocabulário das palavras do conjunto de treinamento V^{wrd}, linha 4. Dessa forma, os *n-grams* de caracteres que estão presentes apenas no conjunto de teste são ignorados. Para cada palavra do vocabulário, consideramos uma representação de lista de caracteres w_0. Então, na linha 5 obtemos uma lista w_p formada pelos caracteres de w_0 envoltos por $n-1$ caracteres de *padding*. Em seguida, na linha 7 analisamos todos os *n-grams* contidos na palavra através de uma janela deslizante de n caracteres. Por fim, na linha 9 utilizamos o tamanho corrente do mapa $|\mathcal{M}|$ como índice único ao adicionar um novo *n-gram*.

3.2.2 Palavra Morfológica

Uma vez computada a tabela $\mathcal{M}$ que mapeia *n-grams* para índices, então podemos descrever palavras através de suas características morfológicas. Introduzimos a palavra morfológica, i.e. a representação da palavras que mapeia seus aspectos morfológicos, representada por um vetor de ocorrência de *n-grams*, Figura 3.2. Como resultado, obtemos uma representação esparsa para cada palavra.

Esse procedimento é formalizado no Algoritmo 4. Nesse algoritmo, cada palavra é representada por um vetor esparso inicializado com valores nulos, linha 7. Os *n-grams* de cada palavra são identificados, linha 10, seus índices consultados, linha 12, e então ativos no vetor de ocorrências w, linha 13.

Algoritmo 3 Construção da tabela de índices

Input:
V^{wrd} ▷ Vocabulário das palavras do conjunto de treino
n ▷ Tamanho do *n-gram*
Output:
$\mathcal{M}$ ▷ Mapa de *n-grams* para índices

$p \leftarrow$ ‘ ’ ▷ Caractere de *padding*
$n_p \leftarrow n - 1$ ▷ Tamanho do *padding*
$\mathcal{M} \leftarrow \{\}$ ▷ Tabela vazia, $|\mathcal{M}| = 0$

for each $w_0 \in V^{wrd}$ **do**
    $w_p \leftarrow (p^{(n_p)}, w_0, p^{(n_p)})$ ▷ Adição de *padding*

    for $i \leftarrow 1, |w_p|$ **do**
        $ngram \leftarrow w_p[i : i + n]$ ▷ Janela do *n-gram*

        if $ngram \notin \mathcal{M}$ **then**
            $\mathcal{M}[ngram] \leftarrow |\mathcal{M}|$ ▷ Índice único
        end if

    end for

end for

return $\mathcal{M}$

$\mathcal{M}_3$ \ w_p	_	_	C	a	s	a	_	_
0	_	_	C					
1		_	C	a				
2			C	a	s			
3				a	s	a		
4					s	a	_	
5						a	_	_
…								

w_0	Casa							
índices	0	1	2	3	4	5	…	$\lvert\mathcal{M}_3\rvert - 1$
w	1	1	1	1	1	1	0	0

Figura 3.2: Ilustração da representação w da palavra através da ocorrência dos *trigrams* presentes na palavra original w_0.

Algoritmo 4 Representação da palavra morfológica

Input:
$\mathcal{D}_0$ ▷ Sentenças de entrada
$\mathcal{M}$ ▷ Mapa de índices de *n-grams*
n ▷ Tamanho do *n-gram*
Output:
$\mathcal{D}$ ▷ Lista de sentenças de representações esparsas

$\mathcal{D} \leftarrow ()$ ▷ Inicializado como lista vazia
$p \leftarrow$ ‘ ’ ▷ Caractere de *padding*
$n_p \leftarrow n - 1$ ▷ Tamanho do *padding*

for each $\mathcal{S}_0 \in \mathcal{D}_0$ **do**
    $\mathcal{S} \leftarrow ()$ ▷ Sentença de representações esparsas

    for each $w_0 \in \mathcal{S}_0$ **do**
        $w \leftarrow 0$ ▷ Vetor de ocorrências
        $w_p \leftarrow (p^{(n_p)}, w_0, p^{(n_p)})$ ▷ Adição de *padding*

        for $i \leftarrow 1, |w_p|$ **do**
            $ngram \leftarrow w_p[i : i + n]$ ▷ Janela do *n-gram*
            if $ngram \in \mathcal{M}$ **then**
                $j \leftarrow \mathcal{M}[ngram]$
                $w[j] \leftarrow 1$ ▷ Ocorrência do *n-gram*
            end if
        end for

        $\mathcal{S} \leftarrow (\mathcal{S}, w)$ ▷ Concatena palavra
    end for

    $\mathcal{D} \leftarrow (\mathcal{D}, \mathcal{S})$ ▷ Concatena sentença
end for

return $\mathcal{D}$

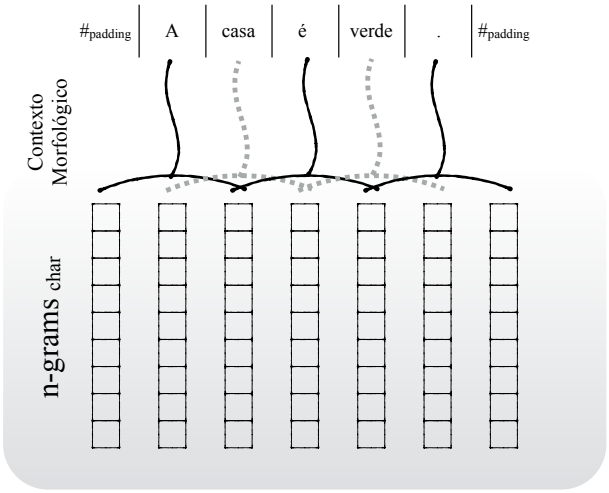

Figura 3.3: Ilustração da abordagem convolucional para a extração de informações de contexto a partir de uma janela de palavras de tamanho três.

3.2.3 Contexto Morfológico

Além de aspectos morfológicos de cada palavra, outra informação que se faz presente nos sistemas de anotação morfossintática é a representação de seu contexto. Atributos de contexto são informações fundamentais pois ajudam a mapear a correlação entre palavras. No caso de classe gramaticais, por exemplo, artigos `ART`, em geral, antecedem adjetivos, nomes ou pronomes.

Incorporamos informações de contexto concatenando as representações das palavras dentro de uma janela de vizinhança de tamanho $2n+1$ em torno de um elemento central a ser descrito, Figura 3.3. Para isso, é necessário a adição de n *tokens* de *padding* no inicio e fim da sentença. Em nosso caso utilizamos o caractere '#' como *padding*.

Essa representação de contexto equivale às abordagens convolucionais de Dos Santos (12) e Fonseca (11), que se baseiam em Collobert (9).

3.3 Considerações

Neste capítulo, propomos uma representação de palavras baseada em descritores morfológicos, associando-a com representações similares utilizadas em outras tarefas de processamento de texto.

A representação de domínio proposta faz uso de *n-grams*, produzindo uma representação de palavra com alto grau de esparsidade. Dessa forma, temos $|w_0| + n - 1$ posições ativas no vetor de ocorrências w de cada palavra w_0.

Tabela 3.1: Número de combinações de *n-grams* considerando apenas letras do alfabeto

N	Caracteres	Combinações
1	26	26
2	26	676
3	26	17.576
4	26	456.976

Em sistemas como o de Riseman (24), temos um tamanho fixo da representação de palavra, que cresce exponencialmente em relação ao tamanho pré-estabelecido da sequência e do conjunto de caracteres utilizados, Tabela 3.1. Nesse sistema, supondo uma representação baseadas em *trigrams* que leve em conta apenas as 26 letras do alfabeto, utilizando uma vizinhança de 3 palavras para a representação de contexto, e considerando uma média de 5 caracteres por palavra, teríamos em média apenas 7 índices ativos de um total de 52.728 (3×26^3). Assim obtendo uma representação com taxa de esparsidade em torno de $99,96\%$.

A esparsidade é uma característica presente nas representações de texto, seja em uma representação local utilizada por sistemas do tipo *bag-of-words* como (10); ou até nas representações distribuídas aprendidas através de *word-embedding* a partir da representação esparsa. Esse é um problema implícito das tarefas de NLP, pois o domínio é inicialmente representado de forma categórica.

Para reduzir a esparsidade e simplificar o modelo, é comum que sistemas não diferenciem letras maiúsculas e minúsculas, ignorando em alguns casos até a presença de dígitos (26). Convencionalmente, em sistemas de anotação morfossintática, todas as palavras são passadas para minúsculas, como também todo algorismo é substituído por um mesmo dígito, como é o caso dos trabalhos de Collobert (9), Dos Santos (12) e Fonseca (11). Essa prática resulta na perda de características morfológicas como a capitalização, que útil importante na identificação nomes próprios, por exemplo, fazendo-se necessário *embeddings*.

Diferentemente dos sistemas convencionais, nossa descrição de domínio baseia-se inteiramente na caixa original das palavras. Dos tratamentos mencionados acima, adotamos apenas a substituição de algorismos por um mesmo dígito. Se considerássemos apenas os 53 caracteres, (`A-Z`, `a-z`) e o dígito `0`, desprezando sinais de pontuação e acentuação, teríamos uma combinação de 148.877 *trigrams*.

A quantidade de combinações real é bem maior, pois ao longo de um texto ainda encontramos caracteres como sinais de pontuação e acentuações. Contudo, como utilizamos de "*hashing*" para mapear os *n-grams* que de fato ocorrem. É esperado uma quantidade bem menor de sequências que a serem

representadas. Assim, o tamanho de nossa representação depende do tamanho do conjunto dos *n-grams* encontrados no conjunto de treinamento.

