Pontifícia Universidade Católica
do Rio de Janeiro

Cristian Enrique Muñoz Villalobos

Redes Neurais de Memória Aumentada para Extração de Relações em Textos

Tese de Doutorado

Tese apresentada como requisito parcial para obtenção do grau de Doutor pelo Programa de Pós–graduação em Engenharia Elétrica do Departamento de Engenharia Elétrica da PUC-Rio.

Orientador : Prof. Ricardo Tanscheit
Co-orientador: Prof. Leonardo Alfredo Forero Mendoza

Rio de Janeiro
Outubro de 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Cristian Enrique Muñoz Villalobos

Redes Neurais de Memória Aumentada para Extração de Relações em Textos

Tese apresentada como requisito parcial para obtenção do grau de Doutor pelo Programa de Pós–graduação da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

Prof. Ricardo Tanscheit
Orientador
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Leonardo Alfredo Forero Mendoza
Co-orientador
Universidade do Estado do Rio de Janeiro – UERJ

Prof. Karla Tereza Figueiredo Leite
Universidade do Estado do Rio de Janeiro – UERJ

Prof. Bianca Zadrozny
IBM Research Brasil – IBM

Prof. Wouter Caarls
Departamento de Engenharia de Elétrica – PUC-Rio

Prof. Maria Claudia de Freitas
Departamento de Letras – PUC-Rio

Prof. Douglas Motas Dias
Universidade do Estado do Rio de Janeiro – UERJ

Prof. Jorge Luís Machado Do Amaral
Universidade do Estado do Rio de Janeiro – UERJ

Rio de Janeiro, 11 de Outubro de 2019

Cristian Enrique Muñoz Villalobos

Graduou-se em Engenharia Mecatrônica pela Universidade Nacional de Engenharia (Lima, Perú). Fez mestrado no Departamento de Engenharia Elétrica da PUC-Rio, especializando-se na área de métodos de apoio à decisão, computação distribuída. Desenvolveu diversos tipos de modelos de inteligência artificial massivamente paralela, especialmente modelos de Deep Learning.

Ficha Catalográfica

Muñoz Villalobos, Cristian Enrique

Redes Neurais de Memória Aumentada para Extração de Relações em Textos / Cristian Enrique Muñoz Villalobos; orientador: Ricardo Tanscheit; co-orientador: Leonardo Alfredo Forero Mendoza. – 2019.

v., 117 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica.

Inclui bibliografia

1. Indução de Programas;. 2. Aprendizado Profundo;. 3. Processamento de Linguagem Natural;. 4. Modelo Transformer;. 5. Extração de Relações;. I. Tanscheit, Ricardo. II. Forero Mendoza, Leonardo Alfredo. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. IV. Título.

CDD: 621.3

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Agradeço também ao meu orientador Prof. Ricardo Tanscheit pelo apoio, simpatia de sempre e incentivo para a realização deste trabalho. Ao meu co-orientador Prof. Leonardo Mendoza pela preocupação, dedicação e auxílio.

Ao Prof. Marco Aurélio e meus competentes colegas do laboratório ICA que ajudaram a criar um ambiente agradável e estimulante para trabalhar durante os quatro anos de pesquisa. Ao CNPq e à PUC-Rio pelos auxílios e estrutura concedida, sem os quais este trabalho não poderia ter sido realizado.

Os meus pais pela formação baseada em valores, disciplina e amor. Seu incondicional apoio para eu continuar, inclusive nos momentos mais difíceis, permitira-me chegar a esta sublime etapa minha vida. A minha esposa pelos conselhos e a compreensão durante os anos de minha pesquisa. Também a minha família, em geral, que sempre me deu força e confiança para cumprir meu objetivo.

Resumo

Muñoz Villalobos, Cristian Enrique; Tanscheit, Ricardo; Forero Mendoza, Leonardo Alfredo. **Redes Neurais de Memória Aumentada para Extração de Relações em Textos**. Rio de Janeiro, 2019. 117p. Tese de Doutorado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

A crescente disponibilidade de grandes corpora – conjunto de corpus de texto – cria a expectativa de sintetizar, organizar e inferir uma quantidade de informação sem precedentes. A extração de conhecimento automática (ECA), área que está no cerne do Processamento de Linguagem Natural (PLN) e da Inteligência Artificial (IA), aponta para o uso de técnicas de aquisição de conhecimento estruturado a partir de dados não estruturados, como os documentos de texto. A ECA compreende essencialmente duas tarefas: o reconhecimento de entidades nomeadas (REN) ou objetos do mundo real, e a extração de relações (ER). Recentemente, as soluções propostas para essas tarefas, em sua maioria, são modelos de aprendizado profundo (AP). Atualmente, os modelos conseguem extrair com boa acurácia informação existente em texto do tamanho de um resumo. Neste trabalho, aborda-se a criação de mecanismos e estruturas de AP que permitam a expansão da capacidade de armazenamento de informação de forma a reconhecer com uma alta eficiência longas dependências entre entidades. Este trabalho tem como objetivo o desenvolvimento, implementação e avaliação de técnicas de AP para aplicações de PLN como o RCE e a ER a partir de um documento de texto. Portanto, exploram-se modelos baseados em estruturas *Transformer*, que otimizam o processamento de sequências, juntamente com mecanismos que se assemelham ao controle de bancos de memória, com a finalidade de incrementar o nível de raciocínio destes modelos. O treinamento dos modelos parte de um conjunto de textos rotulados – anotações – indicando a presença de tipos de entidades ou relações que existem entre elas. O modelo recebe como entrada um texto e deve aprender a reconhecer as entidades e as relações lá contidas. Os resultados obtidos demonstram efetividade dos modelos propostos quando comparados aos baseados em redes neurais já existentes.

Palavras-chave

Indução de Programas; Aprendizado Profundo; Processamento de Linguagem Natural; Modelo Transformer; Extração de Relações;

Abstract

Muñoz Villalobos, Cristian Enrique; Tanscheit, Ricardo (Advisor); Forero Mendoza, Leonardo Alfredo (Co-Advisor.) **Memory-Augmented Neural Networks for Relation Extraction from Text**. Rio de Janeiro, 2019. 117p. Tese de doutorado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

The increasing availability of large corpora – a set of text corpus – creates the expectation of synthesizing, organizing and inferring an unprecedented amount of information. Automatic Knowledge Extraction (AKE), an area that is at the heart of Natural Language Processing (NLP) and Artificial Intelligence (AI), aims at using structured knowledge acquisition techniques from unstructured data such as text documents. In essence, AKE is comprised of two tasks: Named Entity Recognition and Classification (NERC) and Relation Extraction (RE). Recently, the solutions proposed for these tasks are mostly deep learning (DL) approaches. Today, models can accurately extract relation information between entities in different sentences. This work deals with the creation of DL mechanisms that allow increasing the extracted information storage capacity in order to recognize more complex patterns. This work aims to develop, implement and evaluate DL techniques for NLP applications such as NERC and RE from a raw text. Therefore, models based on Transformer structures, that optimize sequence processing, are explored together with mechanisms based on memory networks, in order to increase the reasoning capacity of these models. The training dataset is based on a set of labeled texts – annotations – indicating the presence of entity types or relationships between them. The model receives text as its input and must learn to recognize the entities and relationships contained therein. Results show the effectiveness of the proposed models when compared to those based on existing neural networks approaches.

Keywords

Program Induction; Deep Learning; Natural Language Processing; Transformer; Relation Extraction;

Sumário

Lista de figuras

Lista de tabelas

Lista de Abreviaturas

AP – Aprendizado Profundo
BRAN – *Bi-affine Relation Extraction Networks*
CDP – Condição de Parada Dinâmica
CPU – *Central Process Unit*
CTD – Comparative Toxicogenomics Database
DL – *Deep Learning*
DNC – *Differentiable Neural Computer*
EI – Extração de Informação
ER – Extração de Relações
ECA – Extração de Conhecimento Automatico
GC – Grafos de Conhecimento
GRU – *Gated Recurrent Unit*
IA – Inteligência Artificial
LSTM – *Long Short Term Memory*
MCAM – Módulo de Controle de Acesso a Memória
MAM – Mecanismo de Acesso a Memória
NTM – *Neural Turing Machine*
PLN – Processamento de Linguagem Natural
RD – Representação Distribuída
REN – Reconhecimento de Entidades Nomeadas
RL – Representação Localizada
RNC – Rede Neural Convolutiva
RNR – Rede Neural Recorrente

1
Introdução

1.1
Motivação

Na atualidade,empresas armazenam uma vasta quantidade de dados em repositórios, comumente chamados de *Data Lake*, projetados para a captura, refinamento, arquivamento e exploração de dados brutos [1]. Estes repositórios são compostos principalmente por dados não estruturados. As ferramentas para análise de dados estruturados encontram-se em uma fase madura, porém as ferramentas para análise de mineração de dados não estruturados ainda estão em uma etapa inicial e em desenvolvimento [2]. Os dados não estruturados compõem mais de oitenta por cento dos dados nas empresas [3] e sem as ferramentas adequadas para analisá-los de forma automática deixa-se de aproveitar grandes quantidades de valiosas informações que ajudariam a aumentar o *know-how* das organizações, como, por exemplo, na tomada de decisões.

O Processamento de Linguagem Natural (PLN) é o processamento automático de texto escrito de forma a colocá-lo em um nível que permita ao computador entendê-lo [4]. O PLN orientado para o aprendizado de máquina visa a imitar a inteligência humana através da observação do ser humano ao realizar uma determinada tarefa repetidamente e, a partir de um conjunto de dados, construir um modelo para aprender a replicar tal tarefa. De fato, este esquema de aprendizagem tem sido responsável por grande parte do progresso no processamento de linguagem natural nas duas primeiras décadas do novo milênio. Modelos de redes neurais tornaram-se as ferramentas mais eficazes para uma série de tarefas de PLN, incluindo tradução [5], análise de sentimento [6] e geração de texto [7]. Estas tarefas amadureceram ao ponto de serem úteis tanto para a parte acadêmica, quanto para a indústria.

Umas das tarefas mais populares na área de PLN é a extração de informação (EI) a partir de textos. Segundo Maynard [8], a tarefa de extração de informação consiste tipicamente na sequência de três subtarefas: Pré-processamento linguístico, Reconhecimento e Classificação de Entidades (de forma manual ou por meio de aprendizagem) e Extração de Relações (ER). Anteriormente, as ER eram realizadas por sistemas baseados em regras, também chamados siste-

mas baseados em "engenharia do conhecimento linguístico" [9]. Estes modelos são formados por regras definidas manualmente por especialistas. Uma desvantagem destes modelos é que as regras aprendidas perdem capacidade de generalização para novas relações. Os métodos de aprendizado de máquina permitiam trocar o rigor das regras pelo aprendizado automático, a partir de anotações de entidades e relações dentro do documento. Porém, a criação destas anotações é um trabalho custoso para o especialista. Algumas abordagens tentam reduzir este custo, tal como a criação de anotações de forma automática [10] ou aumentar o conjunto de dados utilizando bases de conhecimento já existentes [11], [12].

Os modelos atuais para ER (de redes neurais, em sua maioria) visam a extrair relações em nível de documento, ou seja, relações que foram definidas através do contexto em várias sentenças. As abordagens para este enfoque melhoram o desempenho dos modelos quando comparadas ao enfoque tradicional, que observa uma única anotação para um único par de entidades em uma frase [11]. No entanto, a criação de um modelo que utilize corpus de textos maiores como entrada, incrementa o tempo de execução e consumo de memória. Assim, a complexidade do problema cresce, visto que o modelo deve aprender a capturar dependências em longas sequências.

Os modelos baseados em aprendizagem profunda (*Deep learning* - DL) são estruturas de redes neurais de múltiplas camadas, especializadas na aprendizagem de representações e resultado do aprimoramento constante das redes neurais ao longo dos anos. Oferecem todo um conjunto de neurônios especializados em detectar diferentes tipos de padrões e têm sido utilizados para resolver diversos problemas em áreas como visão computacional, reconhecimento da fala e processamento de linguagem natural [13]. Os modelos de DL baseados em aprendizado de sequências são utilizados principalmente para resolver problemas de PLN. Estes modelos têm se desenvolvido ao ponto de criar uma nova geração de redes neurais capazes de "observar" e "memorizar" representações de conhecimentos de uma forma mais sofisticada e até intuitiva para o cientista de dados.

Em 2014, D. Bahdanau [14] propôs um novo enfoque para tratar longas sequências em modelos de redes recorrentes: o módulo *Attention*. Este módulo permite à rede concentrar-se em partes relevantes da sequência e ignorar outras para prever a saída. Este enfoque mudou o estado da arte em aprendizado de sequências e foi estendido a vários tipos de mecanismos especializados em processamento sequencial. Um destes mecanismos é o *Transformer* [15], um módulo essencialmente formado por estruturas *Attention* capazes de processar longas sequências em um número de iterações muito menor quando comparado

aos tradicionais modelos recorrentes. Os modelos *Transformer* estabelecem graus de "atenção" entre os elementos da sequência com base em uma função de compatibilidade.

Outro mecanismo que estende o módulo *Attention* são as redes com memória. Segundo Graves [16], o processamento da memória pode ser considerado como um módulo *Attention* ao longo do tempo. Esta habilidade permite à rede neural "selecionar" a informação relevante que deve ser tratada pela memória. As redes com memória permitem a criação de uma geração de modelos de indução de programas baseados em aprendizado profundo.

A indução de programas consiste na criação automática de algoritmos a partir de um conjunto de exemplos. Inicialmente, os algoritmos para criação automática de programas eram baseados apenas em métodos evolutivos [17],[18]. Em 2014, Graves [19] propôs um modelo para aprendizagem de algoritmos baseado em exemplos, nos quais as entradas e saídas são sequências (arbitrariamente longas) de elementos que pertencem a um vocabulário. Este modelo, chamado Neural Turing Machine (NTM), é uma rede neural com uma memória externa que possibilita a realização de processos de leitura e escrita. A rede neural comporta-se, então, como uma unidade de processamento central (CPU) que executa instruções e controla o acesso a uma memória. As instruções de um programa, aprendidas via método do gradiente descendente, são representadas de forma implícita pelos pesos da rede.

Vários outros modelos baseados em aprendizado profundo para a criação automática de algoritmos foram propostos [20][21][22]. Na sua maioria, estes modelos apresentaram maior capacidade para perpetuar informação em memória e melhor generalização em comparação com as redes recorrentes tradicionais, como a Long Short Term Memory (LSTM) ou Gate Recurrent Unit (GRU). Neste contexto, o modelo *Differentiable Neural Computer* (DNC), a segunda versão do NTM, tem apresentado resultados promissores em tarefas como sistemas pergunta-resposta e aprendizado por reforço [22]. Estas novas estratégias prometem levar a inteligência artificial e o processamento de linguagem para o caminho de raciocinar e compreender a natureza dos dados de forma automática.

1.2 Objetivos

Com base no exposto, o principal objetivo deste trabalho é a modelagem, desenvolvimento, implementação e avaliação de uma arquitetura de rede neural com memória externa capaz de raciocinar, armazenar e reutilizar características referentes às inter-relações entre os elementos da sequência de

dados. Esta abordagem é analisada em estudos de caso de PLN, onde é comum encontrar tarefas envolvendo longas sequências. Os casos abordados analisam tarefas de Reconhecimento de Entidades Nomeadas (REN) e Extração de Relações (ER) em documentos de texto.

Os modelos propostos neste trabalho são sistemas que integram a capacidade de processamento sequencial dos modelos *Transformer* e módulos de controle de memória externa. O módulo de controle de memória externa do modelo DNC permite realizar um raciocínio de maior complexidade sobre os dados, mas está limitado pela inerente sequencialidade do modelo. Assim, a criação de um modelo híbrido permite explorar as capacidades das duas arquiteturas.

Pode-se definir como objetivos secundários deste trabalho:

1. Estudo dos conceitos de PLN para reconhecimento de entidades e extração de relações, como também abordagens existentes para a extração de conhecimento a partir de documentos de texto.

2. Análise da arquitetura *Transformer* e modelos relevantes de redes neurais com memória em tarefas de PLN.

3. Modelagem de um módulo *Attention* que permita utilizar estratégias de raciocínio temporal e manipulação de memória em sua arquitetura.

4. Modelagem de um módulo de controle de acesso à memória, baseado no DNC, que torne computacionalmente viável o processamento paralelo de sequências.

5. Implementação de uma arquitetura que permita uma rápida convergência do aprendizado e apresente alta escalabilidade.

1.3 Contribuições

O trabalho prevê as seguintes contribuições:

1. Modelo Híbrido - proposta e desenvolvimento de um modelo híbrido baseado no mecanismo *Transformer* e módulos de controle de acesso à memoria externa que permitam o armazenamento ou raciocínio antes de inter-relacionar os elementos da sequência.

2. Paralelismo do mecanismo de acesso à memória - simplificação e implementação da versão paralela do mecanismo de endereçamento dinâmico, substituindo operações de alto custo computacional.

3. Comparação do Desempenho - entre a metodologia proposta e as metodologias de PLN propostas na literatura.

4. Rápida Convergência - redução drástica no número de iterações necessárias para chegar à convergência e comparação com o modelo base.

5. Escalabilidade - a complexidade computacional do modelo é $O(n.i^2/p)$, ou seja, linear com respeito à dimensionalidade do número de exemplos (n) e quadrático com respeito ao tamanho da sequência (i). Mesmo assim, todas as operações são independentes, o que permite utilizar um alto grau de paralelismo (p).

1.4 Organização do trabalho

O restante deste trabalho é dividido em quatro capítulos adicionais. No Capítulo 2, apresenta-se um resumo dos fundamentos teóricos necessários para a compreensão do trabalho; abordam-se recuperação de informação, extração de relações, aprendizagem de representações e indução de programas. No Capítulo 3, o modelo proposto é apresentado em detalhe, abordando a integração de um controlador de acesso à memória em uma estrutura *Transformer*, o paralelismo e simplificação do mecanismo de acesso à memória e como os processos de REN e ER são encadeados em uma estrutura que permite aproveitar a experiência de aprendizado de ambos. No Capítulo 4 são apresentados os estudos de casos para conjuntos de dados *benchmark* como BioCreative V e SemEval 2018. Finalmente, o Capítulo 5 apresenta discussões e conclusões sobre o modelo e os resultados obtidos, além de apontar direções para trabalhos futuros.

2
Fundamentos

2.1
Extração de Informação e Processamento de Linguagem Natural

A extração de informação (EI) remonta sua história a pelo menos três décadas na quais diferentes abordagens têm sido desenvolvidas. Atualmente, há grande interesse no desenvolvimento desta tecnologia, especialmente em tarefas em que é necessário localizar informação de forma precisa em documentos de texto. Por exemplo, considerem-se os seguintes casos:

- Uma empresa quer verificar a opinião geral sobre seu novo produto em blogs de *websites*.
- Um grupo de pesquisa biomédica está investigando um novo tratamento e deseja conhecer todas as formas possíveis pelas quais um grupo específico de proteínas pode interagir com outras proteínas e quais são os resultados dessas interações. Para isto, existem milhares de artigos, documentos de conferências e relatórios técnicos para analisar.
- Uma agência ambiental necessita mapear a evolução da vida submarina em uma região geográfica a partir de milhares de vídeos coletados por veículos submarinos operados remotamente – ROV.

Estes exemplos apresentam algumas características comuns:

1. A necessidade de solicitar informação.
2. A resposta à solicitação está usualmente presente em fontes de dados não estruturados como textos e imagens.
3. É impossível para os seres humanos processar todos os dados tendo em vista a sua grande quantidade.
4. Os computadores não são capazes de consultar diretamente a informação, já que ela não está estruturada (ex. base de dados).

A extração de informação é uma subárea da inteligência artificial que tenta resolver tal tipo de problema. As definições tradicionais de EI consideravam o texto como única fonte de informação não estruturada [23] [24].

Definições mais recentes como [25] e [28] estendem esta definição para outras fontes não estruturadas. Assim, uma definição mais adequada de EI é a proposta por Marie Francien [25]:

> *"... a EI geralmente é definida como o processo de estruturar e combinar seletivamente dados que estão declarados explicitamente ou implicitamente em uma ou mais fontes não estruturadas."*

Esse processo envolve a classificação semântica de algumas partes da informação, o que é considerado uma forma simples de entendimento dos dados. A realização da tarefa através de textos ocorre por meio do Processamento de Linguagem Natural (PLN). O PLN é um campo da IA que trata do entendimento da linguagem humana ou natural por máquinas (diferenciando-se de linguagens de programação ou de máquinas). Isto acontece mediante a estruturação de fontes não estruturadas, conforme mostra o exemplo apresentado na Figura 2.1. Para cada texto não estruturado, a informação extraída pode ser colocada em formato estruturado. Esta informação pode se dar de forma explicita (1 e 2) ou implícita (3).

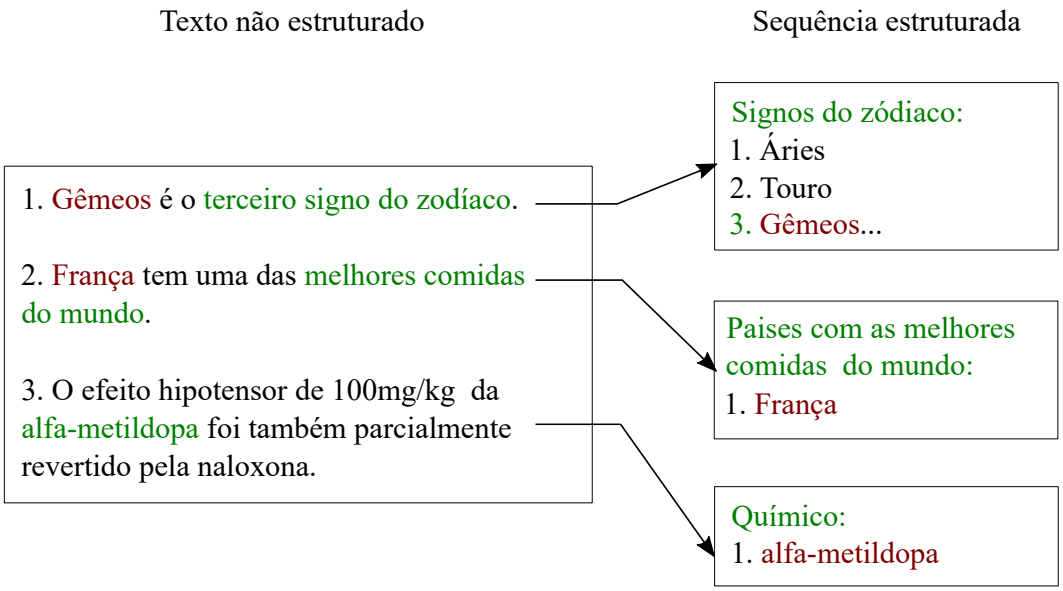


Figura 2.1: O PLN permite a transformação de textos não estruturados em formatos estruturados. Assim, o computador realiza consultas ou *queries* de algum tipo de evento.