Por fim, a representação proposta baseia-se inteiramente na morfologia da palavra, generalizando os atributos de prefixos e sufixos que antes eram manualmente especificados por sistemas baseados em regras. Nesses sistemas, tais atributos eram associados associados associados à *embeddings* adicionais, em uma forma semelhante aos *word-embeddings*. Um ponto forte de nossa representação, ao contrario das representações que utilizam descritores a nível de palavra, é que nossos descritores fornecem informações valiosas sobre aspectos morfológicos de palavras que não participaram do treinamento, OOV, de maneira bem direta.

Os descritores de contexto morfológico apresentados neste capítulo são uma contribuição original deste trabalho.

4
Regularização de Domínio

No capítulo anterior, introduzimos uma descrição de palavras utilizando sequências de caracteres, *n-grams*. A descrição proposta por este trabalho é uma alternativa às representações baseadas em vocabulário, sendo mais adequada para descrever palavras não vistas por utilizar aspectos morfológicos da linguagem. É muito improvável a ocorrência de uma palavra no conjunto de testes que não possua pelo menos um de seus *n-grams* presentes no conjunto de treinamento. Entretanto, os descritores adotados produzem uma representação com alto grau de esparsidade.

Nesse capítulo, abordamos a redução de esparsidade através da regularização de domínio. Apresentamos técnicas de seleção de atributos para identificar os *n-grams* mais significativos, reduzindo assim a esparsidade da nossa representação. Uma técnica usual para seleção de atributos é o uso do algoritmo do *Perceptron* Esparso. Inspirados por essa abordagem, propomos uma variante dessa técnica. Diferentemente do *Perceptron* Esparso, utilizamos gradiente descendente para identificar os *n-grams* mais significativos para uma realizar uma dada tarefa. Como resultado da regularização de domínio, podemos obter modelos mais compactos e um menor esforço computacional na etapa de aprendizagem. Além disso, a regularização proposta pode ser incorporada pela arquitetura de redes neurais como uma de suas camadas.

Na Seção 4.1, definimos os seguintes termos: regularização compartilhada, regularização distribuída e regularização *online*. Na Seção 4.2, detalhamos o algoritmo do *Perceptron* Esparso, convencionalmente utilizado para seleção de atributos em representações esparsas. Na Seção 4.3, propomos um modelo de regularização que baseia-se no aprendizado do *Softmax*. Mostramos duas variações deste modelo para operar uma regularização compartilhada, de forma similar ao *Perceptron*. Porém, essas variações têm seus algoritmos de aprendizagem baseados em gradiente descendente. Na Seção 4.4, generalizamos a regularização de domínio para uma forma distribuída. Assim, a regularização opera a seleção de atributo de forma individual, i.e. para cada neurônio da camada de regularização. Na Seção 4.5, discutimos o modelo de regularização proposto, apresentando algumas vantagens que este apresenta, como a possibilidade de regularização *online*.

4.1 Definições

Abordamos neste capítulo a regularização de domínio para a redução de esparsidade tendo em vista o uso conjunto com redes neurais. Assim, definimos os seguintes termos: regularização compartilhada, regularização distribuída e regularização online.

Regularização Compartilhada Adotamos esse termo para referir-nos à regularização de domínio que seleciona os atributos de forma global. O mesmo subconjunto de atributos selecionado é utilizado por todos os neurônios do classificador, como o *Perceptron* Multiclasse ou *Softmax*.

Regularização Distribuída Adotamos esse termo para referir-nos à regularização de domínio que seleciona os atributos de forma local. Ao contrário da regularização compartilhada, esta regularização consiste em distribuir a escolha dos atributos de domínio para cada neurônio do classificador. Buscamos obter uma seleção de atributos mais refinada, de forma que cada neurônio seja capaz de utilizar apenas as entradas que mais o sensibilizam.

Regularização Online Em geral, a seleção de atributos é assíncrona ou *offline*, tratada como uma etapa à parte do treinamento do modelo final do classificador. Assim temos dois classificadores, em que o classificador extra faz um pré-processamento para a seleção de atributos. Joachims (10) utiliza *information gain* para a redução de esparsidade da representação de palavras baseada em vocabulário. Somente após a seleção de atributos, a categorização de texto é operada com o *Support Vector Machine*—SVM. Motta (8) também faz uso de seleção de atributos em seu *framework* IFIS. Após fase de indução de atributos, os atributos são selecionados através do *Perceptron* Esparso. Em seguida, os atributos selecionados são utilizados tanto para realimentar a fase de indução, quando para treinar o classificador SVM. Suponhamos um caso hipotético de regularização compartilhada em que os atributos selecionados são utilizados para alimentar uma rede neural. A não ser que a primeira camada da rede neural possua a mesma quantidade de neurônios do classificador utilizado na seleção de atributos, a regularização distribuída teria um efeito semelhante à regularização compartilhada. Realizar a seleção de atributos em uma etapa independente do treinamento equivale à compartilhar o mesmo subconjunto de atributos por todo os neurônios do classificador final.

Nosso objetivo aqui é utilizar uma técnica de seleção de atributos baseada em gradiente descendente, de forma que seja possível incorporá-la na arquitetura de uma rede neural. Como resultado, nós projetamos um classificador que opera a regularização de domínio e treinamento em conjunto, *online*. Dessa forma, o classificador final pode fazer uso de uma regularização de domínio distribuída aprendida pelos próprios neurônios da camada inicial da rede neural.

4.2 Perceptron Esparso

No Capítulo 2, apresentamos modelos de aprendizagem tal como classificadores lineares e redes neurais. Detalhamos posteriormente, no Capítulo 3, que a esparsidade é uma característica presente nas representações utilizadas por tarefas de NLP. Apesar de uso de elementos descrito por representações esparsas, os modelos aprendidos ainda seriam muito densos. Por exemplo, o *Perceptron* de Rosenblatt descrito na Seção 2.2.1, mesmo sendo simples e robusto, produziria um modelo denso de mesmo tamanho da entrada esparsa.

O *Perceptron* Esparso proposto por Goldberg (32) é uma variação do *Perceptron* que permite obter modelos menos esparsos, mantendo uma boa acurácia e eficiência computacional. Intuitivamente, essa variação do *Perceptron* consiste em manter apenas os atributos mais relevantes no modelo final. O grau de relevância dos atributos é medido através de um vetor de contadores $\mathbf{u}$. Inicialmente, todos os atributos são considerados irrelevantes $\mathbf{u} \leftarrow \mathbf{0}$, não participando da avaliação do modelo.

Os contadores de relevância são incrementadas à medida em que erros são cometidos na etapa de treinamento. O incremento tem como base os atributos ativos na entrada cuja predição gerou um erro. Um limiar L é utilizado para identificar os atributos mais relevantes. Ao fim de cada época, o grau de relevância dos atributos irrelevantes pode ser zerado. Assim, a interpretabilidade do limiar L é indicar o número mínimo de ocorrências que são necessárias para incorporar um atributo no modelo final, sendo essa uma das vantagens do *Perceptron* Esparso em relação a outro métodos de seleção de atributos. A seleção de atributos utilizando o *Perceptron* Esparso é descrita pelo Algoritmo 5.

Uma extensão do algoritmo do perceptron esparso é proposta por Motta (8) em seu *framework* IFIS. Essa extensão faz uso de *Dropout*, ao incrementar os contadores de relevância. Em vez de sempre adicionar o incremento, linha 9 do Algoritmo 5, Motta considerada a possibilidade de descartar a atualização desse atributo com probabilidade $P_{dropout}$.

Algoritmo 5 Perceptron Multiclasse Esparso, Goldberg (32)

Input:
$\mathcal{D}$ ▷ Dataset de treino
$EPOCHS$ ▷ Número de épocas
L ▷ Limiar de seleção
$ResetCounters$ ▷ Indica se $\mathbf{u}$ deve ser limpo
Output:
$\mathcal{F}$ ▷ Atributos selecionados

$W \leftarrow 0$ ▷ Matriz do modelo
$\mathbf{u} \leftarrow \mathbf{0}$ ▷ Vetor de acumuladores
$t \leftarrow 0$

while $t < EPOCHS$ **do**

  for each $(x, y) \in \mathcal{D}$ **do**
    $\hat{y} \leftarrow arg\max_{k \in 1...K}\{\mathbf{w}_k \cdot x[\mathbf{u} \geq L]\}$
    $\mathbf{w}_y \leftarrow \mathbf{w}_y + x$
    $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - x$
    $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{1}[\hat{y} \neq y][x \neq 0]$
  end for

  if ResetCounters **then**
    $\mathbf{u} \leftarrow \mathbf{u}[\mathbf{u} \geq L]$
  end if

  $t \leftarrow t + 1$
end while

$\mathcal{F} \leftarrow \{i \mid \mathbf{u}_i \geq L\}$
return $\mathcal{F}$

Como já mencionado na Seção 2.5.2, o *Dropout* é uma técnica comumente utilizada em redes neurais e tem o objetivo de promover uma maior regularização do modelo eliminando atributos correlacionados. Consequentemente, a introdução do *Dropout* possibilita uma seleção mais fina do número de atributos variando-se o valor de $P_{droptout}$. A seleção de atributos utilizando o *Perceptron* Esparso com *Dropout* é descrita pelo Algoritmo 6.

4.3 Regularização Compartilhada

Inspirados pela seleção de atributos através do *Perceptron* Esparso, propomos nessa seção duas variações do algoritmo do *Softmax*. Analogamente ao trabalho de Goldberg (32), as variações do *Softmax* que introduzimos geram modelos menos esparsos, ainda com uma boa acurácia e custo computacional.

Conforme apresentamos na Seção 2.3.3, diferentemente do *Perceptron*, a regra de aprendizado do *Softmax* faz uso do gradiente descendente, uma vez que sua função de ativação e perda são diferenciáveis. Nas linha 9 do Algoritmo 5 e linha 10, do Algoritmo 6 temos que a seleção de atributos depende exclusivamente de como o vetor de acumuladores **u** é atualizados.

Nessa seção, propomos duas variações do *Softmax* com relação a atualização desses acumuladores. A atualização dos acumuladores pode basear-se em erro de predição e gradiente da função de perda. A seguir, detalhamos essas duas abordagens e discutimos sobre a normalização do gradiente utilizado na atualização dos acumuladores na segunda abordagem. Em ambos os casos, incluímos o uso de *Dropout*, seguindo a extensão do *Perceptron* Esparso proposta por Motta (8).

4.3.1 Erro de Predição

O Algoritmo 7 descreve a utilização do *Softmax* para regularização de domínio. A variação do *Softmax* produz um modelo esparso a partir da seleção de atributos. Esse algoritmo tem como base *Perceptron* Esparso, em que a atualização dos acumuladores é feita com a partir do erro da predição, linha 12.

4.3.2 Gradiente da Perda

Nessa seção introduzimos uma segunda variação do *Softmax*. Ao contrário do *Perceptron* Esparso, esta variação utiliza a função de perda do *Softmax* para realizar a regularização de domínio. Além disso, incorporamos duas propriedades inferidas a partir do algoritmo do *Perceptron* Esparso:

(i) A atualização dos acumuladores é não decrescente, utilizando sempre valores não negativos.

(ii) O módulo da contribuição de cada amostra na atualização dos acumuladores é no máximo 1.

No Algoritmo 8, descrevemos esta variação do *Softmax* Esparso em que definimos a regra de atualização do vetor de acumuladores U com base no erro acumulado pelos neurônios.

Utilizamos o gradiente da perda com relação à saída da função de ativação dos neurônios, T, para medir o erro propagado por cada neurônio do classificador, linha 10. Então, obtemos uma máscara com os atributos ativos da entrada, e aplicamos o *Dropout* sobre ela, linha 11. Aplicar o *Dropout* neste ponto, ocasiona que atualizações sejam descartadas antecipadamente. Essa

máscara é aplicada sobre o erro, produzindo uma matriz de erro de mesma dimensão de **W** que contém as atualizações para cada atributo ponderadas pelo erro propagado por neurônio. Essa matriz, promove uma seleção de atributos mais fina do que a realizada pela abordagem do *Perceptron* Esparso. Para garantir as propriedades mencionadas anteriormente, consideremos uma função ϕ que normaliza o gradiente para obtermos valores dentro do intervalo $[0, 1]$, linha 12. Por fim, a atualização é de U é realizada com o erro acumulado dos atributos ao longo dos neurônios, linha 13.

4.3.3 Normalização do Gradiente

O Algoritmo 8 pressupõe uma função de normalização ϕ para o gradiente ∇ que é discutida a seguir.

A função sigmóide $\sigma : \Re \mapsto (0, 1)$, Equação 2-12, satisfaz as propriedades apresentadas anteriormente, pois permite mapear a derivada em cada dimensão para o intervalo desejado. Contudo, a sigmóide é inadequada para casos em que $\nabla_i << 0$, pois um erro de alto módulo seria desprezado $\sigma \approx 0$. Ou ainda, $\nabla_i = 0$ produziria indesejáveis incrementos em U, visto que $\sigma(0) = 0.5$.

Por outro lado, a função tangente hiperbólica $tahn : \Re \mapsto (-1, 1)$, em que $tanh(0) = 0$, possibilita uma definição mais direta de ϕ que satisfaça as propriedades já levantadas. Nas Equações 4-1 definimos algumas formas de descrever $\phi : \Re \mapsto [0, 1)$. Essas equações possuem um comportamento muito similar, apresentando leves diferenças na sensibilidade da atualização. Esta sensibilidade pode ser ajustada introduzindo uma constante que multiplica x ou $\phi(x)$, por exemplo.

$$\phi_1(x) = tanh(|x|) \tag{4-1a}$$

$$\phi_2(x) = tanh(x^2) \tag{4-1b}$$

$$\phi_3(x) = tanh^2(x) \tag{4-1c}$$

4.4 Regularização Distribuída

Nessa seção, apresentamos uma modificação da regularização do *Softmax* com gradiente de perda, introduzindo a regularização distribuída.

A regularização distribuída tem o objetivo de projetar a seleção fina de atributos da matriz ϕ_∇ na avaliação do modelo. Consequentemente, cada

neurônio será atualizado de forma distinta, com base no seu próprio conjunto de atributos ativos.

Com a regularização distribuída, utilizamos a seleção de atributos com base em gradiente descendente para especializar os neurônios a partir dos atributos que mais os sensibilizam. O Algoritmo 9 descreve esse procedimento.

4.5 Considerações

A utilização de *n-grams* de caracteres para descrever palavras produz uma representação de domínio com alta esparsidade, o que dificulta o trabalho de aprendizagem dos classificadores, pois tornam o modelo pesado, mantendo parâmetros de centenas ou milhares de atributos, até mesmo para os irrelevantes.

Neste capítulo, estendemos a regularização de domínio do *Perceptron* Esparso propondo o *Softmax* Esparso resultando em seleção de atributos que variam entre regularização de domínio compartilhada e distribuída. Além disso, as regularizações de domínio do *Softmax* com base em gradiente da função de perda resultam em uma generalização de seleção de atributos poderosa que pode ser incorporada a redes neurais. Dessa forma, podemos encapsular a modelagem apresentada como a primeira camada de uma rede neural, realizando regularização de domínio online.

Na fase de aprendizado, as regularizações compartilhada e distribuídas podem ser associadas, respectivamente, ao *Dropout* e *DropConnection* na camada inicial da rede. Pois, no primeiro descartamos atributos, enquanto que no segundo descartamos arestas. Entretanto, enquanto que a redução dos efeitos da correlação de atributos é tratada pelo *Dropout* e *DropConnection* em nível teórico, como uma média dos pesos de redes com diferentes arquiteturas, nossa abordagem tem o potencial de fato de refinar arquitetura da rede na camada de regularização.

As regularizações de domínio com base em gradiente, aqui discutidas, são uma contribuição original deste trabalho.

No capítulo a seguir, aplicamos as técnicas discutidas neste trabalho à tarefa de anotação morfossintática e apresentamos resultados que testificam a abordagem proposta para seleção de atributos.

Algoritmo 6 Perceptron Multiclasse Esparso com Dropout, Motta (8)

Input:
$\mathcal{D}$ ▷ Dataset de treino
$EPOCHS$ ▷ Número de épocas
L ▷ Limiar de seleção
$P_{dropout}$ ▷ Probabilidade de Dropout
$ResetCounters$ ▷ Indica se $\mathbf{u}$ deve ser limpo
Output:
$\mathcal{F}$ ▷ Atributos selecionados

$W \leftarrow 0$ ▷ Matriz do modelo
$\mathbf{u} \leftarrow \mathbf{0}$ ▷ Vetor de acumuladores
$t \leftarrow 0$

while $t < EPOCHS$ **do**

  for each $(x, y) \in \mathcal{D}$ **do**
    $\hat{y} \leftarrow arg\max_{k \in 1...K}\{\mathbf{w}_k \cdot x[\mathbf{u} \geq L]\}$
    $\mathbf{w}_y \leftarrow \mathbf{w}_y + x$
    $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - x$
    $\mathbf{r} \leftarrow$ *UniformDistribution()* ▷ Random $\in (0, 1)$
    $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{1}[\hat{y} \neq y][x \neq 0][\mathbf{r} \geq P_{dropout}]$ ▷ Descarta atualizações
  end for

  if ResetCounters **then**
    $\mathbf{u} \leftarrow \mathbf{u}[\mathbf{u} \geq L]$
  end if

  $t \leftarrow t + 1$
end while

$\mathcal{F} \leftarrow \{i \mid \mathbf{u}_i \geq L\}$
return $\mathcal{F}$

Algoritmo 7 Softmax Esparso – regularização compartilhada com erro de predição

Input:
- $\mathcal{D}$ ▷ Dataset de treino
- $EPOCHS$ ▷ Número de épocas
- L ▷ Limiar de seleção
- $ResetCounters$ ▷ Indica se $\mathbf{u}$ deve ser limpo
- η ▷ Taxa de aprendizagem

Output:
- $\mathcal{F}$ ▷ Atributos selecionados

$W \leftarrow 0$ ▷ Matriz do modelo
$\mathbf{u} \leftarrow \mathbf{0}$ ▷ Vetor de acumuladores
$t \leftarrow 0$

while $t < EPOCHS$ **do**

  for each $(x, Y) \in \mathcal{D}$ **do**
    $Z \leftarrow W \cdot x[\mathbf{u} \geq L]$
    $T \leftarrow \varsigma(Z)$ ▷ Softmax
    $Loss \leftarrow \xi(T, Y)$ ▷ Entropia Cruzada
    $W \leftarrow W - \eta \cdot \frac{\partial}{\partial W} Loss$ ▷ Gradiente Descendente

    $y \leftarrow arg\max_{k \in 1 \ldots K}\{Y_k\}$
    $\hat{y} \leftarrow arg\max_{k \in 1 \ldots K}\{T_k\}$

    $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{1}[\hat{y} \neq y][x \neq 0]$ ▷ Descarta atualizações
  end for

  if ResetCounters **then**
    $\mathbf{u} \leftarrow \mathbf{u}[\mathbf{u} \geq L]$
  end if

  $t \leftarrow t + 1$
end while

$\mathcal{F} \leftarrow \{i \mid \mathbf{u}_i \geq L\}$
return $\mathcal{F}$

Algoritmo 8 Softmax Esparso – regularização compartilhada com gradiente da perda

Input:
- $\mathcal{D}$ ▷ Dataset de treino
- $EPOCHS$ ▷ Número de épocas
- L ▷ Limiar de seleção
- $ResetCounters$ ▷ Indica se $\mathbf{u}$ deve ser limpo
- η ▷ Taxa de aprendizagem