A linguagem natural pode ser expressa de forma escrita e falada. Alguns sistemas de PLN são: *chatbots*, que lidam com solicitações de atendimento ao cliente, autocorretor ortográfico em computadores ou celulares, assistentes virtuais (AVs) como Google Assistant, Cortana, Apple Siri. Por exemplo, quando perguntamos para um AV "pode me indicar um bom restaurante peruano por perto?", o assistente executa várias tarefas de PLN:

– Primeiramente o AV deve converter o enunciado em texto (*speech-to-text*).

– Em seguida, o sistema deve entender a semântica da pergunta (o usuário procura um bom restaurante com cozinha peruana) e realiza a *query* necessária (cozinha = peruana, *rating* = 4 - 5 *stars*, distância $<$ 5 km)

– Então, o assistente deve procurar restaurantes filtrando por localização, cozinha, e em seguida ordenar os restaurantes por *rating*.

– Finalmente, quando o usuário está no restaurante, o assistente pode ajudar a traduzir nomes de pratos de comida peruana para o português.

Este exemplo mostra como o PLN, em rápida evolução, começa a formar parte de nossa vida diária. Este campo de pesquisa apresenta problemáticas desafiadoras, dado que palavras e semânticas apresentam relações não lineares de alta complexidade. Também é necessário capturar esta informação de forma numérica que seja robusta como representação. Para aumentar ainda mais a dificuldade, cada linguagem tem sua própria gramática, sintaxe e vocabulário.

Resumindo, pode-se afirmar que a EI concentra-se na estruturação dos dados a partir de fontes não estruturadas (imagens, vídeos, textos, etc.) e o PLN trata dos métodos que permitem às máquinas entender a linguagem humana (escrita e falada).

Para a extração de informação a partir de fontes como artigos científicos, relatórios do governo, blogs on-line, registros médicos etc. existem vários tipos de informação que são de interesse. Por exemplo, os nomes próprios, também chamados de entidades nomeadas, são componentes-chave em um documento de texto. Outros elementos, como algumas expressões temporais, também são considerados entidades. Estas são conectadas por relações, as quais, por sua vez, podem se relacionar criando relações mais complexas chamadas eventos.

Este trabalho aborda a criação de modelos para o reconhecimento de entidades nomeadas (REN) e a extração de relações que ocorrem entre entidades (ER). Uma exemplificação dos tipos de informação extraídos a partir das tarefas REN e ER é apresentada na Figura 2.2. A frase foi extraída e traduzida da base de dados pertencente a área médica, BioCreative V [1].

[1] A base de dados foi descarregada do repositório auxiliar: `https://github.com/patverga/bran/tree/master/data/cdr`

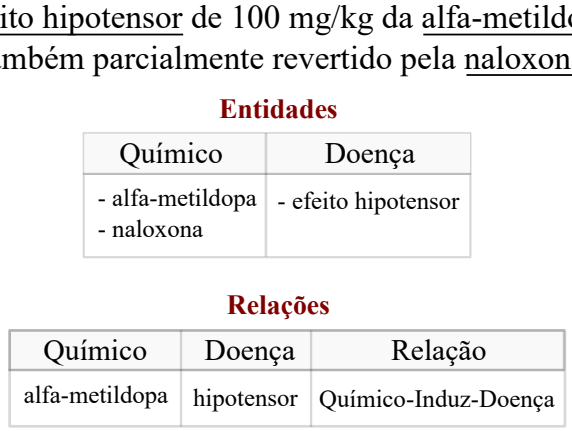


Figura 2.2: Tarefas REN e ER.

Na frase, podem ser identificadas as entidades *alfa-metildopa* e *naloxona* como "Químico" e a entidade *hipotensor* como "Doença". Após o reconhecimento de entidades, a relação "Químico-Induz-Doença" pode ser extraída a partir das entidades *alfa-metildopa* e *hipotensor*. Esta informação encontra-se de forma implícita no texto e a EI a estrutura de forma automática.

2.1.1 Pré-processamento de Texto

As tarefas de pré-processamento são realizadas com a finalidade de colocar o texto em um formato capaz de ser processado posteriormente pelo computador. As transformações ajudam a reduzir o tamanho do espaço de características do texto, removendo parte desnecessárias. Segundo [26], exitem três elementos principais em tarefas de pré-processamento de texto:

1. Tokenização: dada uma sequência de caracteres e uma unidade de documento definida, a tokenização consiste na divisão desta sequência em partes, denominadas tokens [27]. Neste processo pode-se apagar certos caracteres, como pontuação,por exemplo:

   Entrada: Gêmeos é o terceiro signo do zodíaco.
   Saída: Gêmeos | é | o | terceiro | signo | do | zodíaco

   Segundo Manning [27], token é uma sequência de caracteres, em um documento especifico, agrupados como uma unidade semântica útil para processamento. Esta definição é útil para diferenciar token da definição de "palavra", que é considerada a unidade da língua escrita.

2. Normalização: geralmente, refere-se a uma serie de tarefas destinadas a colocar o texto em igualdade de condições. Algumas destas tarefas são:

- Converter todas as palavras e maiúsculas ou minúsculas.
- Converter números em palavras ou remover números.
- Remover pontos, acentos ou outros diacríticos.
- Remover espaços em branco.
- Expandir abreviações.
- Remover *stop words* ou palavras particulares.

Nem todas as tarefas nessa alista devem ser realizadas; isto depende do domínio do texto e do processamento que será realizado posteriormente. Considere-se, no exemplo anterior, que o texto será convertido para minúsculas e que sejam removidas as *stop words* (palavras que podem ser consideradas irrelevantes para a análise, como o, a, em, etc.):

tokens:	Gêmeos é o terceiro signo do zodíaco
minúscula:	gêmeos é o terceiro signo do zodíaco
stop words:	gêmeos terceiro signo zodíaco

3. Remoção de ruído: consiste em remover caracteres ou partes do texto que podem interferir com a análise. Este passo é muito importante e dependente do domínio. Por exemplo, em *tweets*, pode-se considerar como ruído todos os caracteres especiais com exceção de *hashtags*, dado que este normalmente simboliza conceitos que caracterizam o *tweet*.

   Esta tarefa requer a observação dos dados de forma a definir regras que auxiliem na eliminação do ruído. Por exemplo, a Tabela 2.1 apresenta como caracteres especiais ou pontuações impedem o correta interpretação da palavra “problema”.

Palavra Original	Remoção de Ruído
...problema...	problema
problema<	problema
1.problema	problema
problema!	problema

Tabela 2.1: Exemplo de remoção de ruído para a palavra “problema”.

A remoção de ruído evita que os tokens que contêm a palavra “problema” – “...problema...”, “problema<”, “1.problema”, “problema!” – sejam considerados tokens diferentes.

Outro pré-processamento em tarefas de PLN é o tratamento da hiperonímia: relação de significado entre uma palavra no sentido mais abrangente e

as suas ramificações. Esta relação pode prejudicar o desempenho do modelos. Por exemplo, na tarefa REN, a frase "tobacco causes **cancer**" pode ser considerada falsa se a anotação da entidade no documento é "**lung cancer**" e não só "**cancer**". Para isto utiliza-se um vocabulario de hierarquia especifico do domínio.

2.1.2 Reconhecimento de Entidades Nomeadas (REN)

O primeiro passo para a maioria das tarefas de EI é reconhecer e classificar todas as entidades nomeadas contidas no texto (REN). As entidade nomeadas são expressões linguísticas que podem aparecer em várias formas, incluindo nomes, pronomes ("ele", "ela", etc.) e nominais (substantivos, frases nominais, etc). Existem várias abordagens para a resolução do problema REN, as três principais são: sistemas baseados em regras, sistemas baseados em aprendizado de máquina e abordagens híbridas. Este trabalho utiliza um abordagem baseada em modelos de aprendizado de máquina, especificamente de aprendizado profundo. Para a avaliação desta tarefa, existem várias conferências dedicadas à sua avaliação. As mais populares para a língua inglesa são *Text Analysis Conference* (TAC)[1], *BioCreative Challenge Evaluation Workshop*[2] e *International Workshop on Semantic Evaluation* (SemEval)[3]. Para a língua portuguesa, tem-se a *Avaliação de Sistemas de Reconhecimento de Entidades Mencionadas* (HAREM)[4].

A entrada para estes modelos é uma sequência de tokens e a saída é um conjunto de "anotações", ou seja, um grupo de caracteres selecionados através de trechos do texto de entrada. Um exemplo de um modelo genérico para REN é apresentado na Figura 2.3:

[1] `https://tac.nist.gov/`
[2] `https://biocreative.bioinformatics.udel.edu/`
[3] `http://alt.qcri.org/semeval2018/`
[4] `https://www.linguateca.pt/aval_conjunta/HAREM/harem_ing.html`

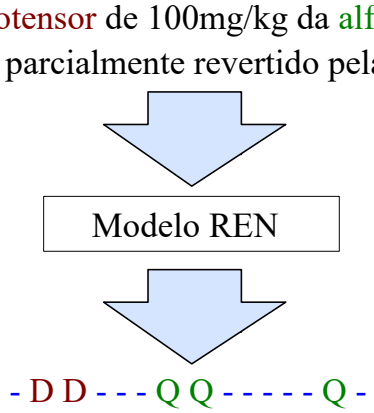


Figura 2.3: Reconhecimento de Entidades Nomeadas: A saída gerada pelo modelo é uma sequência indicando a localização e o tipo de entidade no texto.

Para a criação do conjunto de dados, os formatos de anotações são um fator importante que facilita o pré-processamento dos textos. Inicialmente, utilizava-se o esquema de representação no nível da menção *Standard Generalized Markup Language* (SGML), que anotava entidades em um único bloco [29]. Usando o exemplo da Figura 2.2:

O efeito <Doença>hipotensor</Doença> de 100 mg/kg da <Químico>alfa-metildopa</Químico> foi também parcialmente revertido pela <Químico>naloxona</Químico>.

Dado que a tarefa REN envolve um problema de aprendizado estatístico, os formatos de "anotações de sequências" tornaram-se o enfoque padrão. Neste formato, a frase completa (sequência de tokens) é rotulada. Neste tipo de anotação, os mais utilizados são BIO e BIOLU [30]. Na representação BIO, (*Beginning-Inside-Outside*) proposta por Ramshaw e Marcus [31], os tokens podem ser subcategorizados como início, dentro e fora da entidade nomeada. Alternativamente, a representação BILOU adiciona a subcategoria última (*Last*) e única (*Unit*) palavra da entidade nomeada. Cada anotação segue o formato 'B-entidade', 'I-entidade', 'O-entidade', 'L-entidade','U-entidade', onde 'entidade' representa a classe de entidade nomeada. Um exemplo deste esquema é apresentado na Tabela 2.2.

Token	BIO	BIOLU
O	O	O
efeito	B-Doença	B-Doença
hipotensor	I-Doença	L-Doença
de	O	O
100	O	O
mg/kg	O	O
da	O	O
alfa	B-Químico	B-Químico
-	I-Químico	I-Químico
metildopa	I-Químico	L-Químico
foi	O	O
também	O	O
parcialmente	O	O
revertido	O	O
pela	O	O
naloxona	B-Químico	U-Químico
.	O	O

Tabela 2.2: Exemplo de formatos de anotação BIO e BIOLU.

Ao selecionar o formato de anotação, o número de classes aprendidas pelo classificador é acrescentado. Por exemplo, no caso da anotação BIO, para a classe “Químico“ agora existe "B-Químico","I-Químico"e "O-Químico", e o mesmo ocorre para o formato de anotação BILOU. Terminado o processo de detecção e classificação de entidades, as entidades nomeadas podem ser utilizadas para propósitos como recuperação de informação [32], sistemas pergunta-resposta [33] e máquinas de tradução [34]. Particularmente no *pipeline* de EI, após a tarefa REN, o nível seguinte é a extração de relações entre entidades.

2.1.3 Extração de Relações (ER)

A tarefa de ER consiste em detectar e classificar relações semânticas que ocorrem entre (pares de) entidades que foram reconhecidas previamente na tarefa de REN [8]. Na literatura, encontra-se uma variedade de tipos de relações. Entretanto, uma relação é considerada relevante de acordo com vários fatores, principalmente no que diz respeito ao tipo de informação que se deseja extrair. Um diagrama geral de um modelo para extração de relações é apresentado na Figura 2.4:

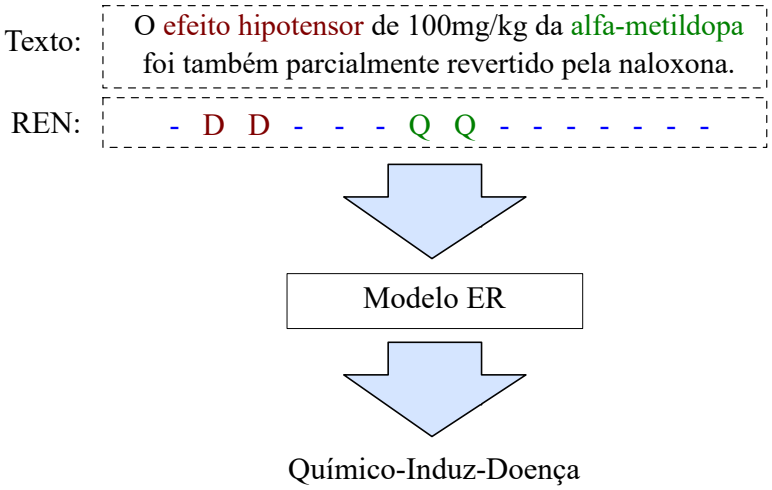


Figura 2.4: Extração de Relações: o modelo recebe o texto e as entidades desejadas para identificar a relação; caso não exista relação, esta é denominada relação nula.

Um problema comum na tarefa de ER acontece quando se trabalha apenas no nível de sentença. Por exemplo, considere-se o seguinte texto:

"O modelo stemming é baseado em métodos de tradução automática estatística. Este usa um stemmer inglês e um pequeno corpus como únicos recursos de treinamento."

No texto, "modelo stemming" e "Este" fazem referência a uma mesma entidade; os modelos de ER que consideram apenas a relação entre pares de entidades dentro de sentença não conseguem estabelecer relações entre sentenças diferentes. Extrair esta informação por separado requer "resolução de correferência" que consiste em agrupar todas as frases nominais (menções) que referenciam a mesma entidade no mundo real. As conexões entre sentenças não podem ser aprendidas automaticamente por modelos que consideram apenas pares de entidades de forma isolada. Também, pode-se intuir que o fato de fazer previsões de forma separada obstrui o aprendizado de várias instâncias. Assim, algumas abordagens de aprendizado de máquina, especificamente aprendizado profundo, consideram operar em nível de documento, processando todas as sentenças que formam o contexto. Estas técnicas de aprendizado serão tema do restante desta seção.

2.2 Inteligência Artificial, Aprendizado de Máquina e Aprendizado Profundo

Ao longo dos anos, vários autores têm definido a Inteligência Artificial (IA) em termos de processos de pensamento, raciocínio ou comportamentos, que podem ser simulados por seres humanos ou estritamente racionais [35][36][37]. Desta premisa, depreende-se que os sistemas de IA exploram algum tipo de inteligência nas tarefas que realizam. As definições de IA não especificam como o sistema adquire essa inteligência. No início, a inteligência era inserida em programas de computador através de regras, definidas por especialistas, que manipulavam o conhecimento. Este esquema pode ser observado na Figura 2.5. Este enfoque, também conhecido como IA simbólica, deu nascimento aos populares sistemas especialistas dos anos 1980s [38].

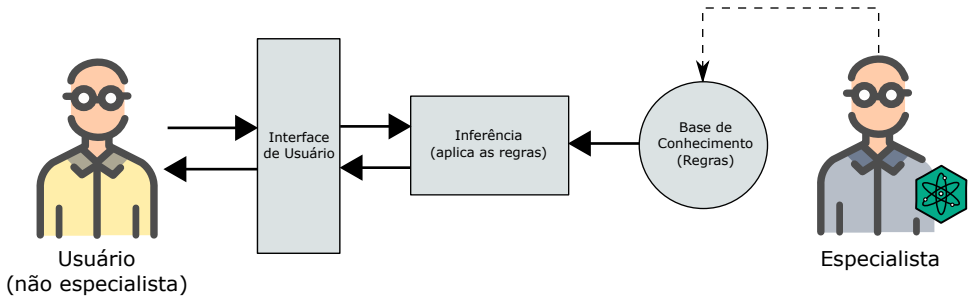


Figura 2.5: Sistemas Especialistas – de forma geral, dividem-se em 2 etapas: base de conhecimento, definido pelo especialista, e inferência, na qual se aplicam regras para um novo fato.

Posteriormente, uma pergunta abriu a porta para um novo paradigma de programação: podem as máquinas aprender automaticamente as regras de um sistema de IA a partir de dados observados? Este novo paradigma é chamado de aprendizado de máquina e envolve a criação de sistemas 'treinados' com muitos exemplos (relativos a uma tarefa), em vez de serem explicitamente programados. Um diagrama geral deste sistema é apresentado na Figura 2.6.

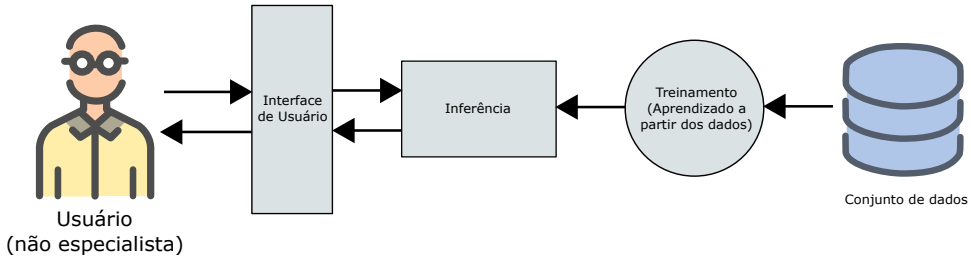


Figura 2.6: Aprendizado de Máquina – o sistema aprende a definir regras de forma automática a partir de um conjunto de dados.

Este enfoque rapidamente se converteu no subcampo mais popular da IA, estando também intimamente relacionado à matemática estatística. O

'treinamento' permite ao aprendizado de máquina definir regras que extraem representações úteis dos dados para explicar uma determinada resposta (a tarefa) [39]. Modelos como máquinas de vetores suporte [40], algoritmos de floresta aleatória *random forest* [41] e redes neurais multi-camadas [42] estão entre os mais conhecidos.

Nas últimas décadas, as aplicações em IA têm abordado problemas como reconhecimento de voz, processamento de imagem e processamento de linguagem natural. Estes problemas criaram a necessidade de interpretação de níveis de maior complexidade do mundo perceptual e de se dispor de sistemas perceptivos capazes de aprender essas interpretações observando seu ambiente. Neste contexto, um novo subcampo no aprendizado de máquina propôs permitir aos computadores aprender da experiência e entender o mundo com base em conceitos hierárquicos. Esta hierarquia permite ao computador aprender e construir conceitos complexos a partir de outros mais simples. Um diagrama deste sistema é apresentado na Figura 2.7

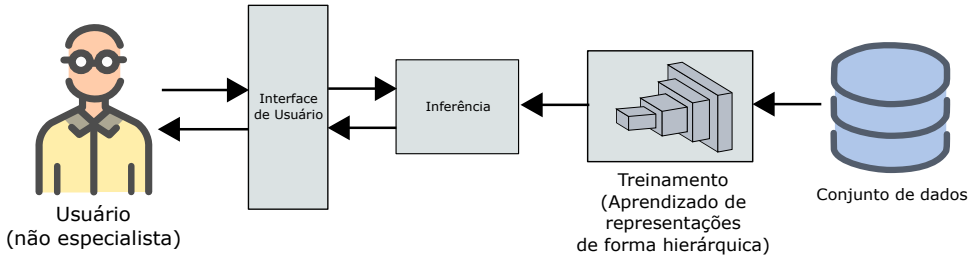


Figura 2.7: Aprendizado Profundo – o aprendizado automático é realizado mediante o aprendizado de representações de forma hierárquica.

Se a construção dos conceitos é representada em um grafo, este é profundo, com muitas camadas. Daí a denominação aprendizado profundo ou *Deep Learning*. Esta perspectiva permite extrair representações que são expressas em função de outras representações mais simples.

Segundo Goodfellow et al. [39], no aprendizado profundo, pode-se dizer que a profundidade do modelo permite ao computador aprender múltiplos passos de um programa de computador. A representação que gera cada camada pode ser considerada como o estado da memória do computador e os neurônios desta camada executam um conjunto de instruções em paralelo. Redes com maiores profundidades podem executar uma maior quantidade de instruções. As instruções sequenciais oferecem um grande potencial dado que, em instruções posteriores, podem se referir a resultados de instruções passadas. Assim, não necessariamente toda a informação codificada pelas ativações das camadas indica fatores de variação que explicam a entrada. As representações também armazenam estados de informação que ajudam a executar o programa

e dar sentido aos dados de entrada. Os estados da informação podem ser análogos a contadores e ponteiros em programas de computador tradicionais.

Finalmente, como resumo desta seção, é importante deixar claro que a inteligência artificial é um campo geral que engloba aprendizado de máquina e aprendizado profundo. O aprendizado profundo é um subcampo do aprendizado de máquina que estrutura de forma hierárquica a extração de representações a partir dos dados de forma a resolver uma tarefa específica.