Output:
- $\mathcal{F}$ ▷ Atributos selecionados

$W \leftarrow 0$ ▷ Matriz do modelo
$\mathbf{u} \leftarrow \mathbf{0}$ ▷ Vetor de acumuladores
$t \leftarrow 0$

while $t < EPOCHS$ **do**

  for each $(x, Y) \in \mathcal{D}$ **do**
    $Z \leftarrow W \cdot x[\mathbf{u} \geq L]$
    $T \leftarrow \varsigma(Z)$ ▷ Softmax
    $Loss \leftarrow \xi(T, Y)$ ▷ Entropia Cruzada
    $W \leftarrow W - \eta \cdot \frac{\partial}{\partial W} Loss$ ▷ Gradiente Descendente

    $\nabla \leftarrow \frac{\partial}{\partial T} Loss$ ▷ Erro propagado por neurônio

    $\bar{x} \leftarrow \mathbf{1}[x \neq 0]$ ▷ Máscara de atualização
    $\phi_{\nabla} \leftarrow \bar{x}^T \cdot \phi(\nabla)$ ▷ Matriz de erros

    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \sum_j \phi_{\nabla ij}$
  end for

  if ResetCounters **then**
    $\mathbf{u} \leftarrow \mathbf{u}[\mathbf{u} \geq L]$
  end if

  $t \leftarrow t + 1$
end while

$\mathcal{F} \leftarrow \{i \mid \mathbf{u}_i \geq L\}$
return $\mathcal{F}$

Algoritmo 9 Softmax Esparso – regularização distribuida

Input:
$\mathcal{D}$ ▷ Dataset de treino
$EPOCHS$ ▷ Número de épocas
L ▷ Limiar de seleção
$ResetCounters$ ▷ Indica se U deve ser limpo
η ▷ Taxa de aprendizagem
Output:
$\mathcal{F}$ ▷ Atributos selecionados

$W \leftarrow 0$
$U \leftarrow 0$
$t \leftarrow 0$

while $t < EPOCHS$ **do**

  for each $(x, Y) \in \mathcal{D}$ **do**
    $Z \leftarrow W[U \geq L] \cdot x$
    $T \leftarrow \varsigma(Z)$ ▷ Softmax
    $Loss \leftarrow \xi(T, Y)$ ▷ Entropia Cruzada
    $W \leftarrow W - \eta \cdot \frac{\partial}{\partial W} Loss$ ▷ Gradiente Descendente

    $\nabla \leftarrow \frac{\partial}{\partial T} Loss$ ▷ Erro propagado por neurônio

    $\bar{x} \leftarrow \mathbf{1}[x \neq 0]$ ▷ Máscara de atualização
    $\phi_{\nabla} \leftarrow \bar{x}^T \cdot \phi(\nabla)$ ▷ Matriz de erros

    $U \leftarrow U + \phi_{\nabla}$
  end for

  if ResetCounters **then**
    $U \leftarrow U[U \geq L]$
  end if

  $t \leftarrow t + 1$
end while

$\mathbf{m}_i \leftarrow \sum_j \{U_{ij} \geq L\}$
$\mathcal{F} \leftarrow \{i \mid \mathbf{m}_i \geq 0\}$
return $\mathcal{F}$

5
Anotação Morfossintática

No capítulo anterior, introduzimos um conjunto de técnicas para regularização de domínio que se propõem a reduzir a dimensionalidade de representações esparsas.

Neste capítulo, abordamos a tarefa de anotação morfossintática. Essa tarefa é uma das mais fundamentais em NLP, pois produz atributos importantes para tarefas de mais alto nível, uma vez que é um dos primeiros estágios em processamento de texto. Mostramos como redes neurais podem ser utilizadas para inferir as classes gramaticais de palavras a partir de uma abordagem profunda, comparando a acurácia de nossas abordagens com o atual estado-da-arte.

Em nosso sistema, aprendemos características específicas para o problema a partir do dado bruto, em que utilizamos *n-grams* de caracteres para descrever o contexto morfológico. Dessa forma, validamos as propriedades dos descritores morfológicos propostos no Capítulo 3.

Além disso, mostramos variações do nosso sistema combinando-o com as técnicas de regularização propostas, dado que a representação de domínio proposta tem alto grau de esparsidade. Assim, constatamos os conceitos de regularização de domínio introduzidos no Capítulo 4.

Na Seção 5.1, definimos o problema a ser resolvido por nosso sistema, e apresentamos os *corpora* utilizados para a avaliação de nossos métodos. Na Seção 5.2, mostramos algumas parametrizações da representações de domínio a partir de diferentes tamanhos de *n-grams*, destacamos como o tamanho da sequência de caractere impacta a complexidade do modelo e acurácia do modelo no conjunto de desenvolvimento. Na Seção 5.3, utilizamos os descritores de domínio propostos para elaborar de nosso sistema de anotação morfossintática. Na Seção 5.4, discutimos empiricamente sobre as técnicas de regularização propostas.

5.1
A tarefa

Conforme introduzimos na Seção 1.2, a anotação morfossintática, ou *part-of-speech*—POS *tagging*, é o processo de associar um POS a cada palavra

Tabela 5.1: Tamanho do Corpora

Corpus		Treino	Desenv.	Teste	Total	Tagset
McMorpho v1	Sentenças	42.022	2.211	9.141	53.374	41
	Tokens	957.439	50.232	213.794	1.221.465	
McMorpho v3	Sentenças	37.948	1.997	9.987	49.932	26
	Tokens	728.497	38.881	178.373	945.751	

Tabela 5.2: Estado-da-Arte das versões do Mac-Morpho

Ano	Corpus	Sistema	PWA (%)	OOV (%)
2015	McMorpho v1	NLM	97,57	93,38
2015	McMorpho v3	NLM	97,33	93,66

em um texto, que identifique sua classe gramatical, como no exemplo a seguir:

```
A/ART desertificação/N tornou/V crítica/ADJ a/ART
produtividade/N de/PREP 52/NUM mil/NUM km²/N em/PREP a/ART
região/N ./.
```

Nesta seção, detalhamos os *corpora* utilizados no desenvolvimento e avaliação desse sistema, de forma que possamos comparar a eficiência de nossa metodologia com os sistemas estado-da-arte atuais.

Apresentamos os *corpora* de língua portuguesa adotados. Em seguida, mostramos como geramos um conjunto de testes para medir os efeitos de erros datilográficos. E, por fim, apresentamos algumas estatísticas complementares.

5.1.1 Corpora

Utilizamos dois corpora na base experimental do nosso trabalho: o Mac-Morpho v1 e o Mac-Morpho v3. O Mac-Morpho é um *corpus* de português do Brasil contendo marcação de texto com anotações morfossintática, ou *part-of-speech*—POS *tags*. Sua primeira versão (33) foi publicada em 2003 e é formada por artigos jornalísticos do ano de 1994, retirados da Folha de São Paulo. O *corpus* foi originalmente dividido por seus autores em treino, desenvolvimento, e teste, respectivamente contendo 76%, 4% e 20% do *corpus*. Duas revisões do Mac-Morpho foram publicadas, v2 (34) e v3 (11), sendo esta última a principal versão utilizada neste trabalho. A Tabela 5.1 traz a divisão detalhada das versões do Mac-Morpho. A acurácia do atual sistema estado-da-arte para esses *corpora* é descrito na Tabela 5.2.

5.1.2

Erros Datilográficos

Visto que propomos uma representação a nível de caractere com base em aspectos morfológicos, geramos uma versão alternativa do Mac-Morpho v3 introduzindo erros datilográficos. Além das avaliações convencionais, observamos o quanto sua predição foi afetada por erros de digitação. Para essa finalidade, geramos uma versão do *corpus* em que inserimos erros apenas no conjunto testes, mantendo o conjunto de treinamento e desenvolvimento intocável. Nosso objetivo é observar os impactos de erros datilográficos ao utilizar a representação proposta. Este é um problema presente em aplicações reais, mas que não tem sido endereçado em conjunto com tarefas de NLP, como a análise morfossintática. A análise do desempenho da anotação morfossintática diante de erros datilográficos é uma contribuição original deste trabalho.

Para tal, definimos neste trabalho três operações a nível de caractere para a inserção de erros para uma dada palavra:

- Permutação: dois caracteres vizinhos são permutados.
- Descapitalização: o primeiro caractere é colocado em caixa baixa.
- Remoção: um caractere em uma posição aleatória é removido.

Para gerar uma versão do conjunto de testes que contenha erros datilográficos, consideramos apenas as palavras com mais de 5 caracteres e a probabilidade de ocorrência de erro de 10% para descapitalização e de 1% para permutação e remoção.

5.1.3 Estatísticas dos Corpora

Ao longo de suas revisões, o *tag set* do Mac-Morpho sofreu várias reformulações. Originalmente haviam 46 *tags*, das quais as *tags* referentes a sinais de pontuação eram ignoradas na avaliação dos sistemas, em que apenas demais 21 tags contabilizadas. Em suas reversões, uma das alterações mais significantes inseridas foi a unificação das tags de pontuação. Dessa forma, os sinais de pontuação passaram a ser incluídos na avaliação do sistema, como indica os experimentos com o sistema NLM apresentado por Fonseca (11). A distribuição detalhada de tokens por tagsets nesta versão são descritas pelas Figuras 5.1 e 5.2

Como destacamos na Seção 3.3, os sistemas que utilizam uma abordagem baseadas em vocabulário ignoram a caixa das letras. Dessa forma, ao construir sua representação a nível de palavra, convertem o texto original para caixa baixa. Essa insensibilidade à caixa diminui o número de caracteres utilizados na composição das palavras. Por um lado, isso ajuda na diminuição da esparsidade

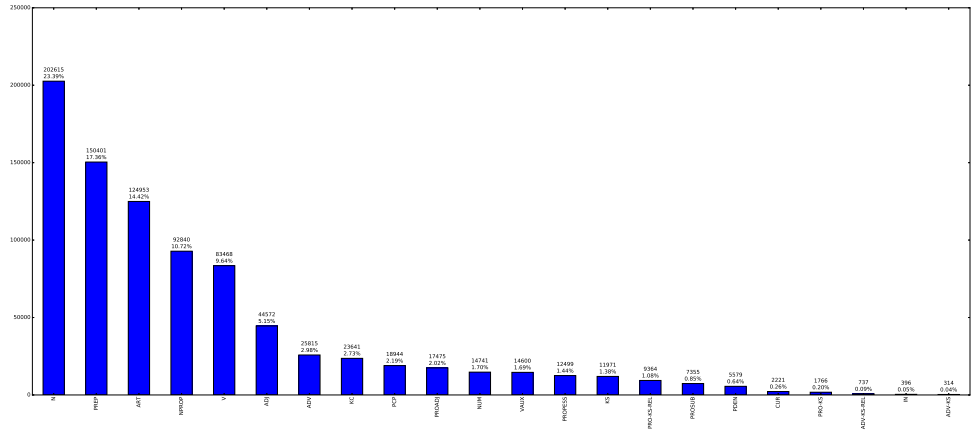

Figura 5.1: Histograma de distribuição do *tag set* do Mac-Morpho v1. Este histograma considera apenas as 21 das suas 41 *tags*, em que as *tags* de pontuação são desconsideradas na avaliação dos sistemas que utilizam essa versão.

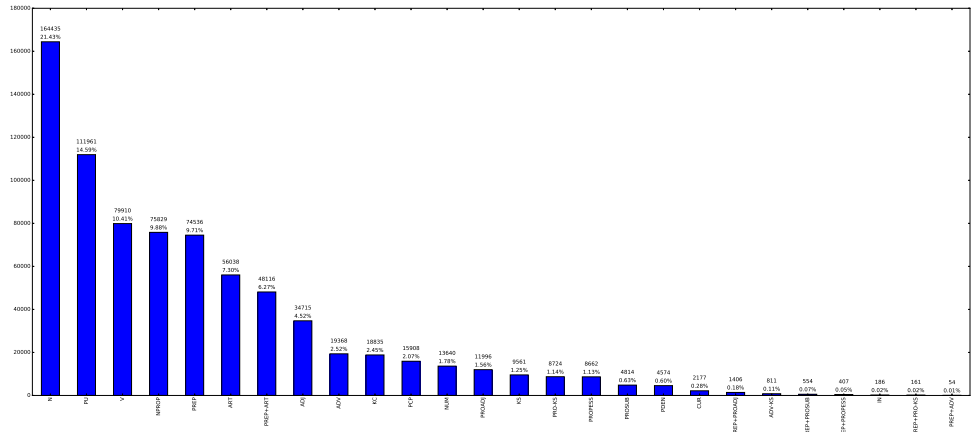

Figura 5.2: Histograma de distribuição do *tag set* do Mac-Morpho v3. Nessa versão, além de apresentar um *tag set* de 26 categorias, as *tags* de pontuação são agrupadas econsideradas na avaliação do sistema.

Tabela 5.3: Estatísticas adicionais do corpora utilizado. Observamos como o tamanho do vocabulário e a quantidade de OOV variam dependendo da presença de sensibilidade à caixa da letra.

Corpus	Caixa da letra	Caracteres	Vocabulário	OOV	Tamanho médio da palavra
McMorpho v1	insensível	75	50.517	7.221	4,123
	sensível	114	57.410	8.336	4,123
McMorpho v3	insensível	70	44.980	6.839	4,324
	sensível	109	50.661	7.983	4,324
McMorpho v3 c/ erros datilográficos	sensível	109	50.661	9.789	4,323

da representação de domínio ao diminuir o tamanho do vocabulário; por outro, essa insensibilidade ocasiona uma perda de de informação morfológica. Na Tabela 5.3, observamos a diferença de nossos descritores brutos de domínio, comparando a sensibilidade e insensibilidade à caixa das palavras.

Ainda, na Tabela 5.3, notamos como a introdução de erros datilográficos no conjunto de testes do Mac-Morpho v3 incrementa a quantidade de de palavras fora do vocabulário.

5.2 Representação de Domínio

Nessa seção, definimos os parâmetros utilizados em nossos descritores de domínio na validação das técnicas propostas neste trabalho. Utilizaremos apenas os corpus Mac-Morpho v3 no desenvolvimento de nossa metodologia. Os corpora Mac-Morpho v1 e Mac-Morpho v3 com erros datilográficos são utilizados apenas na avaliação final de nossos sistemas. Em todo caso, mantemos a caixa original do texto, em uma abordagem sensível à caixa.

Ao manter a caixa original das palavras, em que apenas substituindo os algorismos `[0-9]` pelo dígito `0` do *corpus* Mac-Morpho v3, obtemos uma versão desse *corpus* em que os *tokens* de seu conjunto de treinamento são escritos com base em um conjunto de 109 caracteres, como exibido na Tabela 5.3. Uma vez que utilizamos "*hashing*" para indexar os *n-grams* através da tabela $\mathcal{M}$, como alternativa à representação matricial, temos um número de combinações de *n-grams* que depende diretamente das sequências de caracteres presentes no conjunto de treinamento utilizado. Dessa forma, utilizamos uma quantidade muito inferior de n-grams se comparado com o número totais de combinação. No caso de uma representação baseada em *trigrams* de caracteres, cada palavra do Mac-Morpho v3 é pelo vetor de seu contexto morfológico de 50.073 posições

Tabela 5.4: Dimensionalidade da representação de domíno do Mac-Morpho v3.

	n-gram	Palavra Morfológica	Janela de Contexto	Contexto Morfológico	Esparsidade Média (%)
unigram	1	109	3	327	96,033
bigram	2	2.731	3	8.193	99,841
trigram	3	16.691	3	50.073	99,974
tetragram	4	51.772	3	10.3444	99,991
pentragram	5	100.708	3	302.124	99,995

ao invés de 3.885.087 (3×109^3).

Na Tabela 5.4, mostramos a dimensionalidade obtidas variando-se o tamanho dos *n-grams*. Apesar da indexação através de "*hashing*", ainda observamos um alto grau de esparsidade na representação obtida.

5.3 Anotador Morfossintático

Na seção anterior, aplicamos os descritores de domínio introduzidos pelo Capítulo 3 às versões do *corpus* Mac-Morpho. Como resultado, apresentamos as algumas propriedades das representações obtidas.

É importante notar que, assim como Fonseca (11), nos baseados na abordagem convolucional de Collobert (9) representando cada palavra por uma janela de contexto. Dessa forma, os sistemas apresentados nesse trabalho adotam uma abordagem *token*-a-*token*. Em nossos experimentos, utilizamos uma janela de tamanho três, como ilustrada pela Figura 3.3.

Nesta seção, nós descrevemos o ambiente computacional utilizado. Em seguida, apresentamos como empregamos as técnicas discutidas nessa dissertação para desenvolver nosso sistema de anotação morfossintática. Então, validamos a eficiência desse sistema experimentalmente, expondo os resultados obtidos.

5.3.1 Ambiente Computacional

Para nossos experimentos, utilizamos processamento em placa gráfica através do seguinte ambiente de desenvolvimento:

- Python 3.5.1
- Cuda 7.5
- Theano 0.8.2
- Lasagne 0.1

Diante desse ambiente, configuramos uma máquina com a configuração de *hardware* a seguir:

– CPU Intel(R) Core(TM) i7-5960X 3.00GHz
– GPU NVIDIA Tesla K40 GK110BGL
– RAM 64GiB
– Placa-mãe Gigabyte X99-Gaming G1 WIFI
– 1TB SSD Samsung 850 931GiB

5.3.2 Modelo de Aprendizagem

Nossa metodologia baseia-se no uso de redes neurais artificiais para o aprendizado do anotador morfossintático.

Adotamos o *Multilayer Perceptron*, Seção 2.4, como classificador base dos experimentos desta dissertação, utilizando o *Softmax* como função de ativação e a Entropia Cruzada como função de perda. Além disso, introduzimos uma camada de *Dropout* na entrada da rede, como também entre suas camadas internas. Durante o desenvolvimento do nosso sistema, experimentamos diferentes parametrizações do MLP, variando a quantidade de camadas ocultas e a quantidade de neurônio por camada.

O aprendizado do modelo é feito com gradiente descendente através do algoritmo de otimização de Nesterov Momentum, em que fixamos a taxa de aprendizagem em $0,01$ e o momentum em $0,9$. Além disso, adotamos uma abordagem de atualização estocástica através de *mini-batches* de tamanho 500.

5.3.3 Experimentos

Nessa seção, mostramos alguns experimentos que validam os descritores morfológicos propostos como uma representação eficaz para a tarefa de anotação morfossintática.

Primeiramente, adotamos o mesmo tamanho de *n-grams* utilizado por Carlson (23), $n = 3$. Nosso anotador utiliza um *Multilayer Perceptron* com duas camadas ocultas, combinado com camadas de *Dropout* intercalando as camadas do MLP com probabilidade de 0.5, como também uma camada na entrada da rede com probabilidade de *drop* de 0.2.