2.3 Aprendizado de Representações

A dificuldade em muitas tarefas que envolvem processamento de informação reside, principalmente, em como esta é representada. Por exemplo, dividir 216 por 3 usando longas divisões é uma tarefa relativamente direta. Esta tarefa se torna mais difícil se forem empregados números romanos. Em geral, pode-se afirmar que uma boa representação é aquela que subsequentemente tornará mais fácil a tarefa de aprendizado.

Segundo Yoshua Bengio et al. [43], o objetivo do aprendizado de representações é definir estruturas eficientes que extraiam representações relevantes a partir dos dados para resolver uma tarefa em particular. Por exemplo, um perceptron multi-camada não envolve explicitamente, na estrutura, condições para o aprendizado de representações intermediárias. Na seção 2.4. serão descritos alguns tipos de redes neurais apropriados para estruturar formas particulares de representação. Dependendo de como a representação foi obtida, ela pode ser usada para transferir conhecimento a tarefas, onde poucos (ou nenhum) exemplos são fornecidos, mas existe uma tarefa de representação (Zero-Shot learning) [44] [45]. Alternativamente, pode-se melhorar a generalização, aproveitando as informações especificas do domínio contidas nos sinais de treinamento de tarefas relacionadas (Multi-Task learning) [46].

Para muitos problemas de IA, a disponibilidade de dados rotulados para o treinamento é escassa e a sua geração é custosa. Os humanos e animais têm a capacidade de aprender a partir de poucos exemplos rotulados e ainda não se sabe como isto é possível. Uma hipótese é que o cérebro é capaz de realizar um processo de aprendizado não supervisionado ou semi-supervisionado [39]. Assim, o aprendizado de representações é particularmente interessante no plano de prover formas de realizar tais tipos de aprendizado.

O treinamento supervisionado de um conjunto de dados rotulados pertencentes a um conjunto maior de dados não rotulados tende a cair em *overfitting*. O aprendizado semi-supervisionado permite resolver este problema aproveitando o aprendizado com dados não rotulados. Por exemplo: pode-se aprender

boas representações a partir de dados não rotulados e usar estas representações para resolver tarefas de aprendizado supervisionado. Um clássico modelo não supervisionado que se encaixa neste contexto é a Análise de Componentes Principais (PCA), método utilizado para a redução de dimensionalidade. Este método tem servido de motivação para modelos como a rede neural *autoencoder* [47].

Os modelos atuais de aprendizado profundo definem mecanismos especializados em extração de representações e, por esta razão, outros nomes para este campo poderiam ter sido aprendizado de representações por camadas [48] ou aprendizado de representações hierárquicas [49]. Assim, a noção dos conceitos que são próprios do aprendizado de representações pode ser útil para a criação de estruturas profundas.

2.4 Representação Localizada, Representação Distribuída e Representação Distribuída de Palavras

Entender a forma como os indivíduos representam a informação que lhes permite se comportarem de forma sofisticada é uma das pesquisas centrais das ciências cognitivas. O debate, até a atualidade, refere-se à natureza localizada [50] ou distribuída [51] da representação no momento de ser armazenada pelo cérebro. O aprendizado de representações utiliza estas ideias para as suas principais formas de representação: localizada e distribuída. A representação distribuída tem sido particularmente importante na área de PLN, sendo a representação distribuída de palavras ou *word embeddings* a estratégia mais popular na maioria de modelos de aprendizado profundo para PLN.

1. **Representação Localizada (RL):** denominada em inglês *one-hot representation*, ou representação simbólica, se caracteriza por associar a entrada a um único símbolo ou categoria. Se existem n símbolos no dicionário, então existem n detectores de representação; cada um detecta a presença de uma categoria associada. Neste caso, o número total de configurações diferentes do espaço de representações é igual ao número de regiões definido pelo espaço da entrada.

2. **Representação Distribuída (RD):** denominada em inglês *embedding*, a representação distribuída de conceitos (representações compostas de muitos elementos – características – que podem ser separados uns dos outros) é uma das ferramentas mais importantes do aprendizado de representações. Nessa representação, cada conceito é representado por várias características. Cada característica participa da representação de vários conceitos. Pode-se utilizar

n características com k valores para descrever k^n conceitos diferentes. Um caso particular desta representação é o vetor de n características binárias, que pode representar 2^n configurações.

Por exemplo, na Fig. 2.8, representam-se 5 classes – "menina", "bomba", "bebê", "ok", "olhos" – usando (a) RL e (b) RD com 4 características binárias.

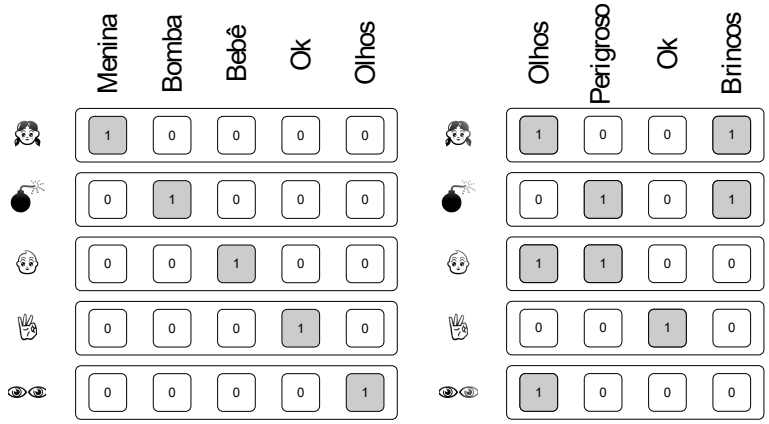


Figura 2.8: Representação localizada e distribuída. (a) RL: cada símbolo é representado por um elemento dentro de um vetor que indexa o vocabulário de símbolos; (b)RD: os elementos do vetor representam características que definem o símbolo (ou conceito).

Usando RL, a classe "bebê" pode ser representada por um vetor de 5 neurônios com um único neurônio ativo que identifica a classe "bebê"; usando RD, identificam-se as características, por meio de vários neurônios, que definem o conceito da classe "bebê" (tem olhos, é perigoso, etc.).

3. **Representação Distribuída de Palavras:** também chamado *Embeddings*, esta representação é baseada no aprendizado de fatores causais subjacentes que explicam os dados. Em PLN, esta estrategia tem sido amplamente utilizada, dado que se pode considerar que palavras em um contexto podem compartilhar informação entre elas.

Esta representação começou a ser comumente utilizada pois os valores do *embedding* podem ser aprendidos a partir de dados não supervisionados. O primeiro modelo desta categoria foi word2vec proposto por [52]. Este modelo apresenta duas versões (BOW e Skip-gram) e aprende relações de vizinhanças de palavras dentro de um contexto. Na Figura 2.9 são apresentadas as duas variantes do modelo. A abordagem BoW realiza o aprendizado tentando prever uma palavra a partir das palavras que se encontram no contexto. O modelo skip-gram tenta prever as palavras do contexto a partir da palavra central.

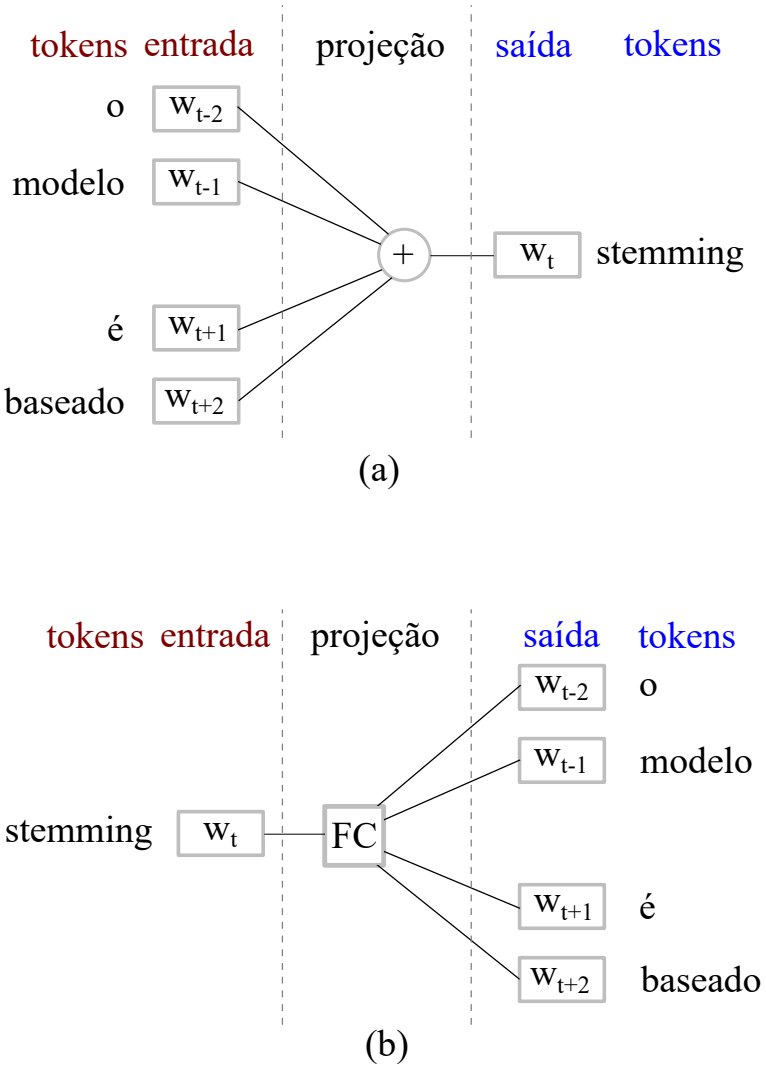
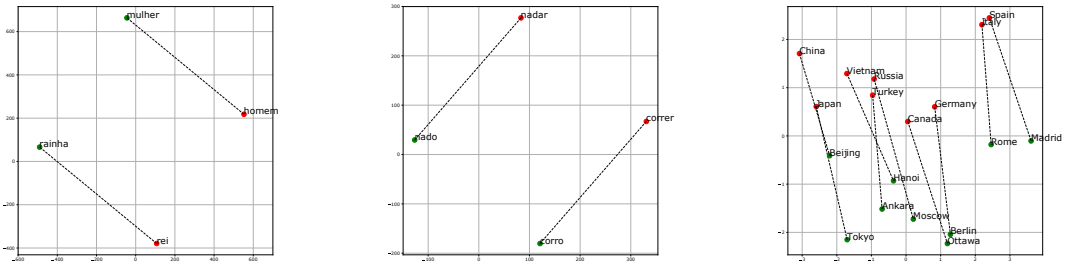


Figura 2.9: Modelos Word2Vec – Bag of Words (a): cada token é representado por um vetor W. A soma dos vetores do contexto deve gerar o vetor W da palavra central; Skip-gram (b): O vetor W da palavra central deve gerar os vetores W do contexto.

Modelos similares [53][54] utilizam esta abordagem para criar vetores de representação (*embeddings*) a partir de um conjunto de textos. Trabalhos atuais indicam que vetores de representação de palavras são o estado da arte em modelos de representação para tarefas de PLN como análise de sentimento [55], extração de informação [56] e similaridalhe entre documentos [57]. Esta mesma representação pode ser utilizado em modelos supervisionados. Neste caso, os vetores de representação aprendem características de contexto que auxiliam na resolução de uma tarefa específica. Na prática, é comum utilizar as duas técnicas em conjunto, inicializando os vetores de representação com o treinamento não supervisionado antes do treinamento supervisionado.

Estes modelos codificam o conteúdo sintático e semântico implicitamente. Assim, a relação entre conceitos ou palavras pode ser simplesmente calculada

como a distância entre suas representações, permitindo analogias lineares em certos casos. Por exemplo, $v(''rei'') - v(''rainha'') \approx v(''homem'') - v(''mulher'')$. Técnicas de redução dimensional, como PCA [58], KPCA [59], ICA [60] e T-SNE [61] permitem visualizar como os *embeddings* codificam a semântica nas posições do espaço vectorial. Por exemplo, na Figura 2.10, apresenta-se a visualização de *embeddings* pré-treinados em português[1] do modelo GloVe [62].



(a) Géneros (b) Verbos (c) País - Capital

Figura 2.10: Vetores de Representação de Palavras – *Word Embeddings*. Os *embeddings* conseguem representar diferentes relações semânticas (masculino-feminino, países-capitais, etc.)

A visualização mostra relações geométricas que capturam as relações semânticas; e podem variar radicalmente dependendo do tipo de documento. Por exemplo, os *embeddings* tem permitido quantificar estereótipos existentes em diferentes épocas e sociedades [63] [64]. Por esse motivo, para o desenvolvimento de novos modelos, é importante identificar e eliminar estas fontes de "bias" de forma a evitar aplicações de inteligência artificial que possam descriminar populações específicas [65].

2.5 Modelos de Aprendizado de Representações

Um dos melhores exemplos de aprendizado de representações é a memória humana. Diferentes áreas de estudo como a neurociência e a biologia pesquisam, do ponto de vista perceptivo, a forma em que se aprendem caraterísticas ou hierarquias de características a partir do ambiente. Por exemplo: Como o córtex visual dos mamíferos na Figura 2.11 consegue perceber? Como a memória aprende os conceitos mais abstratos desde os níveis mais baixos?

[1]Os vetores pré-treinados foram descarregados do site: `http://nilc.icmc.usp.br/embeddings`

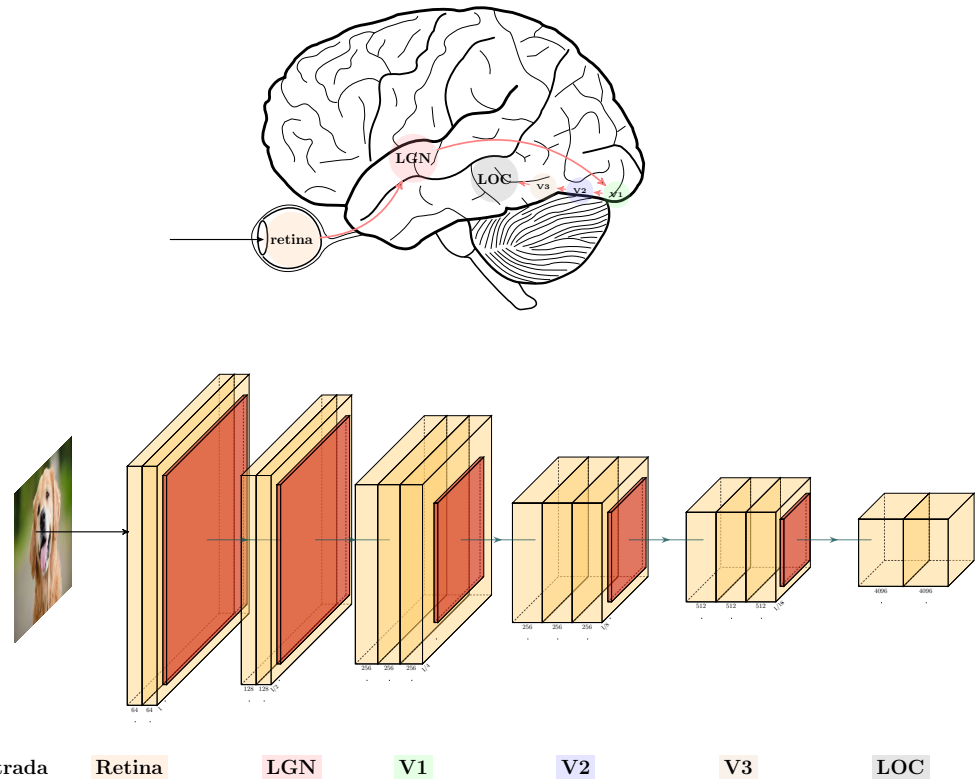


Figura 2.11: Cortex Visual em Mamiferos. A via ventral (de reconhecimento) no córtex visual tem múltiplas etapas de representações intermediárias: Retina-LGN-V1-V2-V3-LOC; cada uma detecta desde bordas (nos níveis primários), formas geométricas, grupos, até os conceitos mais complexos como rostos ou objetos.

Na via ventral no córtex visual existem etapas de representações intermediárias para definir os conceitos mais complexos a partir dos conceitos mais básicos [66]. Da mesma forma, muitos modelos de aprendizado de representações estão projetados para usar uma organização hierárquica de fatores explicativos, ou seja: definir as representações mais abstratas em termos de representações mais simples, formando uma hierarquia.

Outra estratégia importante é a desvinculação dos fatores de variação. Segundo [67], o modelo ideal de extração de características consegue desvincular os fatores de variação, tornando estas características quase que independentes, estatisticamente, entre si. Estas características são aprendidas de forma automática, não havendo necessidade de rótulos para classificá-las internamente. Vários trabalhos [68],[69],[70], [71] têm explorado estes tipos de características que definem os princípios dos modelos de aprendizado profundo.

Um dos objetivos do aprendizado de representações é encontrar um conjunto de estratégias genéricas de regularização (reduzir a variança do modelo) que sejam aplicáveis a uma ampla variedade de tarefas de IA, semelhantemente às tarefas que as pessoas e animais são capazes de resolver.

Bengio et al. (2013) [43] fornecem uma lista de estratégias genéricas de regularização. As mais importantes para este trabalho são:

1. Suavidade: sendo f a função a ser aprendida, considera-se geralmente que $f(x) \approx f(x + dx)$, onde dx é um valor pequeno. Esta suposição permite ao modelo generalizar desde dados de treinamento até pontos próximos no espaço de representações.

2. Linearidade: muitos algoritmos de aprendizado supõem que o relacionamento entre algumas variáveis é linear. Isso permite que o algoritmo faça previsões mesmo muito distantes dos dados observados, mas também pode levar a soluções extremas.

3. Múltiplos fatores explicativos: vários algoritmos de aprendizado de representações são baseados na suposição de que os dados foram gerados por múltiplos fatores explicativos subjacentes, e que a maioria das tarefas pode ser resolvida facilmente, dado o estado de cada um desses fatores.

4. Organização Hierárquica de fatores explicativos: as representações h (ou conceitos) usadas para descrever os dados de entrada x podem ser definidos de forma hierárquica a partir de outras representações. Por conseguinte, os conceitos de maior abstração são definidos em termos de outros menos abstratos.

5. Fatores compartilhados através de tarefas: quando há várias tarefas de aprendizado, correspondentes a diferentes variáveis y_i que compartilham a mesma entrada x. A suposição é que cada y_i está associada a um subconjunto de um conjunto de fatores relevantes (ou representações) h extraídos a partir de x. Dado que existe uma sobreposição entre subconjuntos associada a cada tarefa, aprender toda a probabilidade $P(y_i|x)$ usando a representação intermediária $P(h|x)$ permite aprender e compartilhar fatores explicativos mais generalizáveis entre tarefas.

6. *Manifolds*: concentrar a função de probabilidade em regiões que são conectadas localmente ocupando um volume pequeno. No caso contínuo, estas regiões podem ser aproximadas por *manifolds* de baixa dimensão – de dimensão muita menor do que a dimensão original de onde os dados se encontram. Alguns algoritmos de aprendizado, especialmente *autoencoders*, tentam aprender explicitamente a estrutura *manifold*.

7. Esparsidade: a maioria das características supostamente não devem ser relevantes para descrever a maioria das entradas. Não há necessidade de se usar uma característica que detecta tromba de elefante, por exemplo,

quando se representa a imagem de um gato. Portanto, para qualquer característica que possa ser interpretada como “presente” ou “ausente”, é razoável impor um estado a priori como ausente na maioria de vezes.

Os conceitos de aprendizado de representações integram as várias formas de aprendizado profundo. O perceptron multicamada, redes neurais recorrentes, autoencoders e modelos probabilísticos profundos aprendem e exploram representações.

2.6 Aprendizado Profundo

Nas Seções 2.2, 2.3, 2.4 e 2.5 foi abordada a filosofia do aprendizado profundo e a sua conexão com o aprendizado de representações. Nesta seção será apresentada a estrutura geral dos modelos de aprendizado profundo e uma descrição analítica de como as redes neurais convolutivas e recorrentes conseguem extrair conceitos complexos a partir de dados de treinamento.

As estruturas tradicionais de aprendizado de representações estão divididas em dois processos, como se observa na Fig. 2.12. Anteriormente ao aprendizado profundo, a extração de caraterísticas era um processo manual realizado por um especialista. Esta engenharia requeria um cuidado significativo e um domínio considerável para projetar os dados brutos em uma representação interna adequada para então realizar a classificação.

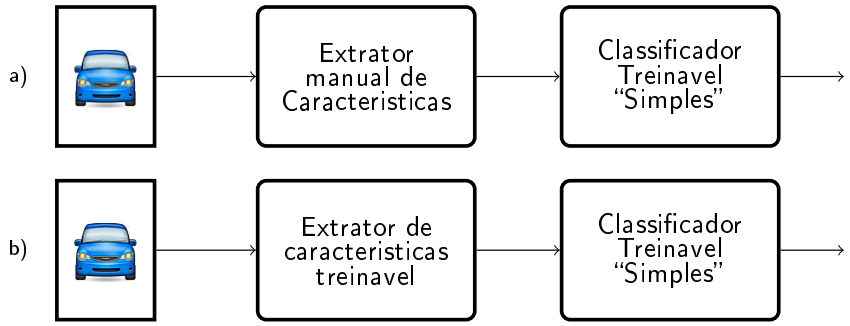


Figura 2.12: Modelos tradicionais e de aprendizado profundo. a) Extração manual de características+ classificador. b) Extractor de características treinável + Classificador treinável.

No aprendizado profundo, há módulos treináveis tanto na etapa de extração de características como na classificação (Fig. 2.12(b)). Geralmente, pode-se dizer que um modelo é considerado profundo se utiliza mais de um nível de representação.

As representações são aprendidas a partir dos dados, por meio de um determinado procedimento de treinamento. Na prática, o método do gradiente

descendente é largamente empregado. Na Fig. 2.13, uma estrutura hierárquica de três níveis de representação é exibida. Na entrada, estão os pixels da imagem (a foto em si). No primeiro extrator de características (nível baixo), tipicamente é detectada a presença ou a ausência de bordas, como também a orientação específica e locação na imagem. O segundo extrator de características (nível médio) extrai os componentes relevantes de um objeto.