A Tabela 5.5 mostra o desempenho desse anotador no conjunto de desenvolvimento ao variar a quantidades de neurônios das camadas ocultas, em que a coluna Δ_{Loss} expressa a diferença entre as o valor médio da função de perda no conjunto de treino e de desenvolvimento. Quanto mais próximo de zero for essa variação, mais confiável é a acurácia obtida no conjunto de desenvolvimento. Uma variação muito negativa ou positiva são indícios,

Tabela 5.5: Experimento utilizando o MLP com *Dropout* em que combinamos a representação de domínio baseada em *trigrams* de caracteres. Este experimento utiliza o conjunto de desenvolvimento para a avaliação, onde fazemos uma busca do número de neurônios por camanda ao longo de 25 épocas de treinamento.

Neurônios	Acurácia	Δ_{Loss}
200	96,91	0,064346
400	97,16	0,034957
600	97,24	0,023823
800	**97,33**	**0,017675**
1000	97,31	0,012991
1200	97,33	0,009642

respectivamente, de sobre-ajuste ou sub-ajuste. Ainda, notamos no campo Δ_{Loss} que ainda há "espaço" para aumentar o número de épocas sem que haja sobre-ajuste.

Em geral, adotamos 800 neurônios por camada, visto que uma quantidade maior apresenta resultados não muito diferentes, e principalmente, porque torna o modelo mais complexo e computacionalmente mais custoso tanto pela quantidade de parâmetros quanto pelo número de épocas necessárias para gerá-los.

O uso de *Dropout* em uma entrada esparsa parece contra-intuitivo, visto que este diminui ainda mais a participação dos atributos durante o treinamento.

A seguir apresentamos dois experimentos em que podemos comparar os efeitos do *Dropout* em nosso sistema.

Para cada descritor apresentado na Tabela 5.4, fizemos dois experimentos, tendo como referência o conjunto de desenvolvimento do Mac-Morpho v3. Testamos as diferentes variações no tamanho dos *n-grams* utilizamos um MLP com duas camadas ocultas, cada uma com 800 neurônios. Nestes experimentos, não utilizamos *Dropout*. O treinamento foi realizado ao longo de 50 épocas, cujos resultados são apresentados na Tabela 5.6.

Em seguida, avaliamos cada representação com o MLP com *Dropout* utilizado anteriormente. O resultado é apresentado na Tabela 5.7.

Nestes experimentos observamos que, a medida em que aumentamos o tamanho dos *n-grams* maior se torna a acurácia do modelo aprendido no conjunto de desenvolvimento devido a maior quantidade de informação morfológica. Assim, mais descritiva se torna nossa representação, apesar de que o aprendizado do modelo se torna bem mais custoso com o aumento da esparsidade ocasionada pelo uso de sequências longas.

Notamos que, a representação baseada em *unigrams* apresenta uma queda

Tabela 5.6: Experimento utilizando o MLP sem *Dropout* em que combinamos a representação de domínio proposta para aprender um anotador morfossintático. Utilizamos o conjunto de desenvolvimento para a avaliação os efeitos de diferentes tamanhos de *n-gram*.

Descritor	Dimensionalidade	Acurácia	Δ_{Loss}
unigram	327	94,29	-0,086028
bigram	8.193	96,64	-0,105428
trigram	**50.073**	**97,07**	**-0,085727**
tetragram	10.3444	97,18	-0,080577
pentragram	**302.124**	**97,33**	**-0,085799**

Tabela 5.7: Experimento utilizando o MLP com *Dropout* em que combinamos a representação de domínio proposta para aprender um anotador morfossintático. Utilizamos o conjunto de desenvolvimento para a avaliação os efeitos de diferentes tamanhos de *n-gram*.

Descritor	Dimensionalidade	Acurácia	Δ_{Loss}
unigram	327	91,16	0,313006
bigram	8.193	96,93	0,077709
trigram	**50.073**	**97,52**	**-0,001717**
tetragram	10.3444	97,67	-0,039409
pentragram	**302.124**	**97,71**	**-0,041524**

em seu desempenho ao utilizar *Dropout*. De fato, o uso de *Dropout* pode prejudicar o desempenho do aprendizado para representações esparsas, visto que os atributos possuem uma baixa frequência na participação da atualização do modelo. Descartar um atributo diminui ainda mais essa frequência. Esse comportamento poderia ser observado de forma mais evidente ao avaliarmos modelos treinados ao longo de uma quantidade menor de épocas, e.g. abaixo de 25.

Sem o uso de *Dropout*, ao utilizar um grande número de épocas no intuito de aumentar a participação de atributos relevantes e obter-se uma melhor generalização pode facilitar que o modelo sofra sobre-ajuste. Entretanto, ao utilizamos o *Dropout* para descorrelacionar atributos que aparecem em conjunto, incrementando "virtualmente" a diversidade das amostras e combinações de atributos. Portanto, aumentamos o número de épocas para não perdermos importantes ativações, ao passo que o uso de *Dropout* favorece o treinamento de um modelo que com uma boa generalização do domínio.

5.4 Regularização de Domínio

Na seção anterior, vimos que nossos descritores possibilitam que anotadores baseados em MLP de duas camadas produzam excelente resultados.

Contudo, nossa representação possui uma alta dimensionalidade, aumentando o custo computacional.

Agora, comparamos os efeitos das técnicas de regularização de domínio propostas por esta dissertação. Motta (8) discute o uso de *Dropout* a atualização dos contadores utilizados na seleção de atributos do *Perceptron* Esparso. Nosso enfoque não é refazer este tipo de avaliação, mas sim comparar as abordagens de atualização propostas no que diz respeito ao uso do gradiente da perda. Para isso, observaremos resultados da regularização que tem como base o erro de predição, semelhante ao *Perceptron* Esparso.

Nesta seção, utilizaremos o *Softmax* Esparso com erro de predição como modelo, Algoritmo 7, análogo ao *Perceptron* Esparso com *Dropout* de Motta (8). Em seguida, comparamos o resultado da seleção de atributo com o *Softmax* Esparso com regularização baseada em gradiente da perda, 8, uma contribuição original desse trabalho. Por fim, apresentamos o desempenho do *Softmax* Esparso com regularização distribuída.

5.4.1 Regularização com Erro de Predição

A técnica convencional de seleção de atributos que tem como base o *Perceptron* Esparso, Algoritmo 5, atualiza um vetor de contadores sempre que um erro de classificação é cometido.

Avaliamos o comportamento da regularização compartilhada do *Softmax* Esparso, Algoritmo 8, que utiliza essa mesma abordagem para como função de atualização de seus acumuladores internos. No gráfico da Figura 5.3, observamos que a acurácia não é afetada significantemente ao reduzirmos o percentual de atributos utilizados. O que é feito aumentando a pressão seletiva à medida em que incrementamos o valor de L. A resposta da taxa de redução da atributos ao aumento da pressão seletiva é bem semelhante ao *Perceptron* Esparso. Neste experimento foi utilizado a representação de domínio com base em *trigrams*.

5.4.2 Regularização com Gradiente da Perda

Sob as mesmas condições do experimento anterior, mostramos a seguir o comportamento da regularização compartilhada do *Softmax* Esparso com base no gradiente da perda, Algoritmo 8. Observamos na Figura 5.4 que L exerce uma pressão seletiva bem maior, dado que o incremento dos acumuladores dependem do valor do erro, e não apenas da condição de haver ou não erro.

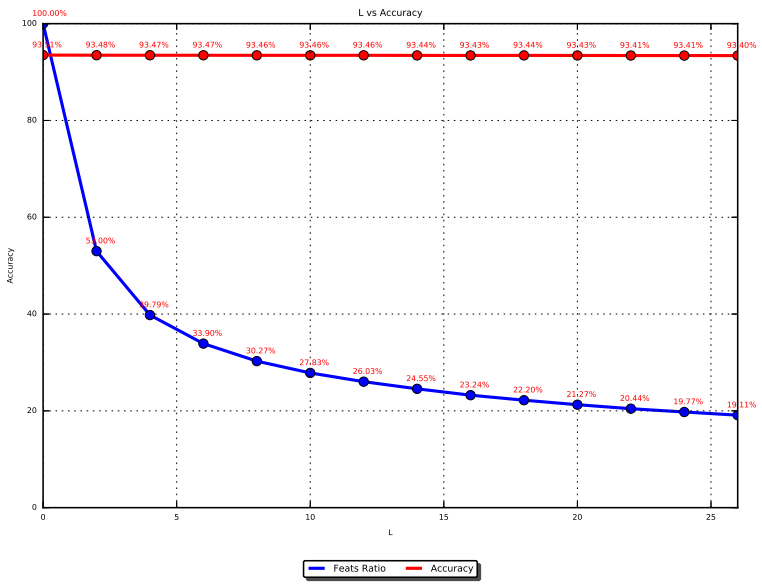

Figura 5.3: Experimento com *Softmax* Esparso que ilustra o comportamento da seleção de atributos com base no erro de predição. Cada iteração de L utiliza um modelo treinado ao longo de 25 épocas. Neste experimento adotamos $P_{dropout} = 0$.

5.4.3 Regularização Distribuída

Ainda sob as mesmas condições do experimento anterior, mostramos a seguir o comportamento da regularização distribuída do *Softmax* Esparso, Algoritmo 9, que assim como a versão anterior, tem como base o gradiente da perda. Observamos na Figura 5.5 que L exerce uma pressão seletiva ainda maior à versão apresentada na Figura 5.4. Isso acontece pois além do incremento os acumuladores dependerem do valor do erro, a derivada ainda é permanece distribuída entre os neurônios, ao invés de ser acumulada como acontece no caso de sua versão no Algoritmo 8.

A regularização distribuída é uma generalização da regularização compartilhada com base no erro, que permite selecionar quais arestas são incluídas no modelo final do *Softmax* Esparso. Entretanto, o tamanho do modelo aumenta, visto que necessitamos de uma matriz de acumuladores U de mesma dimensão de W, com já discutido no capítulo anterior.

Para verificar que de fato cada neurônio se especializa em atributos específicos, foram desenvolvidas algumas métricas mostradas na Figura 5.5, são elas:

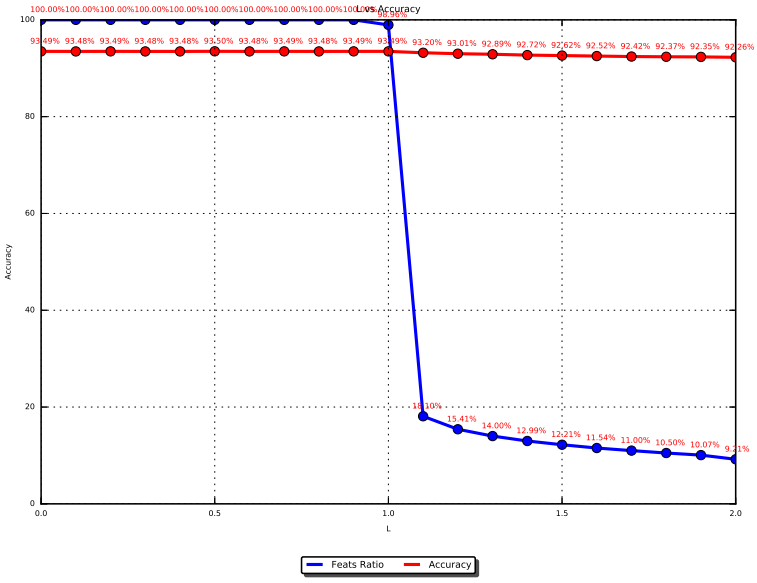

Figura 5.4: Experimento com *Softmax* Esparso que ilustra o comportamento da seleção de atributos com base gradiente da perda. Cada iteração de L utiliza um modelo treinado ao longo de 25 épocas. Neste experimento adotamos $P_{dropout} = 0$.

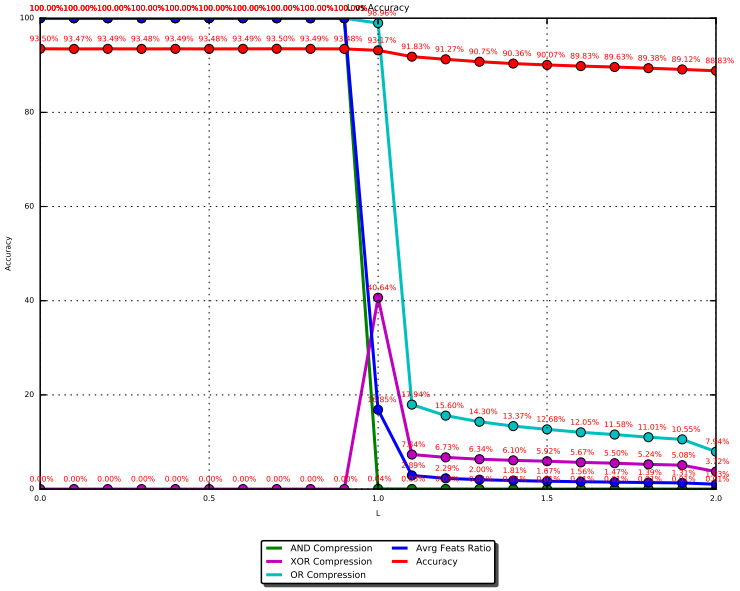

Figura 5.5: Experimento com *Softmax* Esparso que demonstra o comportamento da seleção de atributos com base na regularização distrbuida. Cada iteração de L utiliza um modelo treinado ao longo de 25 épocas. Neste experimento adotamos $P_{dropout} = 0$.

– *AND Compression*: proporção de atributos que são utilizados por todos os neurônios.
– *OR Compression* proporção de atributos que são utilizados por pelo menos um neurônio.
– *XOR Compression* proporção de atributos que são utilizados por apenas um neurônio.
– *Avrg Compression* proporção média de atributos utilizados por neurônio.

5.5 Experimentos Finais

Na seção anterior, examinamos as técnicas de regularização com um classificador simples, como é o caso do *Softmax* Esparso. Utilizamos o conjunto de desenvolvimento do Mac-Morpho v3 para analisar o comportamento da seleção de atributos, em que validamos as técnicas de regularização compartilhada e distribuídas.

Utilizamos as técnicas de regularização propostas nessa dissertação para avaliar o desempenho das variantes do nosso sistema de anotação. Nessa seção, medimos o desempenho destas variantes através dos *corpora* detalhados anteriormente na Tabela 5.3.

Primeiramente, avaliamos o anotador baseado em MLP que introduzimos na Seção 5.3. Em seguida, apresentamos os resultados desse anotador combinado com uma regularização *offline*.

5.5.1 Anotador Simples

A seguir, avaliamos nosso anotador morfossintático em sua versão simples. Este versão consistem em utilizar apenas o MLP em conjunto com o *Dropout*, como descrito na Seção 5.3.

Na Tabela 5.8 mostramos o desempenho deste anotador combinado com os descritores de domínio baseados em *trigrams*. Esse sistema foi treinado ao longo de 50 épocas e utiliza *trigrams* de caracteres como descritores de domínio em conjunto com um MLP de duas camadas ocultas com 800 neurônios combinadas com *Dropout* na entrada da rede de $0,2$, e de $0,5$ nas camadas intermediárias.

Esse sistema contém uma alta quantidade de parâmetros para ser aprendidos, o que resulta em um tempo de treinamento por época relativamente alto. Por exemplo, ao treinar o modelo com o *corpus* MacMorpho v3, só na primeira camada da rede neural temos 50.073 parâmetros para cada um dos 800 neurônios. O que resulta em um tempo médio de 190 segundos por época.

Tabela 5.8: Desempenho do anotador morfossintático simples avaliado no *corpora* do MacMorpho.

Corpus	PWA (%)	OOV (%)
McMorpho v1	96,67	90,07
McMorpho v3	96,76	89,85
McMorpho v3 c/ erros datilográficos	96,07	83,58

Tabela 5.9: Desempenho do anotador morfossintático com regularização *offline* avaliado no *corpora* do MacMorpho.

Corpus	Atributos Utilizados (%)	PWA (%)	OOV (%)
MacMorpho v3	19,72	96,65	89,03
MacMorpho v3 c/ erros datilográficos	19,72	95,05	82,67

5.5.2 Regularização Offline

A seguir, avaliamos nosso anotador morfossintático utilizando uma regularização *offline*.

Este versão consistem em utilizar o *Softmax* Esparso com a regularização baseada na predição do erro para reduzir a dimensionalidade do domínio. Em seguida, alimentamos o MLP com *Dropout* apenas com o atributos selecionados pela etapa de regularização.

Na Tabela 5.9 mostramos o desempenho deste anotador combinado com os descritores de domínio baseados em *trigrams*. Um treinamento de um simples baseado no *Softmax* Esparso foi realizado ao longo de 25 épocas, com o limiar de seleção $L = 25$. Em seguida, esse sistema foi treinado ao longo de 50 épocas e utiliza *trigrams* de caracteres como descritores de domínio em conjunto com um MLP de duas camadas ocultas com 800 neurônios combinadas com *Dropout* na entrada da rede de $0,2$, e de $0,5$ nas camadas intermediárias.

Esse sistema contém uma quantidade reduzida de parâmetros se comparado ao aprezentado anteriormente. Ao treinar o modelo final com o *corpus* MacMorpho v3, na primeira camada da rede neural utilizamos $19,72$ % (9.873/50.073) dos parâmetros para cada um dos 800 neurônios. O que resulta na redução tempo médio de treinamento por época de 190 para 46 segundos.

6
Conclusão

Neste trabalho utilizamos modelos de redes neurais simples, mas que combinados com bons descritores morfológicos, são capazes realizar a tarefa de anotação morfossintática.

Introduzimos descritores de domínio que utilizam sequencia de caracteres, *n-grams*, para representar o contexto morfológico das palavras. Discutimos sobre as abordagens que os sistemas de processamento de texto adotam para representar palavras, em que destacamos que a esparsidade uma característica implícita em processamento de texto por se tratar de atributos nominais.

Abordamos a redução de esparsidade introduzindo novas técnicas de regularização de domínio onde introduzimos o *Softmax* Esparso. Os conceitos desse modelo se mostram extensíveis à redes neurais, em que apresentamos a regularização com base em gradiente, viabilizando um processo de regularização *online*. Além disso, demonstramos como essa regularização permite que neurônios da primeira camada se especializem em certos atributos.

As variações de regularização apresentadas, regularização compartilhada e distribuída, podem ser comparadas ao *Dropout* e *DropConnect*. Entretanto, nossos experimentos indicam que a regularização proposta, diferentemente do *Dropout* e *DropConnect*, possibilitariam que a arquitetura da camada de entrada de redes neurais sejam refinadas.

Propomos as seguintes linhas de pesquisa como trabalhos futuros.

- Explorar a viabilidade e o uso da regularização proposta em camadas escondidas das redes neurais.
- Experimentar a elaboração e *embeddings* dos descritores baseados em *n-grams* que foram propostos.
- Utilizar o Algoritmo de Viterbi para aprimorar nosso anotador morfossintático.

A
Demonstrações Complementares

A.1
Sigmóide

A seguir, demonstramos da derivada da função sigmóide, referenciada na Seção 2.2.3.1.