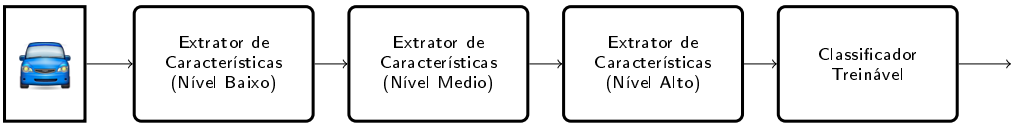


Figura 2.13: Extração hierárquica de características.

Finalmente, o terceiro extrator de características (nível alto) detecta objetos ou a combinação de componentes. Conforme discutido na seção 2.2, as representações de alto nível são mais globais e invariantes. As representações de baixo nível são compartilhadas entre as diferentes representações das camadas posteriores.

Os modelos de aprendizado profundo compreendem um conjunto de neurônios projetados para capturar padrões em contextos específicos – espacial e temporal. Nesta seção são apresentados os principais tipos de neurônios que serão utilizados neste trabalho.

2.6.1 Rede Neural Convolutiva (RNC)

O neurônio convolutivo pode ser descrito a partir de um perceptron. O perceptron apresenta conexões com todos os elementos de entrada. Esta estrutura dificulta a detecção de padrões, devido à procura por interação de todos os elementos. Em vista disso, o neurônio convolutivo apresenta três características:

- Conexões Locais ou Estratégia de Esparsidade: vários neurônios especializam-se em aprender a interação em pequenas regiões da matriz de entrada de forma a ocupar a região completa.
- Compartilhamento de Parâmetros: realiza-se com a finalidade de reduzir a quantidade de parâmetros a ser estimados. Basicamente, ter-se-ia que treinar a quantidade de parâmetros referentes às conexões locais de um único neurônio e compartilhá-los com todo o conjunto de neurônios que mapeiam a entrada. Isto pode ser representado por uma operação de convolução. Por esta razão os parâmetros são chamados de filtro.

– Estrutura Espacial: Três hiperparâmetros definem a sua estrutura espacial: *depth*, *stride* e *zero-padding*. Matricialmente, o número de filtros em uma camada convolutiva é chamado *depth* ou características. O passo utilizado na operação de convolução é chamado *stride*. A operação de convolução modifica a dimensão inicial da matriz de entrada. Em alguns casos esta transformação não é desejada, resultando na adição de linhas e colunas de zeros nas bordas (*zero-padding*) de forma a controlar a dimensão espacial da saída da convolução.

Por exemplo, considere-se a representação $x^{(in)} \in \mathbb{R}^{W \times H \times C_i}$, composta por C_i características de dimensão $W \times H$. Uma camada de C_o neurônios convolutivos formam um kernel $k \in \mathbb{R}^{k_w \times k_h \times C_i \times C_o}$, onde k_w e k_h definem a vizinhança do kernel nos eixos W e H e as C_i características de entrada. A operação de convolução, na Eq. (2-1), gera a representação $x^{(out)} \in \mathbb{R}^{W' \times H' \times C_o}$, onde C_o é o número de neurônios ou novas características detectadas. Esta operação é acompanhado de um e bias $b \in \mathbb{R}^{C_o}$:

$$x^{(out)} = x^{(in)} * k + b, \tag{2-1}$$

A Figura 2.14 apresenta a operação de convolução para $x^{(in)} \in \mathbb{R}^{7 \times 7 \times 1}$ sem considerar padding ($p = 0$) e um kernel $k \in \mathbb{R}^{3 \times 3 \times 1 \times 1}$. Para simplificar a operação, $x^{(in)}$ contém só uma característica ($C_i = 1$) e o kernel k pertence a uma camada com um único neurônio convolutivo ($C_o = 1$).

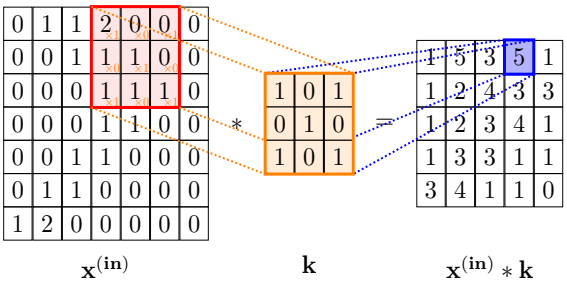


Figura 2.14: Operação de convolução entre a representação $x^{(in)}$ e o kernel k. O cálculo da característica extraída na convolução (elemento azul com valor 5) é resultado do produto escalar entre o kernel k e a região vermelha de $x^{(in)}$.

No exemplo, o *stride* ou passo de uma região (cor laranja) a uma adjacente, é $s = 1$. Para calcular a dimensão da representação de saída de uma camada convolutiva utiliza-se a Eq. (2-2):

$$W' = \frac{W - k_w + 2p}{s} + 1, \qquad H' = \frac{H - k_h + 2p}{s} + 1, \tag{2-2}$$

Assim, $W' = (7-3+2\times 0)/1+1 = 5$, igualmente $H' = 5$, o que pode ser verificado no resultado da convolução na Figura 2.14. Finalmente, para muitas

estrutura de AP, é importante manter as dimensões W e H constantes através de várias camadas. Para isso, considerando $s = 1$, o *Zero-padding* é calculado pela Eq. (2-3):

$$p_{i\in[w,h]} = \frac{k_i - 1}{2}, \tag{2-3}$$

2.6.2 Rede Neurais Recorrentes (RNR)

As RNR são camadas especializadas no processamento de informação sequencial. Considera-se a RNR como uma operação sobre uma sequência de vetores $x_t \in \mathbb{R}^I$, para o instante $t = [1, .., T]$. A RNR possui um estado interno $h_t \in \mathbb{R}^H$, onde H é o número de neurônios atualizado a partir do valor de entrada a cada instante t. A Figura 2.15 apresenta a estrutura básica de uma RNR:

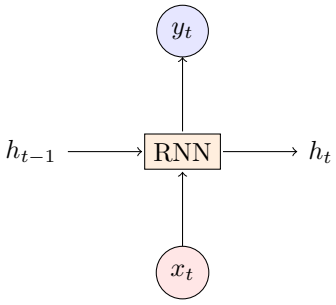


Figura 2.15: Estrutura básica de uma RNR.

Os pesos da RNR são afinados com a finalidade de controlar o comportamento da rede neural por intermédio do estado h_t. A atualização de h_t é realizada pela função f, na Eq. (2-4), e depende de seu estado anterior h_{t-1}, a entrada atual x_t e os parâmetros θ que permanecem fixos durante o processamento de toda a sequência.

$$h_t = f(h_{t-1}, x_t, \theta), \tag{2-4}$$

Uma característica importante da RNR é a completude de Turing [72][73]: para qualquer função computável por uma Máquina de Turing, existe um número finito de RNRs capaz de representá-la. Portanto, existe um número finito de RNRs capaz de implementar qualquer algoritmo. Assim, a saída pode ser lida a partir da RNR após um número de passos que é assintoticamente linear ao número de passos usados pela máquina de Turing e ao tamanho da entrada [39].

Outra característica da RNR é a flexibilidade de sua estrutura. As conexões sequenciais podem apresentar diferentes configurações entrada-saída.

Dependendo da aplicação, a RNR pode apresentar configurações *one-to-many* (O2M), *many-to-one*(M2O), *many-to-many*(M2M). Na Tabela 2.3 são apresentadas algumas aplicações para as configurações citadas:

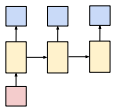
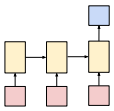
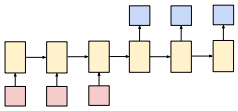
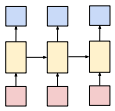
Tipo	(a) O2M	(b) M2O	(c) M2M	(d) M2M
Estrutura				
Exemplos	Legendagem de imagens	Análise de sentimentos	Máquinas de tradução	PoS tagging

Tabela 2.3: Tipos de configuração entrada-saída de uma RNR. Os blocos vermelho, laranja e azul indicam a entrada, a unidade RNR e a saída.

Por exemplo, para a tarefa O2M de legendagem de imagens (Tabela 2.3.a), a rede recebe um único vetor de entrada contendo uma imagem e a RNR processa como saída uma sequência de palavras que descrevem a imagem. Na tarefa M2O de análise de sentimentos (Tabela 2.3.b), a rede recebe uma sequência de palavras e processa um vetor de saída indicando o sentimento da frase. A configuração M2M apresenta duas estruturas básicas; uma delas é a estrutura Encoder-Decoder, cuja configuração codifica a sequência de entrada x no vetor de estado h_{T_x} e, posteriormente, decodifica a sequência de saída y. Esta configuração é amplamente utilizada quando $T_x \neq T_y$ (por exemplo, máquinas de tradução). A configuração entrada-saída emparelhada ($T_x = T_y$) processa uma saída a cada passo que recebe uma entrada. Um exemplo clássico é a tarefas de categorização gramatical de palavras ou *PoS-tagging*.

A arquitetura proposta neste trabalho foi projetada para operar com uma configuração M2M de entrada-saída emparelhada. Neste caso, a função de perda total é definida pela soma das funções de perda em cada instante t. Desta forma, seja x_t a entrada, y_t a saída e L_t a verossimilhança negativa de y_t dado $x_1, ..., x_\tau$, a função de perda é definida na Eq. (2-5):

$$\begin{aligned} L(\{x_1, ..., x_\tau\}, \{y_1, ..., y_\tau\}) &= \sum_t L_t, \\ &= -\sum_t \log p_{\text{model}}(y_t | x_1, ..., x_t), \end{aligned} \tag{2-5}$$

onde $p_{\text{model}}(y_t|x_1, ..., x_t)$ é calculado entre a sequência y_t e a saída do modelo $\hat{y}_t$.

Rede Recorrente por configuração de portas

Em princípio, uma RNR pode memorizar sequências de eventos passados em forma de ativações. Porém, na prática, o aprendizado torna-se muito complexo ou irrealizável, especialmente quando as dependências entre eventos da sequência são muito longas [74]. Isto se deve à dificuldade de resolver o problema de desvanecimento – *vanish* – ou explosão – *blow up* – do gradiente do erro durante a retropropagação. Este tipo de RNR introduz a ideia de se utilizarem laços internos de forma a manter uma dependência linear entre a memória passada e a memória atual [75]. Esta linearidade permite ao gradiente fluir durante longas sequências. Long Short-Term Memory (LSTM) e Gated Recurrent Unit (GRU) têm sido os modelos mais populares em muitas tarefas de PLN, como, por exemplo, em máquinas de tradução e compreensão de linguagem. Estes modelos são compostos por várias "células RNR" que realizam recorrências internas (laços internos), além da saída recorrente. As "células RNR" comportam-se como RNRs ordinárias, formando um sistema de configuração de portas que controla a informação armazenada no modelo.

1. **Long short-term memory (LSTM):** modelo proposto por [75]; um diagrama desta estrutura é apresentado na Figura 2.16.

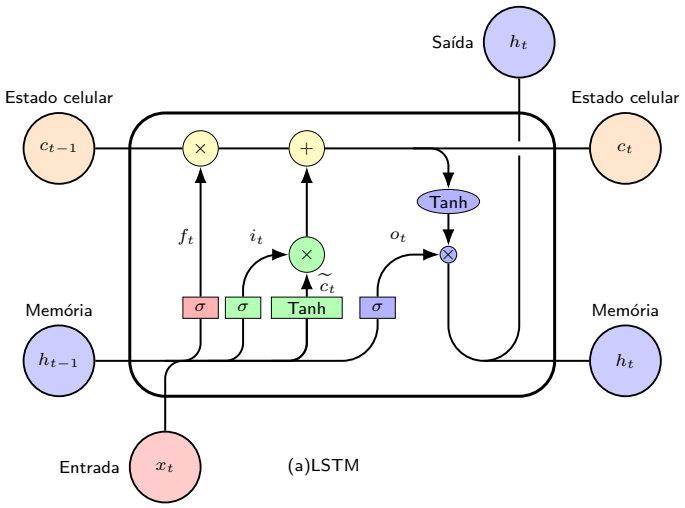


Figura 2.16: Diagrama de uma Long short-term memory (LSTM).

O componente mais importante da LSTM é o estado celular $c_t \in \mathbb{R}^H$, onde H é o número de neurônios. Esta célula comporta-se como uma faixa transportadora, na qual se pode colocar informação com a menor transformação (apenas linear) durante o processamento da sequência. A informação mantida em c_t é controlada pela porta de esquecimento $f_t \in [0,1]^H$. A porta

de entrada $i_t \in [0,1]^H$ determina quais elementos do vetor de novos candidatos $\widetilde{c}_t \in [-1,1]^H$ serão adicionados à célula c_t.

$$f_t = \sigma(W_f[x_t, h_{t-1}] + b_f), \tag{2-6}$$
$$i_t = \sigma(W_i[x_t, h_{t-1}] + b_i), \tag{2-7}$$
$$\widetilde{c}_t = \tanh(W_c[x_t, h_{t-1}] + b_c), \tag{2-8}$$
$$c_t = f^t \odot c_{t-1} + i_t \odot \widetilde{c}_t, \tag{2-9}$$
$$o_t = \sigma(W_o[x_t, h_{t-1}] + b_o), \tag{2-10}$$
$$h_t = o_t \odot \tanh(c_t) \tag{2-11}$$

onde σ é a função sigmoide logística, x_t é o atual vetor de entrada e h_{t-1} é o vetor de estado da camada oculta no passo anterior; b, W são o *bias* e pesos para a porta de esquecimento, porta de entrada, novos candidatos e porta de saída. A atualização do estado celular c_t é apresentada na Eq. (2-9), onde $\odot$ simboliza o produto elemento a elemento entre dois vetores. Na Eq. (2-11), a saída da célula LSTM h_t é calculada usando a porta de saída $o_t \in \mathbb{R}^H$.

2. **Gated Recurrent Unit (GRU):** proposta por [76], é uma variante simplificada da LSTM. Este modelo simplifica a estrutura da LSTM utilizando apenas as portas *update* e *reset* e utiliza a mesma variável como estado celular e saída da rede. Um diagrama desta estrutura pode ser observado na Figura 2.17.

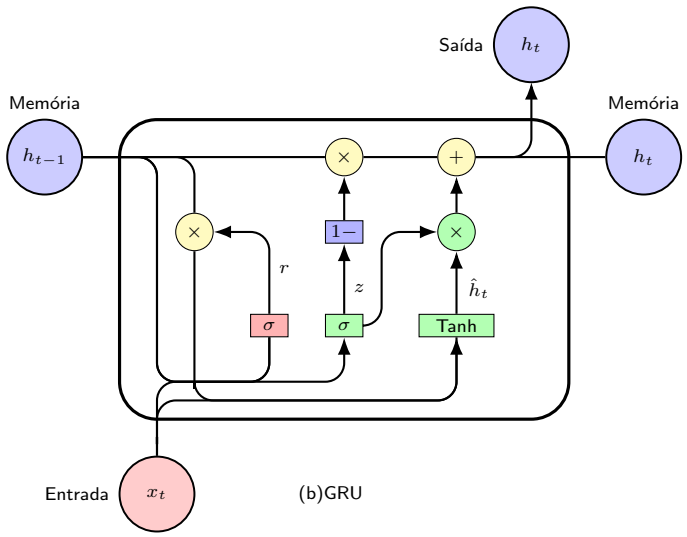


Figura 2.17: Diagrama de uma Gated Recurrent Unit (GRU).

A GRU sintetiza as portas de esquecimento, entrada e saída em portas

de atualização z_t e *reset* r_t nas Eq. (2-12) e (2-13). Os novos candidatos $\widetilde{h}_t$ são calculados na Eq. (2-14), onde r_t seleciona que informações apagar do vetor de estado para a atualização de $\widetilde{h}_t$:

$$z_t = \sigma(W_z[x_t, h_{t-1}] + b_z), \tag{2-12}$$

$$r_t = \sigma(W_r[x_t, h_{t-1}] + b_r), \tag{2-13}$$

$$\widetilde{h}_t = \tanh(W_h[x_t, r_t \cdot h_{t-1}] + b_h), \tag{2-14}$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \widetilde{h}_t, \tag{2-15}$$

Finalmente, h_t é atualizado linearmente na Eq. (2-15).

2.7 Modelos baseados em *Attention*

A habilidade de se concentrar em um padrão específico e ignorar os outros é um papel vital em nossa orientação cognitiva humana. A seletividade no que é observado, pensado, escutado e, até mesmo, nas palavras que compõe um texto, são exemplos dessa "atenção". Os modelos baseados em *Attention* funcionam de forma similar ao verificado em seres humanos: eles observam a sequência de entrada e decidem que partes da sequência são as mais relevantes. Estes mecanismos apareceram como suporte para processar longas sequências de entrada em máquinas de tradução (*Neural Machine Translation* - NMT).

Existem várias formas para se classificar os tipos de *Attention*: pela natureza diferenciável do modelo [77] (*Soft* e *Hard*, pelo espaço de interesse do contexto [78] (*Global Local*) e pela forma de relacionar os elementos da sequência [15] (*Dot*, *Concat*,...). Este trabalho propõe modelos baseados em um tipo de mecanismo de *Attention* denominado *Scaled Dot-Product Attention* [15].

2.7.1 *Scaled Dot-Product Attention*

Proposto por [15], o mecanismo na Figura 2.18 decompõe cada elemento $h_i \in \mathbb{R}^{d_{modelo}}$ da sequência de entrada H em três vetores: *query* q_i, *key* k_i e *value* v_i, onde $q_i, k_i \in \mathbb{R}^{d_k}$ e $v_i \in \mathbb{R}^{d_v}$:

$$q_i = W_q h_i \quad , \quad k_i = W_k h_i, \quad v_i = W_v h_i \tag{2-16}$$

Os valores de d_k,d_v são escolhidos pelo especialista. Por simplicidade, pode-se considerar $d_k = d_v = d_{modelo}$. O modelo relaciona v entre todos os elementos da sequência por meio de uma função de compatibilidade.

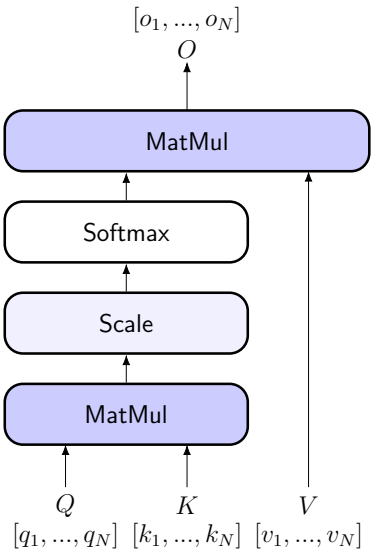


Figura 2.18: Diagrama da estrutura *Scaled Dot-Product Attention*. A operação MatMul representa a multiplicação matricial entre a matrizes Q e K e, posteriormente, o resultado do softmax com a matriz V. Scale representa a multiplicação por um fator de escala que compensa o incremento da magnitude do produto QK.

As operações matriciais do *Attention* são efetuadas sobre as matrizes Q, K e V, que representam as sequências de q_i, k_i e v_i. A saída o_i é calculada como a soma ponderada de v_j, onde o peso atribuído a cada valor v_j é calculado por uma função de compatibilidade entre q_i e o correspondente k_j, onde i, j são elementos da sequência. O mecanismo utiliza o produto ponto como função de compatibilidade entre cada *querie* q_i com todas as *keys* k_j. Esta operação é representada matricialmente na Eq. (2-17).

$$Attention(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V \tag{2-17}$$

O fator de escala $\frac{1}{\sqrt{d_k}}$ compensa o incremento da magnitude do produto $q_i \bullet k_j$ para grandes valores de d_k [15].

2.7.2 *Multi-head Attention*

O módulo *Attention* é estendido de forma a capturar informação de diferentes subespaços de representações em diferentes posições [15]. Em vez de usar uma única função *Attention* com vetores q_i, k_i e v_i de dimensão d_{model}, pode-se projetar linearmente q, k e v para h "cabeças" $(head_1, head_2, ...)$ de *Attention*, onde:

$$\begin{gathered}\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_h)W^O \\ \text{onde head}_i = \mathit{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{gathered} \tag{2-18}$$

em que $W_i^Q, W_i^K \in \mathbb{R}^{d_{model}\times d_k}$ e $W_i^V \in \mathbb{R}^{d_{model}\times d_v}$ são as matrizes que projetam as entradas Q, K e V para cada cabeça i. O peso $W^O \in \mathbb{R}^{hd_v\times d_{model}}$ multiplica a matriz que concatena as h cabeças de *Attention* para calcular a saída . Assim, um esquema global do módulo *Multi-head Attention* é apresentado na Fig. 2.19

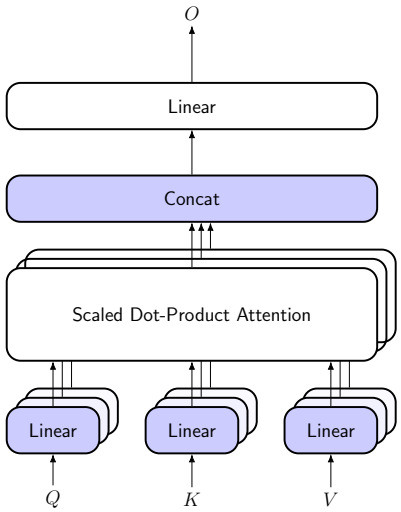


Figura 2.19: *Multi-head Attention*. As matrizes Q, K e V são projetadas para três cabeças de *Attention* ($h = 3$); cada módulo "Linear" multiplica a entrada por uma matriz de pesos W.

Na prática, $d_k = d_v = d_{model}/h$. Como a dimensão de cada cabeça é reduzida, o custo computacional total é similar a uma única cabeça de *attention* com dimensão d_{model}.