Dado,

$$\sigma(u) = \frac{1}{1 + e^{-\lambda u}} \tag{A-1}$$

Temos que,

$$\begin{aligned} \frac{d}{du}\sigma(u) &= \frac{-e^{-u}(-1)}{[1 + e^{-u}]^2}\ du \\ &= \frac{e^{-u}}{[1 + e^{-u}]^2}\ du \end{aligned} \tag{A-2}$$

Rescrevendo,

$$\begin{aligned} \sigma'(u) &= \frac{e^{-u} + 1 - 1}{[1 + e^{-u}]^2}\ du \\ &= \frac{e^{-u} + 1}{[1 + e^{-u}]^2} - \frac{1}{[1 + e^{-u}]^2}\ du \\ &= \sigma(u) - \sigma(u)^2\ du \end{aligned} \tag{A-3}$$

Logo,

$$\sigma'(u) = \sigma(u)(1 - \sigma(u))\ du \tag{A-4}$$

A.2
Entropia Cruzada

A seguir, demonstramos a derivada com relação a w da função de perda com base na entropia cruzada, referênciada na Seção 2.2.3.3.

Dado,

$$L = -[y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})] \tag{A-5}$$

Derivando com relação a $\hat{y}$,

$$\begin{aligned} \frac{d}{d\hat{y}}L &= (-1)\ \left[y\frac{1}{\hat{y}} + (1 - y)\frac{1}{1 - \hat{y}}(-1)\right]\ d\hat{y} \\ &= (-1)\ \left[\frac{y}{\hat{y}} - \frac{(1 - y)}{1 - \hat{y}}\right]\ d\hat{y} \end{aligned} \tag{A-6}$$

Uma vez que temos $\hat{y} = \sigma(z)$ da Seção 2.2.3.1, onde $z = wx$, aplicamos a regra da cadeia e derivamos com relação a w, assim,

$$\begin{aligned}
\frac{d}{dw}L &= \frac{dL}{d\hat{y}}\ \frac{d\hat{y}}{dz}\ \frac{dz}{dw} \\
&= (-1)\ \left[\frac{y}{\hat{y}} - \frac{(1-y)}{1-\hat{y}}\right]\ \sigma'(z)\ \frac{dz}{dw} \\
&= (-1)\ \left[\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right]\ \sigma'(z)\ x \\
&= (-1)\ \left[\frac{[1-\sigma(z)]y}{[1-\sigma(z)]\sigma(z)} - \frac{\sigma(z)(1-y)}{\sigma(z)[1-\sigma(z)]}\right]\ \sigma'(z)\ x \\
&= (-1)\ \left[\frac{y - y\cancel{\sigma(z)} - \sigma(z) + y\cancel{\sigma(z)}}{\sigma(z)[1-\sigma(z)]}\right]\ \sigma'(z)\ x
\end{aligned} \tag{A-7}$$

Substituindo $\sigma(z)[1-\sigma(z)]$, segue que,

$$\begin{aligned}
\frac{d}{dw}L &= (-1)\ \left[\frac{y-\sigma(z)}{\cancel{\sigma'(z)}}\right]\ \cancel{\sigma'(z)}\ x \\
\therefore \frac{d}{dw}L &= (\sigma(wx) - y)\ x
\end{aligned} \tag{A-8}$$

Referências bibliográficas

[1] BRILL, E.; MOONEY, R. J.. **An overview of empirical natural language processing**. AI magazine, 18(4):13–24, 1997.

[2] JURAFSKY, D.; MARTIN, J. H.. **Speech and Language Processing (2Nd Edition)**. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[3] BRILL, E.. **A simple rule-based part of speech tagger**. In: PROCEEDINGS OF THE THIRD CONFERENCE ON APPLIED NATURAL LANGUAGE PROCESSING, ANLC '92, p. 152–155, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[4] BRILL, E.. **Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging**. Comput. Linguist., 21(4):543–565, Dec. 1995.

[5] DOS SANTOS, C. N.; MILIDIÚ, R. L.. **Entropy Guided Transformation Learning**, p. 159–184. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[6] UNESON, M.. **When errors become the rule: Twenty years with transformation-based learning**. ACM Comput. Surv., 46(4):50:1–50:51, Apr. 2014.

[7] FERNANDES, E. L. R.. **Entropy guided feature generation for structure learning**. PhD thesis, PUC-Rio, 2012.

[8] MOTTA, E. N.. **Induçao e Seleçao Incrementais de Atributos no Aprendizado Supervisionado**. PhD thesis, PUC-Rio, 2014.

[9] COLLOBERT, R.; WESTON, J.; BOTTOU, L.; KARLEN, M.; KAVUKCUOGLU, K. ; KUKSA, P.. **Natural language processing (almost) from scratch**. Journal of Machine Learning Research, 12(Aug):2493–2537, 2011.

[10] JOACHIMS, T.. **Text categorization with Support Vector Machines: Learning with many relevant features**, p. 137–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[11] FONSECA, E. R.; G ROSA, J. L. ; ALUÍSIO, S. M.. **Evaluating word embeddings and a revised corpus for part-of-speech tagging in portuguese**. Journal of the Brazilian Computer Society, 21(1):1–14, 2015.

[12] DOS SANTOS, C. N.; ZADROZNY, B.. **Learning character-level representations for part-of-speech tagging.** In: ICML, p. 1818–1826, 2014.

[13] MANNING, C. D.. **Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?**, p. 171–189. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[14] ROSENBLATT, F.. **The perceptron: a probabilistic model for information storage and organization in the brain.** Psychological review, 65(6):386–408, 1958.

[15] RATNAPARKHI, A.; OTHERS. **A maximum entropy model for part-of-speech tagging**. In: PROCEEDINGS OF THE CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, volumen 1, p. 133–142, 1996.

[16] COLLINS, M.. **Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms**. In: PROCEEDINGS OF THE ACL-02 CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING - VOLUME 10, EMNLP '02, p. 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[17] MCCULLOCH, W. S.; PITTS, W.. **A logical calculus of the ideas immanent in nervous activity**. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

[18] RUDER, S.. **An overview of gradient descent optimization algorithms**, Jan. 2016.

[19] SETHI, B.; GOEL, R.. **Exploring adaptive learning methods for convex optimization**. 2015.

[20] NIELSEN, M. A.. **Neural Networks and Deep Learning**. Determination Press, 2015.

[21] KLINE, D. M.; BERARDI, V. L.. **Revisiting squared-error and cross-entropy functions for training neural network classifiers**. Neural Computing & Applications, 14(4):310–318, 2005.

[22] BISHOP, C. M.. **Pattern recognition**. Machine Learning, 128, 2006.

[23] CARLSON, G.. **Techniques for replacing characters that are garbled on input**. In: PROCEEDINGS OF THE APRIL 26-28, 1966, SPRING JOINT COMPUTER CONFERENCE, AFIPS '66 (Spring), p. 189–193, New York, NY, USA, 1966. ACM.

[24] RISEMAN, E. M.; EHRICH, R. W.. **Contextual word recognition using binary digrams**. IEEE Transactions on Computers, C-20(4):397–403, April 1971.

[25] RISEMAN, E. M.; HANSON, A. R.. **A contextual postprocessing system for error correction using binary n-grams**. IEEE Transactions on Computers, C-23(5):480–493, May 1974.

[26] DAMASHEK, M.. **Gauging similarity with n-grams: Language-independent categorization of text**. Science, 267(5199):843, 1995.

[27] HARGRAVE III, J. E.; SAVOUREL, Y. I.. **Machine assisted translation tools utilizing an inverted index and list of letter n-grams**, Oct. 10 2000. US Patent 6,131,082.

[28] HOUVARDAS, J.; STAMATATOS, E.. **N-Gram Feature Selection for Authorship Identification**, p. 77–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[29] KANARIS, I.; STAMATATOS, E.. **Webpage genre identification using variable-length character n-grams**. In: 19TH IEEE INTERNATIONAL CONFERENCE ON TOOLS WITH ARTIFICIAL INTELLIGENCE(ICTAI 2007), volumen 2, p. 3–10, Oct 2007.

[30] KANARIS, I.; KANARIS, K.; HOUVARDAS, I. ; STAMATATOS, E.. **Words versus character n-grams for anti-spam filtering**. International Journal on Artificial Intelligence Tools, 16(06):1047–1067, 2007.

[31] BRANTS, T.. **Tnt: A statistical part-of-speech tagger**. In: PROCEEDINGS OF THE SIXTH CONFERENCE ON APPLIED NATURAL LANGUAGE PROCESSING, ANLC '00, p. 224–231, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[32] GOLDBERG, Y.; ELHADAD, M.. **Learning sparser perceptron models**. Technical report, Tech. Rep.[Online]. Available: http://www. cs. bgu. ac. il/˜ yoavg/publications, 2011.

[33] ALUÍSIO, S.; PELIZZONI, J.; MARCHI, A. R.; DE OLIVEIRA, L.; MANENTI, R. ; MARQUIAFÁVEL, V.. **An Account of the Challenge of Tagging**

a Reference Corpus for Brazilian Portuguese, p. 110–117. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[34] FONSECA, E. R.; ROSA, J. L. G.. **Mac-morpho revisited: Towards robust part-of-speech tagging**. In: PROCEEDINGS OF THE 9TH BRAZILIAN SYMPOSIUM IN INFORMATION AND HUMAN LANGUAGE TECHNOLOGY, p. 98–107, 2013.

[35] SRIVASTAVA, N.; HINTON, G. E.; KRIZHEVSKY, A.; SUTSKEVER, I. ; SALAKHUTDINOV, R.. **Dropout: a simple way to prevent neural networks from overfitting.** Journal of Machine Learning Research, 15(1):1929–1958, 2014.

[36] WAN, L.; ZEILER, M.; ZHANG, S.; CUN, Y. L. ; FERGUS, R.. **Regularization of neural networks using dropconnect**. In: PROCEEDINGS OF THE 30TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING (ICML-13), p. 1058–1066, 2013.