2.7.3 Transformer

O modelo *Transformer* [15], foi proposto inicialmente para tarefas de *Neural Machine Translation*. Estas, além de reduzir a complexidade computacional, também superaram o estado da arte. Este modelo tem sido utilizado em outras tarefas de modelamento sequencial como extração de relações [80] e geração de grandes imagens [79]. A arquitetura é baseada no mecanismo *Attention*, definido na seção anterior. Este modelo apresenta uma arquitetura Encoder-Decoder, mas, diferentemente de modelos Seq2Seq, não utiliza redes recorrentes (LSTM, GRU, etc). Esta diferença elimina a recorrência, reduzindo o número de operações sequenciais e a complexidade computacional. A arquitetura do modelo é apresentada na Figura 2.20:

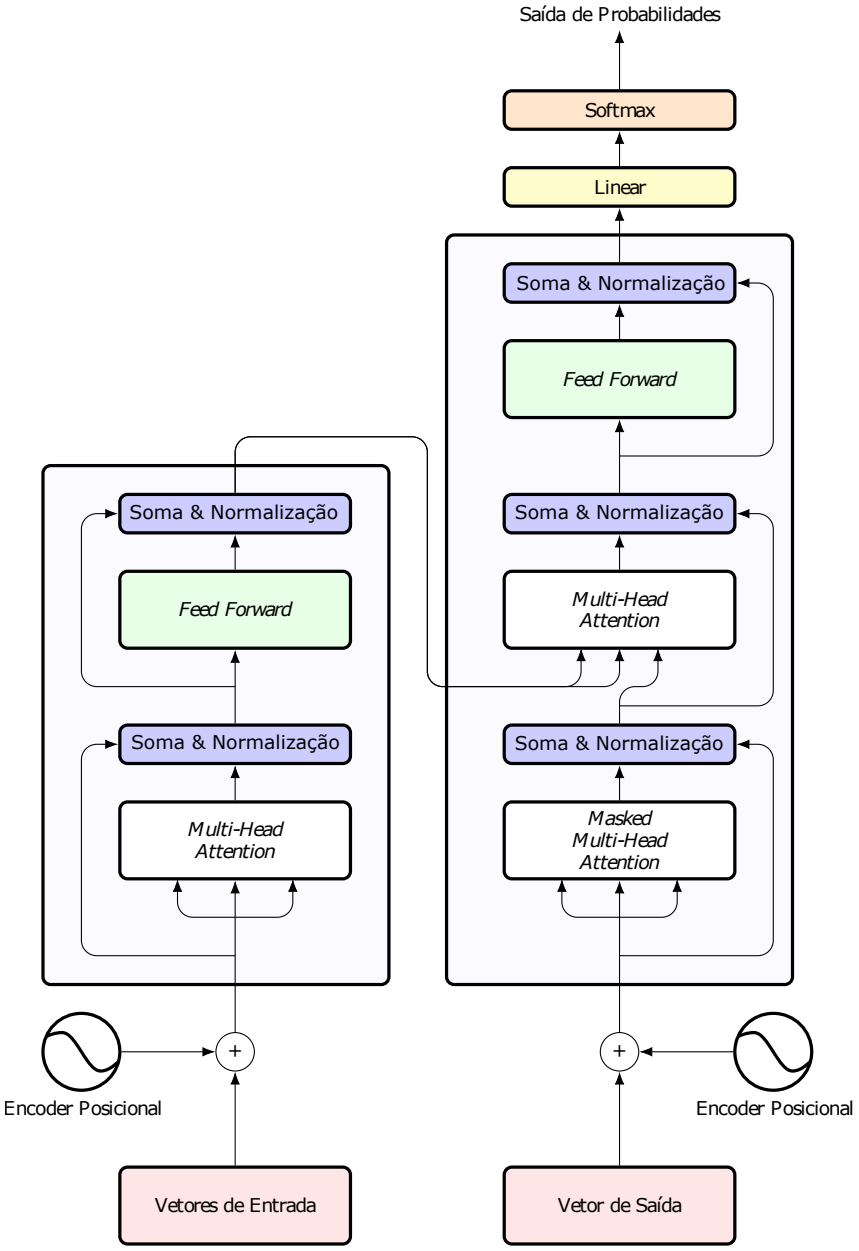


Figura 2.20: *Transformer*

Uma das características importantes para eliminar a recorrência é o *encoding* de posição [81], que permite manter as características de ordem dentro da sequência. A arquitetura *encoder-decoder* do *transformer* baseia-se em mecanismos *Multi-Head Dot-Product Attention* e camadas *full-connected*. O *encoder* recebe uma sequência de entrada de tamanho I. Cada elemento da sequência é inicializado como um *embedding* de dimensão d, que representa o símbolo de cada posição da sequência. Assim, a sequência é representada pela matriz de entrada $H_0 \in \mathbb{R}^{I \times d}$. O modelo calcula, para cada iteração t, a representação H_t em paralelo para todas as posições usando o mecanismo de *Attention* e uma função de transição. O *encoder* utiliza dois tipos de mecanismos que ajudam a rede manter estabilidade da rede e aumentar a

profundidade, evitando reduzir a perda de informação através da distorção da informação a cada camadas:

- Conexão Residual: o valor da entrada é adicionado à saída da camada. Isto permite que informações das camadas iniciais possam chegar até as camadas de maior profundidade, dado que estas são alteradas apenas linearmente ao passar por cada camada.
- Camada de Normalização: subtrai-se a entrada da média e divide-se o resultado pelo desvio padrão; a média e o desvio padrão são estimados durante o treinamento.

Desta forma, camadas de conexão residual e normalização são aplicadas à saída de cada camada. Para evitar a iteração sequencial no *decoder* e garantir o princípio do modelo autoregressivo (dependência apenas de eventos passados), uma máscara é aplicada na primeira camada *Self-Attention* do *decoder*. Detalham-se a seguir outras camadas que compõem a estrutura interna do modelo *transformer*:

1. ***Encoder* Posicional**: utilizado para incorporar informação relativa à ordem da sequência. Para cada *embedding* de entrada e saída, o *encoder* posicional adiciona um *embedding* de dimensão d. Este vetor é definido utilizando-se funções seno e cosseno:

$$\begin{aligned} P^{i,2j} &= \sin\left(\frac{i}{10000^{2j/d}}\right) \\ P^{i,2j+1} &= \cos\left(\frac{i}{10000^{2j/d}}\right) \end{aligned} \tag{2-19}$$

onde i e j são as posições nas dimensões do I e d de H.

2. ***Attention***: o modelo usa a versão *Multi-Head Attention* do módulo *Scaled Dot-Product Attention*, descrito na seção 2.7, de forma a detectar múltiplas informações em vários subespaços de representação a cada posição da sequência.

3. ***Feed-Forward***: são duas transformações lineares com função de ativação ReLU. Pode-se considerar este módulo como duas camadas convolutivas com *kernel-size* $= 1$. Sendo d a dimensão *embedding* do modelo *transformer*, o *embedding* é expandido até $4d$ pela primeira camada e reduzido novamente para d na segunda camada.

4. ***Softmax***: para calcular a saída de probabilidades, os vetores *embedding* são transformados linearmente ao número de classes de saída e depois submetidos à função softmax.

2.8 Modelos Baseados em Memória

A configuração de portas dentro de uma célula recorrente foi um dos aprimoramentos mais importantes nas redes neurais recorrentes. Mesmo assim, estas redes ainda estão longe de atingir seu máximo potencial. Na seção 2.6.2 mencionou-se que a arquitetura RNR apresenta completude de Turing. Isso significa que, dada as conexões e os parâmetros adequados, uma RNR pode aprender a resolver qualquer problema que possa ser resolvido por um algoritmo de computador ou, equivalentemente, por uma máquina. Na prática, essa universalidade é extremamente difícil de obter, já que se está analisando um imenso espaço de busca de possíveis conexões e valores de parâmetros da RNR. Porém, este espaço é suficientemente grande para que o gradiente descendente encontre uma solução apropriada para qualquer problema arbitrário. Nesta seção serão apresentadas algumas abordagens do aprendizado profundo que tentam explorar esse potencial.

2.8.1 Neural Turing Machine

Em 2014, Graves et al. [19], do Google DeepMind, introduziram uma nova arquitetura de rede neural: *Neural Turing Machine* (NTM). Este modelo é composto por uma rede neural (usualmente uma RNR) que controla uma memória externa semelhante operacionalmente à memória do computador. Da mesma forma que a RNR, a rede itera sobre uma sequência de entrada realizando acessos a memória em cada iteração. Durante o acesso, os pesos ou "cabeças" (*heads*) de escrita e leitura escolhem as posições de memórias que serão usadas. A arquitetura da NTM é apresentada na Fig. 2.21:

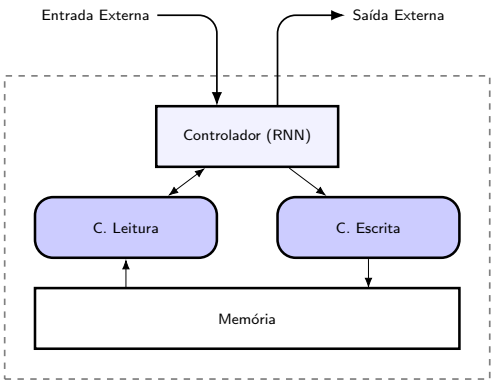


Figura 2.21: Arquitetura de uma Neural Turing Machine ()NTM). Em cada ciclo de funcionamento o controlador recebe entradas externas e processa as saídas. A rede escreve e lê da memória utilizando um conjunto de cabeças de leitura e escrita.

Considerando o conceito de completude [72], o fato de se acrescentar à RNR uma memória externa para armazenamento transitório (isto é, durante o processamento da sequência) remove uma grande parte do espaço de possíveis soluções, pode ocasionar a perda da completude de Turing. Dado que agora não são exploradas todas as RNRs que processam e armazenam informações, o escopo de busca é reduzido apenas para as RNRs que podem processar as informações armazenadas fora delas.

Para treinar a NTM com o método do gradiente descendente, o modelo deve ser completamente derivável. Esta propriedade é chamada *End-to-End-differentiable*. A NTM consegue ser derivável pois apenas acrescenta à estrutura da rede neural mecanismos deriváveis compostos principalmente por pequenos módulos de *Attention*.

A NTM foi o primeiro modelo de um grupo de redes neurais com acesso a uma memória externa. Em 2016, Graves et al. [22] propuseram a segunda versão deste modelo, denominado *Differentiable Neural Computer* (DNC).

2.8.2 *Differentiable Neural Computer*

Este modelo tenta reduzir as limitações do mecanismo de acesso a memória de seu antecessor. Similarmente à NTM, o DNC consiste em um controlador que interage com um módulo de acesso a uma memória externa. A memória, composta por N palavras de tamanho W, é representada pela matriz $M \in \mathbb{R}^{N \times W}$. Em cada instante t, o controlador recebe um vetor de entrada $x_t \in \mathbb{R}^X$ e R vetores de leitura $r^1_{t-1}; r^2_{t-1}; ...; r^R_{t-1}$, lidos da memória externa na iteração anterior, onde R é o número de cabeças de leitura. A Figura 2.22 apresenta um diagrama do controlador neural.

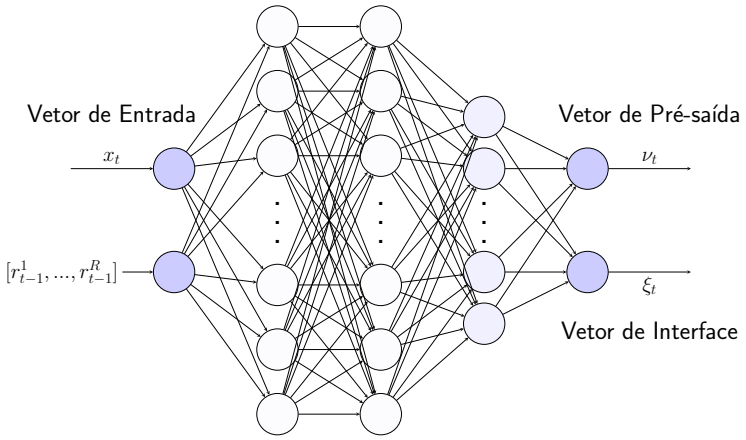


Figura 2.22: Rede Neural - Controlador. O Vetor de Pré-saída ν_t e o Vetor de Interface ξ_t são calculados a partir do Vetor de Entrada x_t e R vetores de leitura do estado anterior $[r^1_{t-1}, ..., r^R_{t-1}]$.

Após processado x_t, o controlador calcula o vetor de interface ξ_t, que contém toda a informação necessária para controlar o acesso a memória (leitura/escrita), e um vetor de pré-saída ν_t, que é parte da saída final da rede. Para simplificar a nomenclatura, define-se a variável $\chi_t = [x_t, r^1_{t-1}; r^2_{t-1}; \ldots; r^R_{t-1}]$ como vetor de entrada. Neste trabalho, define-se o controlador com uma única camada recorrente $h_t = \text{LSTM}(\chi_t, h_{t-1})$, onde a LSTM foi descrita na seção 2.6.2. Na Eq. (2-20), os vetores ν_t e ξ_t são definidos em função do estado h_t.

$$\begin{aligned} \nu_t &= W_\nu h_t, \\ \xi_t &= W_\xi h_t, \end{aligned} \tag{2-20}$$

O módulo de acesso à memória retorna R vetores de leitura que posteriormente são concatenados e integrados com vetor de pré-saída ν_t para processar o vetor de saída y_t, como se observa na Eq. (2-21).

$$y_t = \nu_t + W_r[r^1_t; ...; r^R_t], \tag{2-21}$$

A Fig. 2.23 resume este funcionamento.

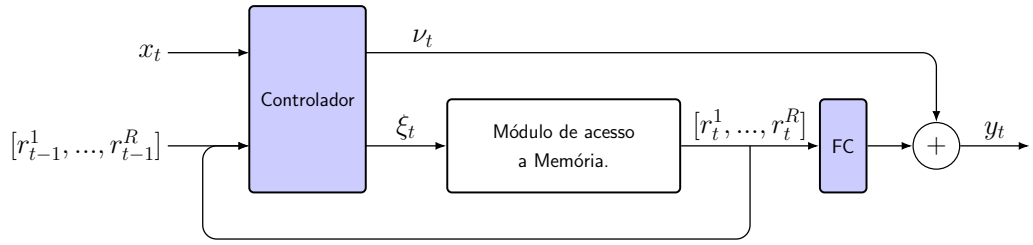


Figura 2.23: *Differentiable Neural Computer.*

O vetor de interface ξ_t contém todos os parâmetros necessários para o módulo de acesso a memória, conforme apresentado na Eq. 2-22. Este módulo realiza uma única escrita e lê R novos vetores da memória.

$$\xi_t = \{k^{r,1}_t; ...; k^{r,R}_t; \hat{\beta}^{r,1}_t; ...; \hat{\beta}^{r,1}_t; k^{w,1}_t; \hat{\beta}^{w,1}_t; \hat{e}_t; v_t; \hat{f}^1_t; ...; \hat{f}^R_t; \hat{g}^a_t; \ \hat{g}^w_t; \hat{\pi}^1_t; ...; \hat{\pi}^R_t\} \tag{2-22}$$

Cada componente de ξ_t é processado usando diferentes funções para assegurar a permanência dos parâmetros no domínio correto. As transformações são apresentadas na Tabela 2.4.

Função	Descrição	Domínio
sigmoid	$\frac{1}{1+e^{-x}}$	$[0, 1]$
oneplus	$1 + log(1 + e^x)$	$[1, \infty)$
softmax	$\frac{e^{x_i}}{\sum_j^N e^{x_j}}$	$\mathcal{S}_N = \{\alpha \in \mathbb{R}^N : \alpha \in [0, 1], \sum_{i=1}^N \alpha_i = 1\}$

Tabela 2.4: Transformação de domínio dos parâmetros do vetor de interface ξ_t.

Realizadas as transformações, obtêm-se os seguintes parâmetros:

Porta de alocação: $g_t^a = \sigma(\hat{g}_t^a) \in [0, 1]$;
Porta de escrita: $g_t^w = \sigma(\hat{g}_t^w) \in [0, 1]$;
Vetor apagar: $e_t = \sigma(\hat{e}_t) \in [0, 1]^W$;
Vetor de escrita: $v_t \in \mathbb{R}^W$;
Chave de escrita: $k_t^w \in \mathbb{R}^K$;
Força de escrita: $\beta_t^w = \text{oneplus}(\hat{\beta}_t^w) \in [1, \infty)$;
R portas livres: $\{f_t^i = \sigma(\hat{f}_t^i) \in [0, 1]; 1 \leq i \leq R\}$;
R chaves de leitura: $\{k_t^{r,i} \in \mathbb{R}^W; 1 \leq i \leq R\}$;
R forças de leitura: $\{\beta_t^{r,i} = \text{oneplus}(\hat{\beta}_t^{r,i}) \in [1, \infty); 1 \leq i \leq R\}$;
R modos de leitura: $\{\pi_t^i = \text{softmax}(\hat{\pi}_t^i) \in \mathcal{S}_3; 1 \leq i \leq R\}$;

A Porta de alocação g_t^a e a Porta de escrita g_t^w são usadas para determinar a posição de memória onde o Vetor de escrita v_t será armazenado. O Vetor apagar e_t deve decidir se a informação previamente armazenada nessa posição de memória será apagada antes de realizar a escrita. As Chaves de escrita k_t^w e de leitura $k_t^{r,i}$ são utilizadas para escolher a posição de memória por similaridade entre a chave e o conteúdo. As Forças de escrita β_t^w e leitura $\beta_t^{r,i}$ ajudam a incrementar a maior probabilidade dentro de uma função softmax. As R Portas Livres f_t^i permitem reutilizar posições de memória e os R modos de leitura π_t^i selecionam o tipo de endereçamento de memória durante a leitura. O controle de acesso a memória é realizado mediante o armazenamento ou leitura da informação. Isto é realizado via Pesos de escrita e leitura. A Figura 2.24 mostra quais valores do Vetor de interface são utilizados para calcular os pesos de escrita e leitura:

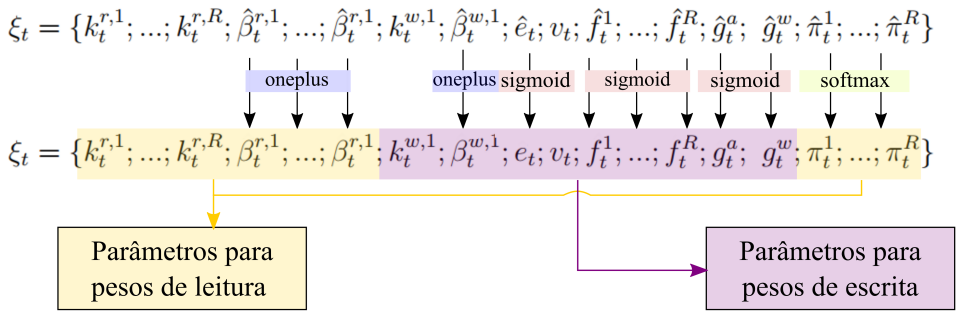


Figura 2.24: Valores do vetor de interface para a escrita e leitura de memória.

O procedimento do controle do acesso à memória é detalhado a seguir.

Módulo de Controle de Acesso à Memória

O controle de acesso à memória realiza-se via pesos de escrita/leitura obtidos a partir dos parâmetros da interface ξ_t. Os pesos são formados por valores não negativos, cuja soma é no máximo 1. Este espaço é representado pelo conjunto Δ_N na Eq. (2-23):

$$\Delta_N = \{\alpha \in \mathbb{R}^N : \alpha_i \in [0,1], \sum_{i=1}^{N} \alpha_i \leq 1\} \tag{2-23}$$

A cada iteração t, o peso de escrita $w_t^w \in \Delta_N$ indica a posição de memória na qual será adicionada a nova informação: o vetor de escrita $\nu_t \in \mathbb{R}^W$. Esta atualização é representada pela Eq. (2-24):

$$M_t = M_{t-1}(E - w_t^w e_t^\top) + w_t^w v_t^\top, \tag{2-24}$$

o vetor de remoção $e_t \in [0,1]^W$ permite limpar esse endereço de memória antes de se realizar a escrita. Uma representação gráfica da atualização de memória é apresentada na Figura 2.25:

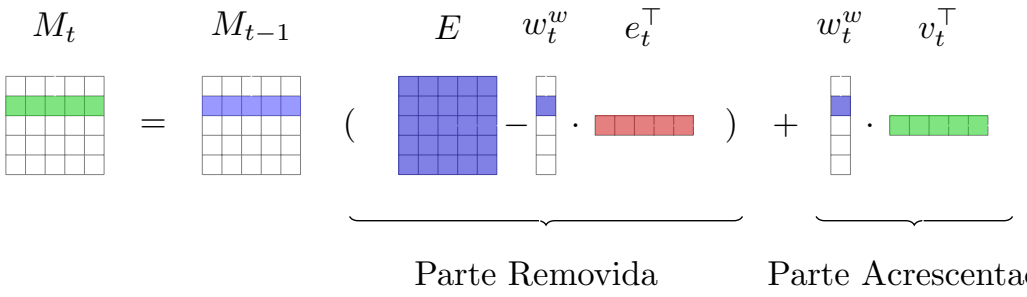


Figura 2.25: Diagrama de atualização de memória. As linhas de cada matriz e vetor indicam um endereço da memória.

onde E é uma matriz com todos os valores iguais a 1 e da mesma dimensão de M. O fator $(E - w_t^w e_t^\top)$ indica a parte que será removida e $w_t^w v_t^\top$, a informação que será acrescentada. Os pesos de escrita e leitura são definidos pelos tipos de endereçamentos. Para o cálculo do peso de escrita e leitura, o endereçamento baseado no conteúdo é um método comum utilizado por ambos acessos à memória.

Endereçamento baseado no conteúdo: procura o conteúdo da posição n da memória $M[n,:]$ com maior compatibilidade com a chave $k \in \mathbb{R}^W$. A função de compatibilidade D utilizada na literatura é a distância de cossenos. Assim, o peso baseado no conteúdo é calculado usando a função de compatibilidade D, a função softmax em conjunto com uma força de escrita β_t^w:

$$C(M, k, \beta)[n] = \frac{\exp\{D(k, M[n,:])\beta\}}{\sum_j \exp\{D(k, M[j,:])\beta\}}, \tag{2-25}$$

Um exemplo do cálculo do peso baseado no conteúdo é apresentado na Figura 2.26:

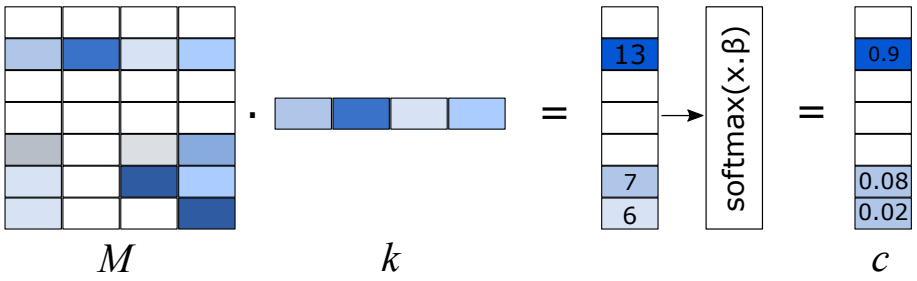


Figura 2.26: Diagrama do calculo do peso baseado em conteúdo.

A chave k realiza o produto escalar com cada linha da matriz M, obtendo-se um valor de similaridade não normalizado. Este valor passa por uma função softmax, sendo previamente multiplicado pelo valor escalar de força β. Ele incrementa a probabilidade dos valores com maior grau de similaridade. Finalmente é obtido o vetor C, que aponta para a posição de memória com o maior grau de compatibilidade.

Apresentam-se a seguir os cálculos dos pesos de escrita e leitura do mecanismo de acesso a memória.

1. **Peso de Escrita**

O cálculo do peso de escrita é apresentado na Eq. (2-26). A porta de alocação g_t^a seleciona a posição de memória a partir de dois tipos de endereçamento: alocação de memória dinâmica (a_t) e endereçamento baseado no conteúdo (c_t).

$$w_t^w = g_t^w(g_t^a a_t + (1 - g_t^a)c_t), \tag{2-26}$$

A porta de escrita g_t^w decide se no instante t é necessário realizar a escrita. O comportamento da Eq. (2-26) é similar à versão suave de um circuito eletrônico onde as portas g_t^a e g_t^w são representados por *switches* e os vetores a_t, c_t e w_t^w são os terminais do circuito. Este esquema é apresentado na Figura 2.27:

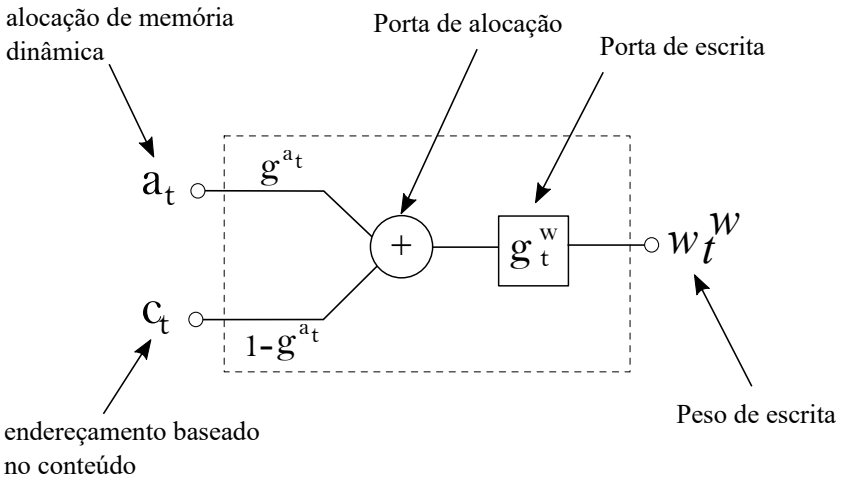


Figura 2.27: Representação do circuito do cálculo do peso de escrita w_t^w.

Observa-se que, independentemente do valor da porta g_t^a e dos endereçamentos de entrada, a porta de escrita g_t^w controla o comportamento global do sistema, avaliando a necessidade de escrita a cada instante de tempo t. Os dois mecanismos de endereçamento são detalhados a seguir:

Alocação de memória dinâmica (a_t): permite controlar a liberação e alocação de informação mediante a lista de uso de posições de memória $u_t \in [0,1]^N$, onde $u_0 = 0$. Para calcular a_t, é necessário responder a duas perguntas sobre a memória:

- P1: Como liberar as posições de memória utilizadas? Usa-se o vetor de retenção de memória $\varphi \in [0,1]^N$, na Eq. (2-27), que representa o quanto de cada posição de memória será preservado. Neste caso, consideram-se as posições lidas recentemente cujo conjunto de portas livres f_t^i não foram liberadas.

$$\varphi_t = \Pi_{i=1}^R (1 - f_t^i w_{t-1}^{r,i}) \tag{2-27}$$

- P2: Como evitar reescrever em posições previamente escritas? Adiciona-se ao vetor u_{t-1} o fator $(1 - u_{t-1})w_{t-1}^w$, que calcula as novas posições utilizadas pelo vetor de escrita. Deste modo, integram-se as duas atualizações. O cálculo de

u_t é definido na Eq. (2-28):

$$u_t = (u_{t-1} + (1 - u_{t-1})w_{t-1}^w) \odot \varphi_t \tag{2-28}$$

O vetor de alocação a_t é definido multiplicando-se cada elemento da lista "posições de memória não usadas" $(1 - u_t)$ por um fator que incentiva a liberação das posições de memória menos usadas. Este fator é determinado usando-se uma lista de liberação ϕ_t, que ordena os índices das posições de memória de forma decrescente em relação ao uso. Por exemplo, $\phi_t[1]$ é o índice da última posição de memória utilizada. O vetor de alocação é calculado como:

$$a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]], \tag{2-29}$$

Este processo tem um elevado custo computacional, dado que envolve operações de ordem e indexação. Um diagrama do cálculo do vetor de alocação é apresentado na Figura 2.28:

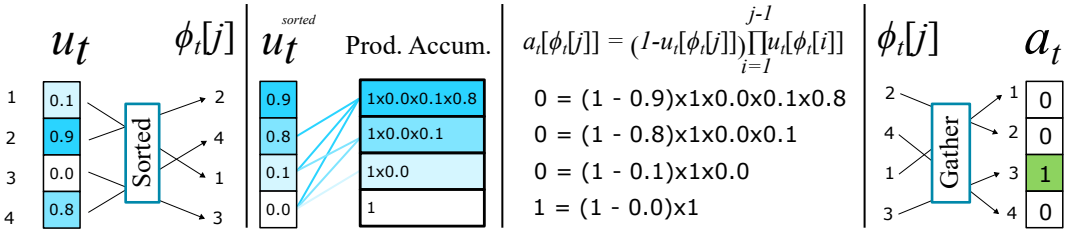


Figura 2.28: Proceso de cálculo do vetor de alocação.

No exemplo, considere-se a lista de posições de memória usadas $u_t = [0.1, 0.9, 0, 0.8]$. A primeira etapa é ordenar a lista de uso u_t^{sorted}, onde $\phi_t[j]$ é a lista de posições de ordem de u_t. O ordenamento permite realizar a operação de produto acumulado para calcular os vetores de alocação em cada posição. Este resultado precisa ser reordenado via a operação de distribuição *Gather*.

Endereçamento baseado no conteúdo (c_t^w): similarmente, o peso de escrita é baseado no conteúdo na Eq. (2-25). Cada cabeça de leitura i calcula o vetor de peso de leitura por conteúdo $c_t^w \in \mathcal{S}_N$ da forma:

$$c_t^w = C(M_{t-1}, k_t^w, \hat{\beta}_t^w), \tag{2-30}$$

2. **Peso de Leitura**

O peso é calculado mediante a seleção do modo de leitura, realizada pelo vetor $\pi_t^i \in S_3$. Os modos de leitura são definidos por dois tipos de endereçamento: conexão temporal (f_t^i e b_t^i) e endereçamento baseado no

conteúdo ($c_t^{r,i}$). O vetor π_t^i realiza uma interpolação entre os vetores b_t^i, f_t^i e $c_t^{r,i}$ para determinar os vetores de pesos de leitura $w_t^{r,i}$:

$$w_t^{r,i} = \pi_t^i[1]b_t^{r,i} + \pi_t^i[2]c_t^{r,i} + \pi_t^i[3]f_t^{r,i} \tag{2-31}$$

De forma similar ao peso de escrita, o peso de leitura também pode ser representado por um circuito eletrônico onde o vetor de ordem de leitura π_t^i comporta-se como um *switch* entre as portas b_t^i, f_t^i e $c_t^{r,i}$ (Figura 2.29).

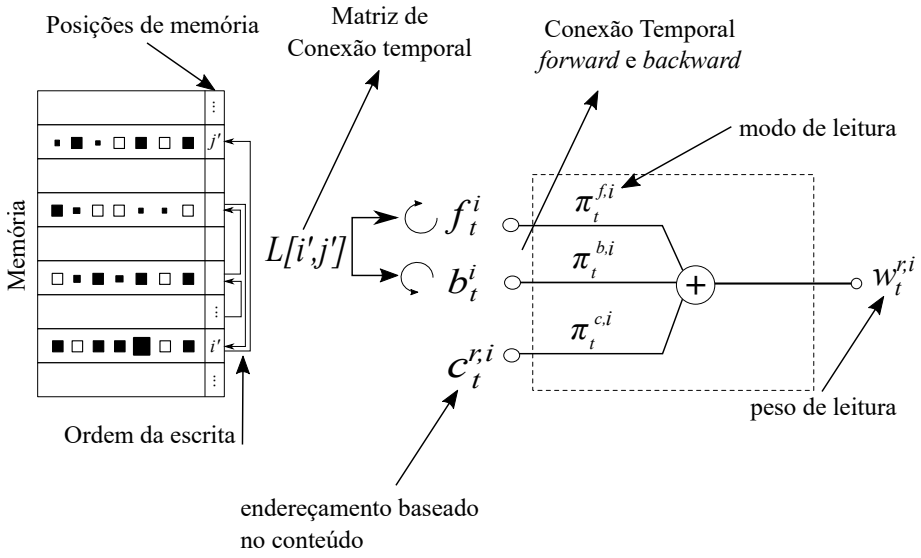


Figura 2.29: Representação do circuito do cálculo do peso de leitura $w_t^{r,i}$.

Observe-se que i' e j' são posições de memória e i indica a cabeça de leitura que está sendo tratada. Dado que o modo de leitura deve selecionar entre mais de duas opções, π_t^i é calculado a partir do vetor de interface e a função softmax. Após o calculo do $w_t^{r,i}$, o vetor de leitura r_t^i é calculado utilizando o peso $w_t^{r,i}$ e a memória M_t:

$$r_t^i = M_t^\intercal w_t^{r,i} \tag{2-32}$$

Conexão Temporal (f_t^i e b_t^i): a matriz de conexões temporais L_t armazena informação sobre a ordem em que as posições de memória foram acessadas pelo peso de escrita. Para isto, define-se o vetor de precedência p_t na Eq. (2-33), onde $p_t[i]$ indica o grau da posição de memória i sendo a "última gravação":

$$\begin{aligned} p_0 &= 0, \\ p_t &= (1 - \sum_t w_t^w[i])p_{t-1} + w_t^w, \end{aligned} \tag{2-33}$$

Para cada instante t, a matriz L_t é atualizada e apaga ligações anteriores (pares $(i-j)$ do instante $(t-1)$), utilizando a seguinte lógica:

$$L_0[i,j] = 0 \quad \forall i,j,$$
$$L_t[i,i] = 0 \quad \forall i, \tag{2-34}$$
$$L_t[i,j] = (1 - w_t^w[i] - w_t^w[j])L_{t-1}[i,j] + w_t^w[i]p_{t-1}[j],$$

Para cada cabeça de leitura i, os vetores $b_t^i, f_t^i \in S_N$ retornam à direção de memória anterior ou seguinte que foi acessada a partir da direção indicada pelo peso de leitura $w_{t-1}^{r,i}$. Para tanto, utiliza-se a equação 2-35.

$$f_t^i = L_t w_{t-1}^{r,i} \quad , \quad b_t^i = L_t^\top w_{t-1}^{r,i} \tag{2-35}$$

Endereçamento baseado no conteúdo ($c_t^{r,i}$): Da mesma forma, o peso de escrita é baseado no conteúdo da Eq. (2-25). Cada cabeça de leitura i calcula o vetor de peso de leitura por conteúdo $c_t^{r,i} = C(M_t, k_t^{r,i}, \hat{\beta}_t^r)$.

2.8.3 Modelos de Aprendizado Profundo para Extração de Relações

Existem várias abordagens de AP para a tarefa de ER. Muitas delas baseiam-se em camadas convolutivas [82],[83],[84],[85], camadas recorrentes, [86],[87],[88],[89] mecanismos *attention* [56],[80], sistemas híbridos [90], etc. O estado de arte considera principalmente dois tipos de arquiteturas: Redes Convolutivas e Modelos baseados em *Transformers*. Nesta seção, é apresentado o modelo estado da arte para cada tipo.

1. **Abordagem baseada em Redes Convolutivas**: o modelo proposto por Dat Quoc et. al. [82], apresentou resultados estado da arte para o conjunto de dados da competição BioCreative V. A característica principal é a construção de representações de palavras usando *embedings* de caracteres. Este vetor, os *embeddings* de palavra e a posição relativa a uma entidade formam o vetor de entrada que representa uma palavra. A arquitetura deste modelo é apresentada na Figura 2.30.

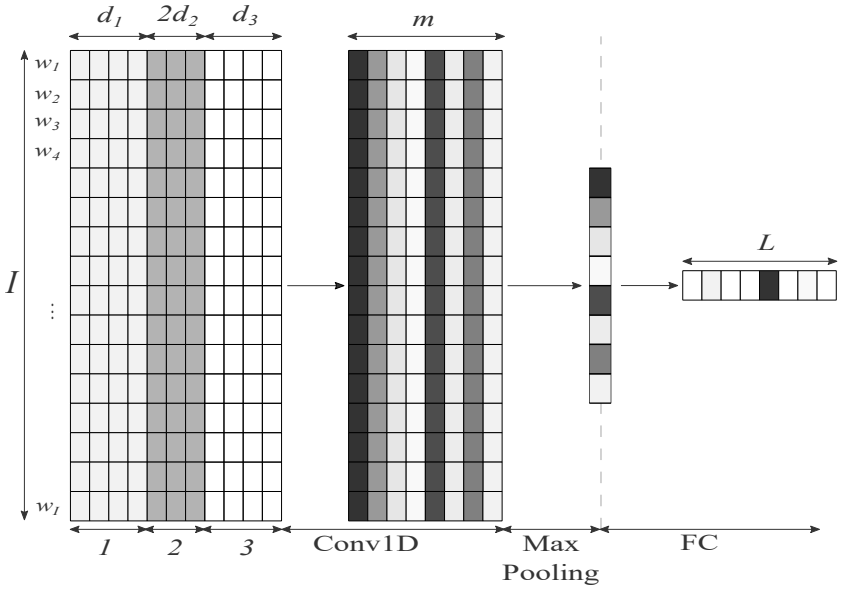


Figura 2.30: Arquitetura de uma rede convolutiva para extração de relações. Os números 1, 2 e 3 indicam os vetores concatenados para representação de palavra, posição relativa e baseada em caracteres.

Considera-se como entrada a sequência de *tokens* $[w_1, ..., w_I]$ de tamanho fixo I. Seja $s^i \in \mathbb{R}^d$ o vetor de representação para cada *token* da sequência. **Entrada:** Cada vetor s^i é o resultado da concatenação dos *embeddings* de palavra, posição e palavra em nível de caractere:

$$s^i = [s^{w,i}, s^{e_1,i}, s^{e_2,i}, s^{c,i}] \tag{2-36}$$

onde $s^{w,i} \in \mathbb{R}^{d_1}$ é o vetor de representação de palavra, $s^{e_1,i}$, $s^{e_2,i} \in \mathbb{R}^{d_2}$ são os embeddings de posição relativa $(i - i_1)$ e $(i - i_2)$ respeito à entidade e_1 e e_2 e $s^{c,i} \in \mathbb{R}^{d_3}$ é o vetor de representação a nível de caracter. O vetor $s^{c,i}$ é definido a partir da sequência de caracteres para cada palavra (*token*) $w = c^1c^2...c^l$. Para processar a sequência de caracteres, utiliza-se uma camada convolutiva 1D C_k, onde k é o tamanho do kernel, e uma camada *Max-Pooling* aplicada sob a dimensão da sequência de caracteres de forma a eliminar esta dimensão:

$$c = [c^1, ..., c^l] \tag{2-37}$$

$$s^{c,i} = \max_l(C_k(c^\top)) \tag{2-38}$$

Rede Neural: a sequência de *tokens* é representada pela matriz de entrada $s = [s^1, ..., s^I] \in \mathbb{R}^{I \times d}$. A entrada passa por outra camada convolutiva 1D C_k, gerando m novas características na Eq. (2-40);uma operação de Max-Pooling,

realizada sobre a dimensão I, elimina esta dimensão, gerando o vetor $s_1 \in \mathbb{R}^m$:

$$y = \text{softmax}(W_y s_1 + b) \tag{2-39}$$

$$\text{onde} \quad s_1 = \max_I(C_k(s)^\top) \tag{2-40}$$

onde $W_y \in \mathbb{R}^{mxL}$ e $b \in \mathbb{R}^L$ e a função softmax processam a saída final $y \in \mathbb{R}^L$ da rede na Eq. (2-39), onde L é o número de tipo das relações.

2. **Abordagem baseada em *Transformers*:** O modelo BRAN (*Bi-affine Relation Attention Netoworks*), proposto por Verga et. al.[80], foi projetado para codificar longas sequências de texto, ou seja, a sequência pode ser composta por múltiplas sentenças. Isto permite tratar a tarefa de ER quando as entidades se encontram em sentenças diferentes. Outra característica é o treinamento multitarefa com a tarefa REN, que permite compartilhar representações aprendidas por ambas as tarefas e melhorar a generalização do modelo, como explicado na seção 2.5. A estrutura utiliza uma variante do módulo *Transformer* descrito na seção 2.7,em um número constante de iterações, de forma paralela, sobre todos os vetores de entrada. Na saída da estrutura, um classificador *bi-affine* gera uma matriz de "afinidade" de relações para cada par de elementos da sequência. Um diagrama deste modelo é apresentado na Figura 2.31.

O *Transformer* processa os *embeddings* de cada *token* e os transforma em *embeddings* representativos para resolver as tarefas de REN e ER. Para a tarefa REN, cada elemento da sequência passa por um classificador (*full-connected*) para determinar o tipo de entidade. Para a tarefa de ER, cada elemento da sequência é projetada em dois vetores de representação: e_{head} e e_{tail}, entradas para o classificador *bi-affine*, quem determina o tipo de relação.

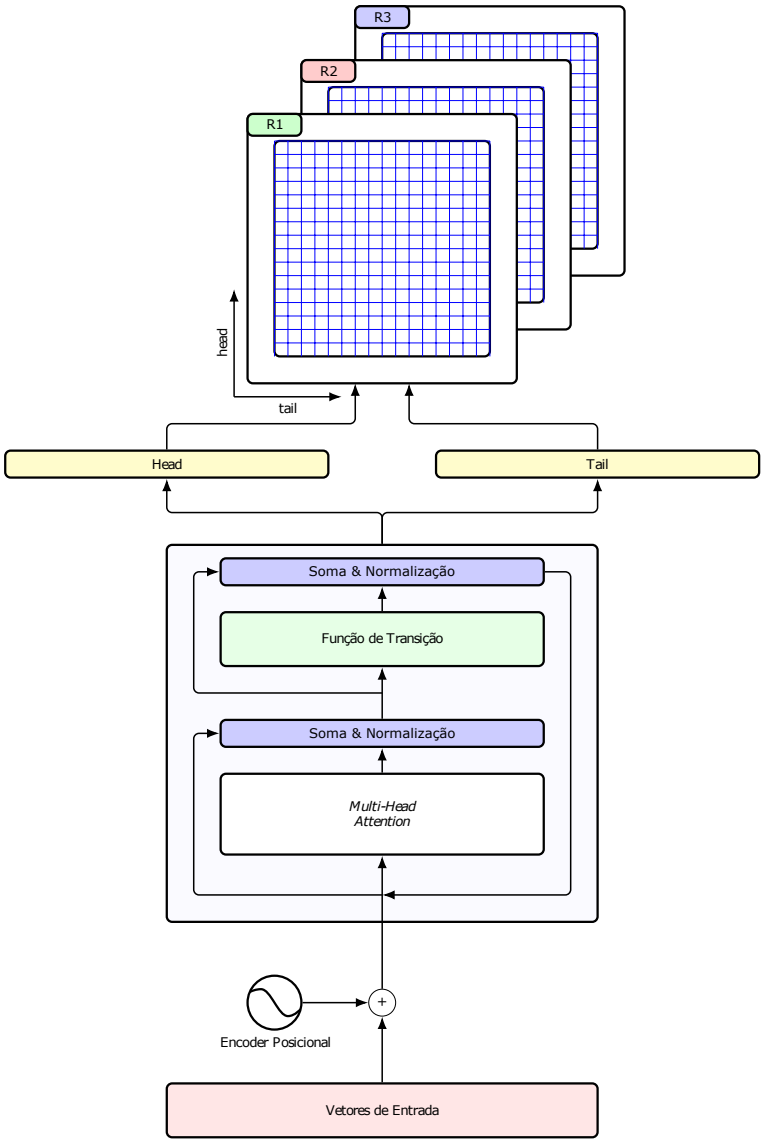


Figura 2.31: Modelo BRAN (*Bi-affine Relation Attention Networks*).

Entrada: Considera-se como entrada a sequência de *tokens* $[w_1, ..., w_I]$ e $s^i \in \mathbb{R}^d$ como o vetor de representação para cada *token* da sequência. Cada vetor s^i é o resultado da soma de *embeddings* de palavra e posição.

$$s^i = s^{w,i} + s^{p,i} \tag{2-41}$$

Rede Neural: a sequência de entrada $s = [s^1, ..., s^I] \in \mathbb{R}^{I \times d}$ é processada durante T iterações pelo módulo *Transformer*. Este módulo é composto pelo mecanismo *Scaled Dot-Product Attention* seguido por uma função de transição, como descrito na seção 2.7. As equações do *Transformer* são apresentadas na

Eq. (2-42) e (2-43), onde $H_0 = s$ é a matriz de entrada.

$$H_t = A_t + \text{Transition}(A_t) \tag{2-42}$$

$$\text{onde} \quad A_t = \text{LN}(H_{t-1} + \text{MultiHeadSelfAttention}(H_{t-1})) \tag{2-43}$$

A função de transição utilizada é um conjunto de camadas convolutivas com ativação ReLU, apresentada na Eq. (2-44), onde C_5 e C_1 são camadas convolutivas 1D com kernel 5 e 1 respectivamente.

$$\begin{aligned} \text{Transition}(A_t) &= C_1(A_t^{(1)}) \\ \text{onde} \quad A_t^{(1)} &= \text{ReLU}(C_3(A_t^{(0)})) \\ A_t^{(0)} &= \text{ReLU}(C_1(A_t)) \end{aligned} \tag{2-44}$$

Extração de Relações: A saída do módulo *Transformer* é projetada para calcular as entidades e^i_{head} e e^i_{tail} de cada elemento h^i_T da sequência H_T. As Eq. (2-45) e (2-46) apresentam o cálculo para toda a sequência.

$$E_{head} = W^{(1)}_{head}(\text{ReLU}(W^{(0)}_{head}H_T)) \tag{2-45}$$

$$E_{tail} = W^{(1)}_{tail}(\text{ReLU}(W^{(0)}_{tail}H_T)) \tag{2-46}$$

onde $E_{head}, E_{tail} \in \mathbb{R}^{I\times d}$ são as matrizes de sequências de e^i_{head} e e^i_{tail} respectivamente. O tensor de afinidade $A_{ff} \in \mathbb{R}^{I\times L\times I}$ é calculado usando a operação *bi-affine* da Eq. (2-47), na qual $L \in \mathbb{R}^{d\times L\times d}$ é a matriz de *embeddings* de relações para as L relações:

$$A_{ff} = (E_{head}L)E^\top_{tail} \tag{2-47}$$

A previsão é realizada em nível de pares de entidades, ou seja, sobre a agregação do *score* de todas as posições de cada par de entidade anotada. Para o par de entidades (p_{head}, p_{tail}), sejam P_{head} e P_{tail} os índices de todas as anotações das entidades p_{head} e p_{tail} respectivamente. Então, a função *score* é definida como:

$$score(p_{head}, p_{tail}) = \log \sum_{\substack{i\in P_{head} \\ j\in P_{tail}}} \exp(A_{ff}[i, :, j]) \tag{2-48}$$

Reconhecimento de Entidades Nomeadas: A saída do módulo *Transformer* é projetada por uma transformação linear de forma a preceder a classe de entidade para cada elemento da sequência:

$$C = W_{ner}H_T \tag{2-49}$$

3
Modelo Proposto

3.1
Introdução

Este trabalho propõe duas extensões do modelo *Transformer* aplicadas à tarefa de extração de informação, com ênfase na tarefa de extração de relações, dado que esta é uma de maior complexidade. Os dois enfoques partem da ideia proposta por Verga [80], que utiliza o módulo *Transformer* como centro de processamento. A primeira abordagem explora e integra vários recursos do AP, como *embeddings* de caracteres e posição, uma função de transição mais eficiente para o módulo *Transformer* e a adição do mecanismo de Parada Dinâmica. A segunda abordagem é uma extensão da primeira. Nesta, é proposto um mecanismo de acesso à memória sob a estrutura *Transformer*. Isto permite um maior controle sobre as interações entre os elementos da sequência, o que se traduz em um aumentando da acurácia. O mecanismo utilizado é um Módulo Controle de Acesso à Memória (MCAM) do modelo DNC [22], o qual foi modificado de forma a paralelizar o processamento de longas sequências de texto.

3.2
Primeira Abordagem: *Universal Transformer* para extração de relações (UTRE)

O modelo proposto nesta abordagem foi baseado e visou a aprimorar o modelo *Transformer* para ER, descrito na seção 2.8.3. Para tanto, foram realizadas três modificações na estrutura:

- Encoder de Entrada: acréscimo de recursos linguísticos do texto em forma de *embeddings*, o que facilita o aprendizado destas representações.
- Função de Transição: uso de blocos convolutivos que apresentam excelentes resultados em tarefas de visão computacional, podendo ser aproveitados da mesma forma em tarefas de PLN.
- Condição de Parada dinâmica (CPD): proposta no modelo Universal Transformer [91], permite controlar os recursos de computação necessários para o processamento de cada elemento da sequência. A CPD dota

o modelo da completude de Turing, conseguindo assim representar qualquer tipo de programa.

Na Figura 3.1, apresenta-se a estrutura do modelo base BRAN (a) e o modelo proposto (b). Um detalhe importante é que as alterações são realizadas no início (encoder de entrada), no meio (função de transição) e ao final (condição de parada dinâmica) de cada iteração. A CPD requer que o encoder de posição e o encoder de iteração sejam adicionados no inicio de cada iteração.

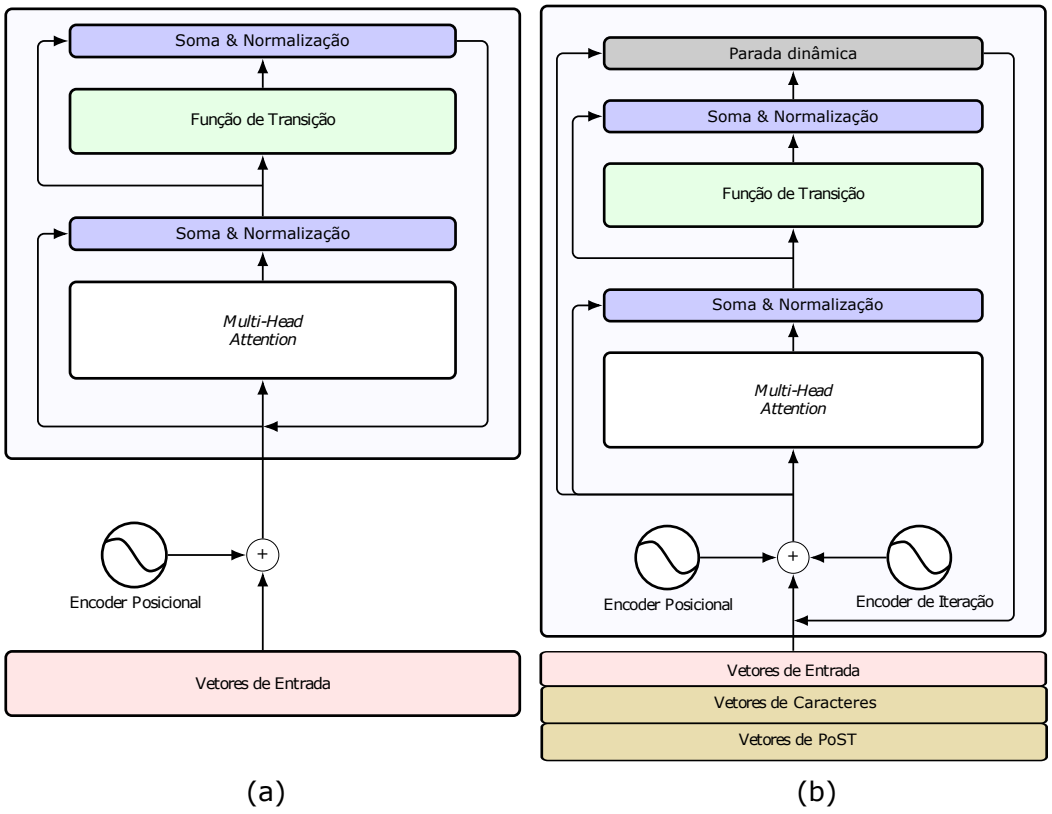


Figura 3.1: Comparação do módulos *Transformer*: modelo BRAN (a) e a Primeira abordagem (b): *Transformer* com CPD.

Encoder de Entrada: trabalhos prévios [82][100][86] mostram que a composição de *embeddings* a partir de diferentes tipos de informação linguística auxiliam na captura de características semânticas do texto, melhorando o treinamento do modelo. Para isto, as informações devem ser categóricas, sendo vetorizadas, somadas ou concatenadas a cada vetor de palavra. Algumas destas informações sãp PoS-tag (estrutura gramatical), classe de entidade (exclusivas da tarefa) e n-gram de caracteres (prefixos, sufixos, etc.).

Considera-se como entrada a sequência de I *tokens* $[w_1, ..., w_I]$, na qual cada *token* w_i tem um *embedding* atribuído $s^i \in \mathbb{R}^d$. Então, a sequência de *embeddings* de entrada é representada pela matriz $s = [s^1, ..., s^I]^\top \in \mathbb{R}^{I \times d}$. Cada *embedding* s^i é o resultado da soma de *embeddings* de todas as informações fornecidas ao modelo:

$$s^i = s^{w,i} + s^{(n-gram),i} + s^{(PoS-tag),i} + s^{(ner-tag),i} \tag{3-1}$$

onde $s^{w,i}, s^{(n-gram),i}, s^{(PoS-tag),i}, s^{(ner-tag),i} \in \mathbb{R}^d$ são *embeddings* que representam palavras, palavras baseadas em 'n-gram de caracteres', 'PoS-tag de palavras' e 'classe de entidades' respectivamente. De forma a obter $s^{(n-gram),i}$ é necessário processar previamente a sequência n-gram de caracteres. Para isto, utiliza-se uma função que sintetiza esta sequência. Por exemplo, para representar a palavra *superstar*, são utilizados marcadores <> para diferenciar os gramas de início e fim. Considerando uma divisão n-gram (n=3), a palavra é representada pela sequência de gramas: *<su, sup, upe, per, ers, rst, sta, tar, ar>*. Cada grama é representado pelo vetor $(s')^i_j$. O vetor $s^{(n-gram),i}$ é obtido calculando o máximo valor na dimensão da sequência para cada elemento do *embedding* de n-gram, como se apresenta na Eq. 3-2 e 3-3:

$$s^{(n-gram),i} = \text{FC}(\text{Max}(C_3(s'^i))) \tag{3-2}$$

$$\text{onde} \quad s'^i = [s'^i_1, ..., s'^i_G] \tag{3-3}$$

onde G é o número máximo de gramas por token. O esquema de funcionamento destas estruturas pode ser observado na Figura 3.2.

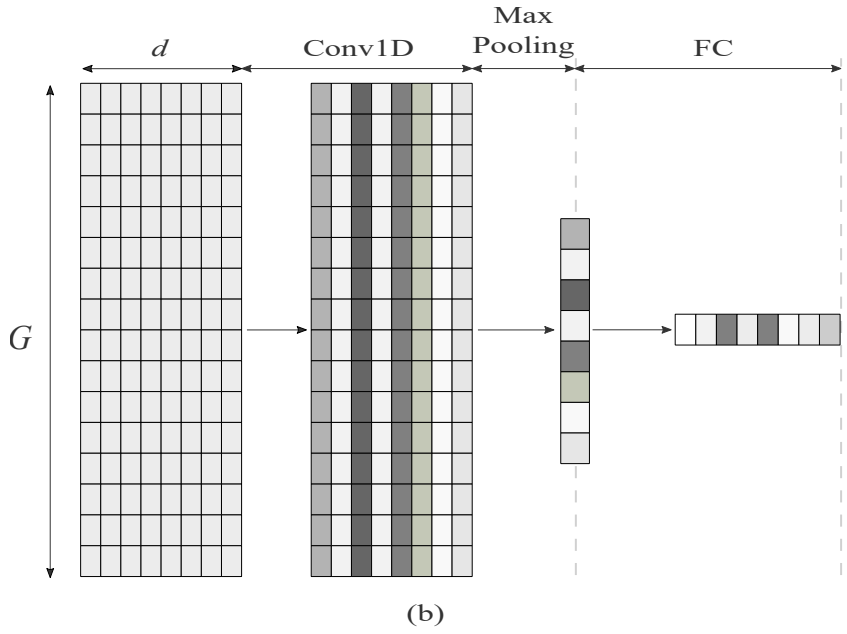


Figura 3.2: Encoder de caracteres baseado no máximo nas Eq. 3-2 e 3-3.

Estas duas alternativas serão avaliadas posteriormente no capítulo 4.

Módulo Transformer: a sequência s, saída do encoder de entrada, é processada durante T iterações pelo módulo *Transformer*. Este módulo é composto

pelo mecanismo *Scaled Dot-Product Attention* e seguido por uma função de transição. As equações do *Transformer* são apresentadas nas Eqs. (3-4) e (3-5), onde o valor inicial da matriz de estado $H_0 = s$.

$$H_t = A_t + \text{Transition}(A_t) \tag{3-4}$$

$$\text{onde} \quad A_t = \text{LN}((H_{t-1} + P_t) + \text{MultiHeadSelfAttention}(H_{t-1} + P_t)) \tag{3-5}$$

LN representa a camada *Batch Normalization* proposta por [92], $P_t \in \mathbb{R}^{I \times d}$ é uma matriz de representação de coordenadas posição-iteração proposto pelo modelo *Universal Transformer*. A diferença do modelo proposto para o modelo BRAN é a adição do número de iteração como uma variável, fato que auxilia o modelo a identificar a iteração em que ele se encontra. Na Eq. (3-6), P_t é definido em função da posição $i \in [1, I]$, iteração $t \in [1, T]$ e cada posição do *embedding* $j \in [1, d]$.

$$\begin{aligned} P_t^{i,2j} &= \sin(i/10000^{2j/d}) + \sin(t/10000^{2j/d}) \\ P_t^{i,2j+1} &= \cos(i/10000^{2j/d}) + \cos(t/10000^{2j/d}) \end{aligned} \tag{3-6}$$

A função de transição proposta é baseada no bloco residual da MobileNetV2 [93], considerando as convoluções 1D. A estrutura é composta por três camadas convolutivas, como se observa na Figura 3.3. A camada de "Expansão" permite expandir por um fator o número de características. Na prática, este fator é 4. Nesta camada não existe interação entre elementos da sequência.

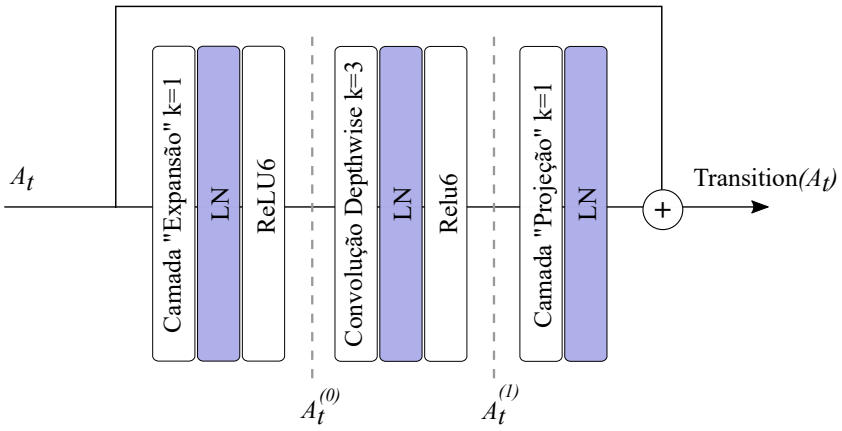


Figura 3.3: Função de Transição: Bloco convolutivo.

A convolução *depthwise* realiza a convolução ao longo da dimensão da sequência. Esta operação é realizada de forma independente para cada característica. A camada de "Projeção", permite reduzir número de características a seu valor inicial.

Entre cada camada existem as operações *Batch Normalization* e ReLU6, que regulam o sistema e tornam não linear a função de transição. A Eq. (3-7)

apresenta as operações da função de transição:

$$\begin{aligned} \text{Transition}(A_t) &= \text{LN}(C_1(A_t^{(1)})) \\ \text{onde} \quad A_t^{(1)} &= \text{ReLU6}(\text{LN}(C_3(A_t^{(0)}))) \\ A_t^{(0)} &= \text{ReLU6}(\text{LN}(C_1(A_t))) \end{aligned} \tag{3-7}$$

Condição de Parada Dinâmica (CPD): controla o número de iterações para cada elemento da sequência H_t. Este procedimento é realizado ao final de cada iteração do *Transformer*. Para isto, uma unidade neuronal com ativação sigmoide $h_{t-1}^i \in [0, 1]$ determina a saída para cada elemento i da sequência H_{t-1}.

$$h_t = \sigma(W_h H_{t-1} + b_h) \tag{3-8}$$

O valor h_t^i determina o peso p_t^i que indica a porcentagem do estado H_t, agregado para a saída final.

$$\hat{H}_T = \sum_{t=1}^{N_{max}} p_t H_t \tag{3-9}$$

A partir deste ponto, de forma a simplificar as equações, o elemento i da sequência p_t é chamado p_t^i. Cada elemento i realiza a operação de CPD de forma independente. Quando o valor acumulado de p_t^i é maior que o limiar (*threshold*) $(1 - \epsilon)$, o elemento i não realiza mais atualizações. Esta condição é validada na Eq. 3-11, sendo N^i a última iteração que cumpre a condição de parada dinâmica.

$$p_t^i = \begin{cases} h_t^i & \text{Se} \quad t < N^i \\ R_t^i & \text{Se} \quad t = N^i \\ 0 & \text{outro caso} \end{cases} \tag{3-10}$$

$$N^i = \min\{t' : \sum_{t=0}^{t'} h_t^i >= 1 - \epsilon\} \tag{3-11}$$

Após ultrapassar o limiar, o valor p_t^i é ajustado com o fator de lembrança R_t^i, o que permite seguir a definição $\sum_t^{N^i} p_t = 1$.

$$R_t^i = 1 - \sum_{i=1}^{N^i - 1} h_t^i \tag{3-12}$$

Um exemplo deste funcionamento é apresentado na Figura 3.4. No módulo CPD, os elementos comportam-se de forma independente; assim, apresenta-se o procedimento para o elemento na posição i. Quando um elemento da sequência deixa de satisfazer à condição $> \varepsilon$, utiliza-se o fator de lembrança R_t para ponderar a última atualização deste elemento.

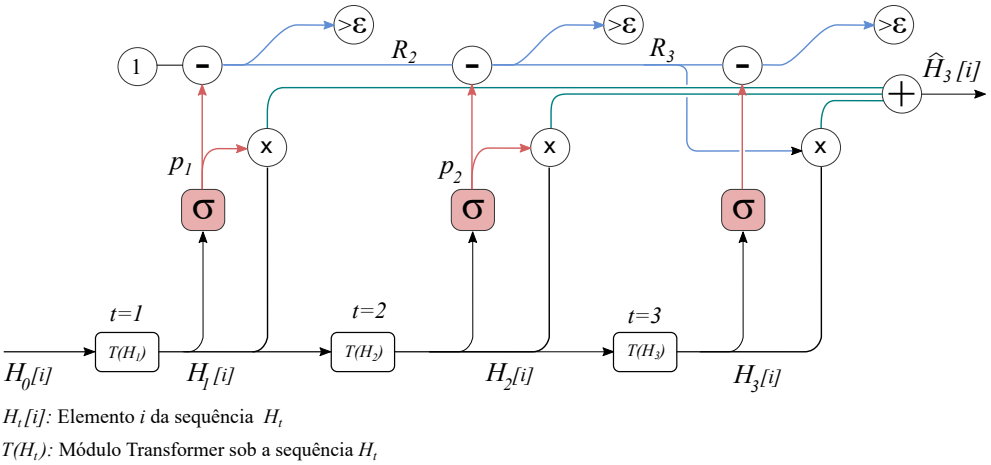


Figura 3.4: Exemplo do funcionamento do módulo de CPD para o elemento i da sequência H_t.

A cada iteração o valor de 1 inicial (na parte superior esquerda da Figura) é rebaixado pelo valor ponderado p_t e comparado com o limiar ϵ. No gráfico de exemplo, na terceira iteração esta condição é satisfeita, de forma que o fator de lembrando R_3 multiplica a última iteração.

Estas três configurações no módulo *Transformer* definem a primeira abordagem. A próxima seção detalha a segunda etapa de aprimoramento deste módulo.

3.3 Segunda Abordagem: *Neural Transformer Computer* para extração de relações (NTCRE)

O módulo *self-Attention* utilizado no modelo *Transformer* relaciona todos os elementos de uma sequência usando a função de compatibilidade. Cada elemento é obrigado a receber informações de todos os elementos da sequência de forma proporcional à compatibilidade determinada pelos pesos de *Attention*. Um enfoque mais sofisticado e interessante consiste em dispor de um mecanismo capaz de saber quando desvincular-se de certa informação ou utilizá-la em várias etapas da recorrência, especialmente no caso de sequências muito longas, dado que a complexidade do problema aumenta. Esta abordagem propõe prover a rede neural de um sistema de raciocínio antes de decidir como interagir com a informação compartilhada. Isto se torna possível pelo emprego de um controlador Neural que decide o destino da informação compartilhada para cada elemento da sequência. O controlador neural é composto por um Módulo de Controle de Acesso a Memória (MCAM) dentro da estrutura *Transformer* descrita na seçao 3.2. O MCAM processa as representações

$H_t \in \mathbb{R}^{I \times d}$ da seguinte forma:

$$H_t = \mathrm{LN}(A_t + \mathrm{Transition}(A_t)) \tag{3-13}$$

$$\text{onde} \quad A_t = \mathrm{LN}((H_{t-1} + P_t) + \mathrm{MCAM}(H_{t-1} + P_t)) \tag{3-14}$$

em que a função de transição na Eq. (3-13) e a matriz P_t são as mesmas da primeira abordagem. Uma comparação entre a estrutura da primeira e segunda abordagens é apresentada na Figura 3.5:

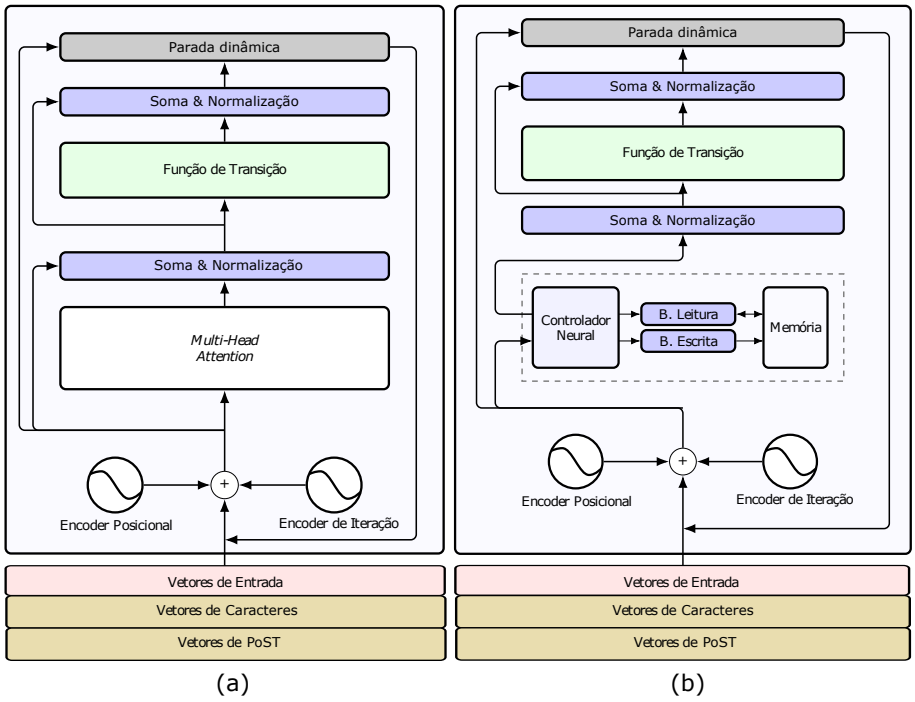


Figura 3.5: Comparação de modelos *Transformers*: Primeira abordagem (a) e Segunda abordagem (b).

No MCAM, a sequência de entrada é processada pela camada *Multi-Head Attention* e se utiliza a saída como vetor de interface para o Mecanismo de Acesso à Memória (MAM). Isto permite a manipulação dos vetores de *Attention* ao longo da recorrência. O MAM utiliza os pesos de escrita e leitura para calcular o vetor de pré-saída e os vetores de leitura, similarmente ao modelo DNC; a seguir são somados e transformados por uma camada *Full-Connected* (FC). O diagrama da estrutura proposta é apresentado na Figura 3.6.

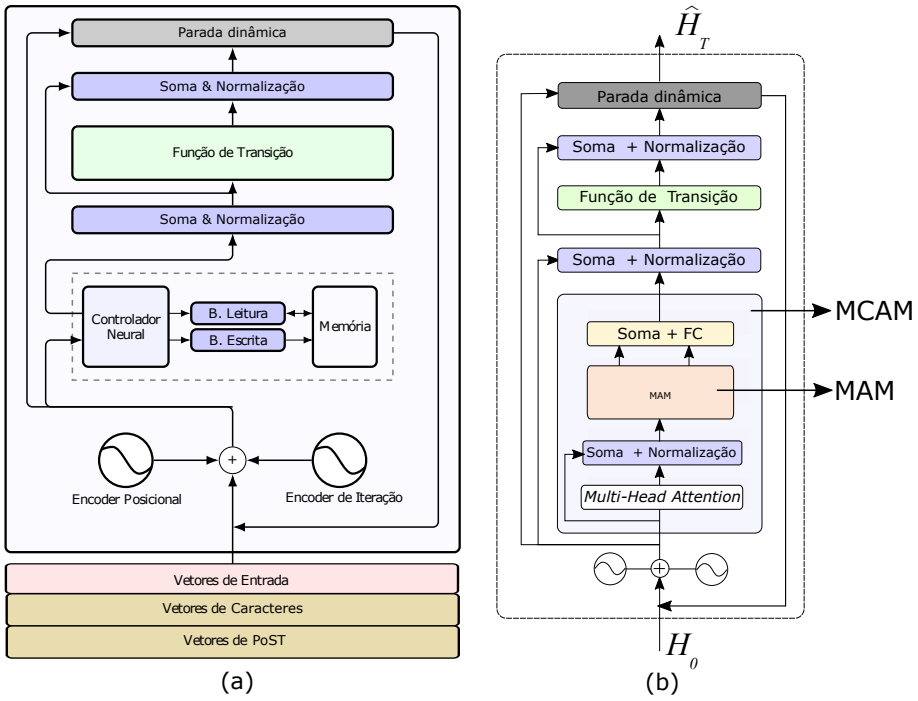


Figura 3.6: MCAM e MAM do modelo *Transformer*. (a) Diagrama geral do funcionamento do controlador de memória dentro do modelo *Transformer*. (b) Estruturação do MCAM e MAM que integram a estrutura *Multi-Head Attention* e o controlador DNC.

Na próxima seção descreve-se o funcionamento do módulo de controle de acesso à memória.

Módulo de Controle de Acesso à Memória (MCAM)

Este mecanismo realiza operações de escrita e leitura sobre uma memória externa para processar a sequência de entrada. Neste trabalho simplificam-se as operações de escrita e leitura da seção 2.8.2, reduzindo o tempo de acesso à memória e tornando possível o paralelismo ao longo da sequência. A cada iteração do módulo *Transformer*, o MCAM processa a sequência $(H_{t-1}+P_t)$ de forma paralela. O módulo *Multi-Head Attention* com conexão residual calcula a matriz de *Attention* A_t. Para cada elemento i, o vetor de *Attention* $A_t[i]$ conteém toda informação necessária para calcular o vetor de pré-saída $\nu_t^i \in \mathbb{R}^d$ e o vetor de interface $\xi_t^i \in \mathbb{R}^{4+2K+2R+2W+R\times W}$. Isto é apresentado na Eq. (3-15), onde R é o número de barramentos para leitura, W é o tamanho da palavra na memória e K é o tamanho da chave de escrita.

$$\begin{aligned} A_t &= \mathrm{LN}((H_{t-1}+P_t)+\mathrm{MultiHeadAttention}(H_{t-1}+P_t)) \\ \boldsymbol{\nu}_t &= W_\nu A_t, \quad \boldsymbol{\xi}_t = W_\xi A_t \end{aligned} \tag{3-15}$$

onde $\boldsymbol{\nu}_t$ e $\boldsymbol{\xi}_t$ são sequências de vetores de pré-saída e de interface respectivamente, como apresentado na Eq. 3-16.

$$\boldsymbol{\nu} = [\nu_t^1, \nu_t^2, ..., \nu_t^I] \quad \boldsymbol{\xi} = [\xi_t^1, \xi_t^2, ..., \xi_t^I] \tag{3-16}$$

A sequência de vetores de pré-saída $\boldsymbol{\nu}_t$, juntamente com a sequência de R vetores de leitura processados pelo MAM, processam a saída do MCAM na Eq. 3-17:

$$\text{MCAM}(H_{t-1} + P_t)[i] = \boldsymbol{\nu}_t^i + W_r[\boldsymbol{r}_t^{1,i}; ...; \boldsymbol{r}_t^{R,i}], \tag{3-17}$$

Mecanismo Acesso à Memória (MAM)

Segundo o modelo apresentado na seção 2.8.2, para cada elemento da sequência o vetor de interface correspondente ξ_t^i é subdividido e transformado de forma a assegurar a permanência dos parâmetros no domínio correto. Assim, as operações de controle de acesso à memória são realizadas mediante a atualização descrita na Eq. 2-24. O peso de leitura é calculado para cada elemento da sequência da mesma forma que no modelo DNC descrito na seção 2.8.2. O cálculo de peso de escrita demanda operações de alta custo computacional, pelo qual esta etapa é analisada detalhadamente.

Peso de Escrita: para o cálculo do peso de escrita, a porta de alocação $g_t^{a,i}$ seleciona dentre os tipos de endereçamento a alocação de memória dinâmica paralela (a_t^i) e o endereçamento baseado no conteúdo ($c_t^{w,i}$). A alocação dinâmica tradicional descrita na seção 2.8.2 realiza uma operação de ordem entre elementos da sequência que é altamente sequencial e independente entre os elementos da sequência. Isto torna os elementos ineficientes para aproveitar o paralelismo SIMD (*Single Instruction Multiple Data*) das GPUs. Este trabalho propõe um mecanismo de alocação de memória dinâmica altamente paralelizável que aproveita o paralelismo das GPUs. Os detalhes da implementação são descritos a seguir.

Alocação de memória dinâmica paralela (a_t^i): um problema presente no mecanismo de alocação de memória dinâmica, apresentado na seção 2.8.2, é a complexidade computacional. Para calcular a_t, são necessárias operações de ordem e indexação. Isto se torna computacionalmente inviável quando o processo é estendido para longas sequências. De forma a resolver essa desvantagem, este trabalho propõe um conjunto de transformações que simulam o mecanismo original, com a diferença de que este modelo apresenta um alto grau de paralelismo. Dependendo do tamanho da sequência, pode-se obter velocidades 15 vezes maiores do que com o mecanismo original. Os detalhes da análise de desempenho são apresentados na seção 4.5.1.

Primeiramente, calcula-se a lista de uso u_t^i da mesma forma que no modelo DNC:

$$\varphi_t^i = \Pi_{j=1}^{R}(1 - f_t^j w_{t-1}^{r,j,i}) \tag{3-18}$$

$$u_t^i = (u_{t-1}^i + (1 - u_{t-1}^i) w_{t-1}^{w,i}) \odot \varphi_t \tag{3-19}$$

Na seção 2.8.2, foi mostrado que o peso de alocação em cada endereço de memória $a_t^i[n]$ do modelo DNC é calculado a partir do valor de uso nessa posição $u_t[n]$ e posições acessadas mais recentemente $u_t[n'] > u_t[n]$. A abordagem proposta neste trabalho consiste em calcular os valores de uso $u_t[n']$ por meio de uma máscara binária $g_t \in [0,1]^{N \times N}$ que seleciona as posições da memória com valores de uso $u_t[n'] > u_t[n]$:

$$g_t^i[n][n'] = \begin{cases} 1 & u_t^{\epsilon,i}[n] > u_t^{\epsilon,i}[n'] \\ 0 & \text{outro caso} \end{cases} \tag{3-20}$$

$$\text{onde} \quad u_t^{\epsilon,i}[n] = u_t^i[n] + n\epsilon \tag{3-21}$$

Para o cálculo de g_t^i, adiciona-se a cada valor do vetor $u_t^i[n]$ um fator $n\epsilon$ onde $\epsilon \approx 1e{-}6$, de forma a evitar posições com valores de uso iguais ou zerados. Isto é similar ao comportamento da operação *Sorted*, no modelo DNC, que aumenta ordenadamente a prioridade a cada posição da memória, em se tratando de valores iguais. A matriz ϕ_t contém os valores de uso $u_t[n'] > u_t[n]$ que devem ser multiplicados da mesma forma que no modelo DNC. Assim, o vetor de alocação a_t^i é calculado na Eq. (3-23).

$$\phi_t^i = 1 - g_t^i + g_t^i \odot u_t^i \tag{3-22}$$

$$a_t^i = (1 - u_t^i) \prod_{n}^{N} \phi_t^i[:, n] \tag{3-23}$$

Todas as equações são transformações lineares nas quais o custo computacional é inferior ao do modelo DNC. Para um elemento i da sequência, a Figura 3.7 mostra passo a passo o cálculo do vetor de alocação. No exemplo, a lista de uso de memória $u_t^i = [0.1, 0.9, 0, 0.8]$ contém uma posição de memória vazia (com valor 0), detectada pelo mecanismo de alocação dinâmica paralela $a_t = [0, 0, 1, 0]$. A máscara binária g_t^i é o elemento principal que filtra diretamente os valores $u_t^i[n']$ necessários para os cálculos em cada posição de memória sem necessidade de reindexação (realizada na função *Sorted* do modelo DNC).

$$g_t^i[n][n'] = \begin{cases} 1 & u_t^{\epsilon,i}[n] > u_t^{\epsilon,i}[n'] \\ 0 & \text{outro caso} \end{cases}$$

$$\phi_t^i = 1 - g_t^i + g_t^i \odot u_t^i$$

$$a_t^i = (1 - u_t^i) \prod_n^N \phi_t^i[:, n]$$

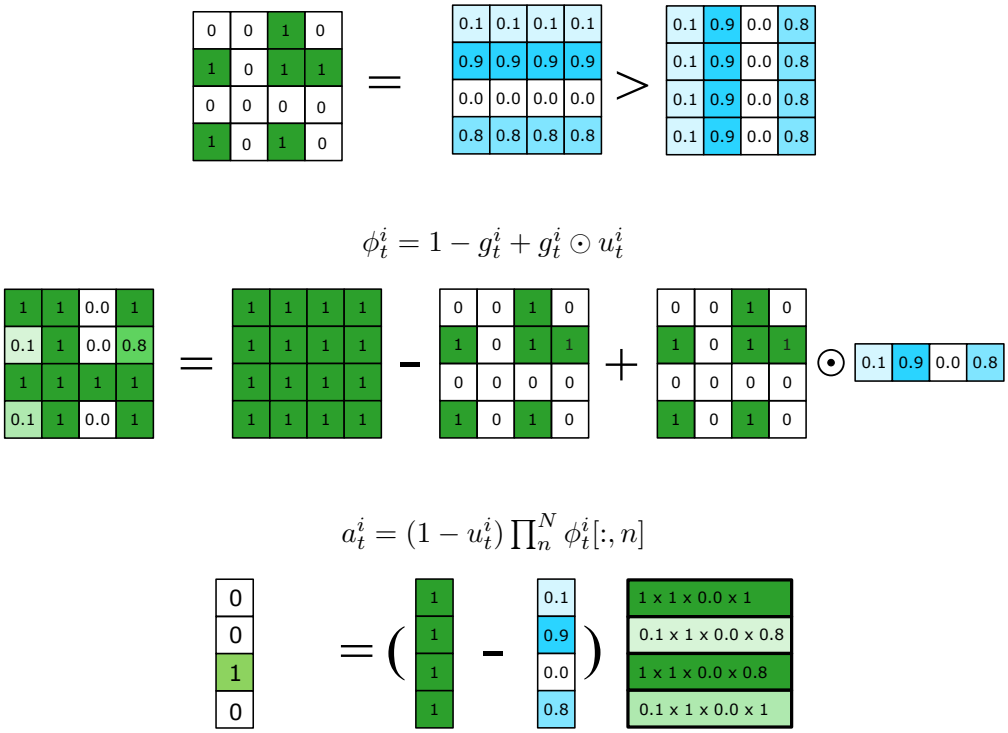


Figura 3.7: Proceso de cálculo do vetor de alocação.

Este processo pode ser facilmente implementado. O código do algoritmo em tensorflow é apresentado no Anexo A.1.

Endereçamento baseado no conteúdo($c_t^{w,i}$): é calculado na Eq. (3-24), usando a função de compatibilidade definida na Eq. (2-25).

$$c_t^{w,i} = C(k_t^{w,i}, W_w M_{t-1}^i, \beta_t^{w,i}) \tag{3-24}$$

Assim, o peso de escrita $w_t^{w,i}$ é calculado na Eq. (3-25). A porta de alocação $g_t^{a,i}$ seleciona o tipo de endereçamento e a chave de escrita $g_t^{w,i}$ habilita realizar a escrita em memória.

$$w_t^{w,i} = g_t^{w,i}[g_t^{a,i} a_t^{k,i} + (1 - g_t^{a,i}) c_t^{w,i}] \tag{3-25}$$

3.4 Extração de Relações e Reconhecimento de Entidades Nomeadas

Para a tarefa de ER e REN, as transformações realizadas na sequência H_T de forma a a calcular o $score(p_{head}, p_{tail})$ e C para as tarefas de ER e REN foram abordadas na seção 2.8.3.

4
Experimentos

A parte experimental apresentada neste capítulo foi desenvolvida com o objetivo de avaliar o desempenho do modelo aqui proposto. Para tal, foram executados experimentos comparativos entre as abordagens propostas e os modelos com melhor resultado registrado em cada estudo de caso.

Foram escolhidos três estudos de caso para a avaliação dos modelos. O primeiro e o segundo estudo de caso são tarefas de extração de relações em resumos de artigos médicos (Biocreative V e CTD + PubMed). O terceiro estudo de caso trata a extração de relações em resumos de artigos científicos (SemEval 2018). Nos dois primeiros, foca-se em analisar relações em nível de documento; ou seja, necessita-se ler o contexto das frases para identificar e categorizar as relações. Para isto, anotações de entidades nomeadas estão disponíveis. No terceiro estudo de caso, as relações podem ser identificadas em nível de uma única frase. As entidades das frases podem ser identificadas, mas não categorizadas.

Para todos os estudos de caso, o processo de tokenização foi realizado utilizando um tokenizador simples da biblioteca spaCy[1] para separar as palavras em blocos de letras, números e pontuações localizadas no final das frases. Durante este processo, considerou-se qualquer outro caractere especial como uma separação e o modelo aprendeu com as conexões entre as palavras compostas. Esta mesma biblioteca foi utilizada para realizar o processo de PoS-Tagging. Outro pré-processamento foi a criação de exemplos negativos. Para isso, os pares de entidades sem relações anotadas no documento foram catalogados como NULL.

Em todos os estudos de caso, os experimentos foram avaliados usando as métricas:

- Precisão (P): fração de exemplos da classe X do conjunto de exemplos classificados como classe X.
- Revocação ou *Recall* (R): fração de exemplos da classe X classificados corretamente a partir do conjunto de exemplos da classe X.
- F1-score (F1): Combina precisão e recall de forma a se ter um único número que represente a qualidade do modelo.

[1] `https://spacy.io/`

Estas métricas são baseadas na quantidade de Verdadeiros Positivos (TP), Falsos Positivos (FP) e Falsos Negativos (FN):

$$F1 = \frac{2PR}{P+R} \quad \text{onde:} \quad P = \frac{TP}{TP+FP} \quad R = \frac{TP}{TP+FN} \tag{4-1}$$

O segundo e terceiro estudos de caso (problemas multiclasse) apresentam uma distribuição de dados por classe altamente desbalanceada. Foi utilizada uma função de custo ponderada para balancear o aprendizado entre as classes. O peso para cada classe é determinado pela expressão:

$$w_{i \in C} = \frac{\#\text{ total de exemplos}}{\#\text{ classes} \times \#\text{ exemplos da classe } i} \tag{4-2}$$

onde C é o conjunto de classes de relações. Definidas as métricas gerais para os estudos de caso, a configuração dos modelos propostos é apresentada a seguir.

4.1 Configuração dos modelos

Parâmetros	**Valor**	**Intervalo**
Gerais		
Batch size	32	16 - 64
Peso relações positivas/negativas	50/50	-
Otimização		
Taxa de aprendizado	0.0005	0.0001 - 0.01
Corte de gradiente	3	-
Beta (Regularização)	0.0001	0 - 0.01
Camada de Entrada		
Todos os *embeddings*	300	32 - 300
Tamanho n-gram de caracteres	3	-
Transformer		
Dropout interno e final	0.15	0 - 0.3
Iterações	3	2 - 6
Número de cabeças de *Attention*	4	-

Tabela 4.1: Configuração geral dos modelos UTRE e NTCRE para os experimentos.

O modelo proposto utiliza vários tipos de hiperparâmetros. Convenientemente, a bibliografia recomenda vários critérios. Os valores de configuração para os modelos UTRE (abordagem 1) e NTCRE (abordagem 2) apresentados

na Tabela 4.1 são os melhores obtidos ao longo de todos os experimentos para os dois estudos de caso.

O peso de relações positivas/negativas indica a chance de treinar um *batch* de dados com presença de relações (positivas) ou sem nenhuma relação (negativas). O corte do gradiente é um fator importante para evitar mudanças abruptas na atualização dos pesos da rede. Para simplificar o modelo, todos os tipos de *embeddings* utilizam a mesma dimensão na camada de entrada. O *dropout* interno necessita de um valor baixo, dado que é utilizado na recorrência do *transformer*. O *dropout* final é utilizado na operação bilinear para a tarefa de extração de relação. O NTCRE utiliza um conjunto de hiperparâmetros que definem o módulo de acesso à memória. Estes parâmetros são apresentados na Tabela 4.2. Em todos os experimentos, o tamanho da palavra foi igual ao do *embedding*. O tamanho da memória foi considerado igual a 4, dado que o módulo *Transformer* não realiza muitas iterações. Isto permitiu que uma "pequena" memória fosse utilizada para cada elemento da sequência.

MCAM	
Tamanho da memória	4
Tamanho da palavra	300
Barramento de escrita	1
Barramento de leitura	2
Beta-1 (Regularização *hard sigmoid*)	0.00001

Tabela 4.2: Configuração do módulo MCAM para os modelos UTRE e NTCRE.

O modelo foi projetado para realizar uma única escrita em memória a cada iteração. O módulo de condição de parada dinâmica foi configurado apenas com os parâmetros recomendados pela literatura.

Condição de Parada Dinâmica	
Limiar de ativação $(1 - \epsilon)$	0.99
Beta-2 (Regularização)	0.00001

Tabela 4.3: Configuração do módulo CPD para os modelos UTRE e NTCRE.

No primeiro estudo de caso, foi necessário configurar o peso para cada tarefa do treinamento multi-tarefa. A configuração 50/50 gerou os melhores resultados. Para todos os estudos de caso foram analisados os tempos de processamento com a finalidade de se comparar o custo computacional para cada abordagem. Assim, os tempos de processamento de todos os experimentos foram compuatdos em um PC com a seguinte configuração básica:

– Processador: Intel (R) Xeon (R) CPU E5-2660 2.00Ghz
– Capacidade de Memória RAM: 32GB
– Placa Gráfica: V100 de 16GB (arquitetura Volta)
– Sistema Operacional: Ubuntu 16.4

4.2 Extração de relações em resumos de artigos médicos Biocreative V

O enorme volume de documentos da área de biomedicina, oriundo do crescente índice de publicações científicas, é uma das motivações mais comuns para a automatização da extração de conhecimento. Assim, este estudo de caso tem como foco a tarefa de entender as relações entre produtos químicos e doenças, com aplicações em muitas áreas de pesquisa biomédica e de saúde, incluindo estudos de toxicologia, descoberta de novas drogas, etc. A importância desta tarefa é evidente pelo fato de que produtos químicos, doenças e suas relações estão entre os tópicos mais buscados pelos usuários de PubMed [94].

4.2.1 Descrição da tarefa

As relações do tipo "induz" entre doenças e químicos são tipicamente determinadas em nível de documento. Quer dizer, podem ser expressas através de várias sentenças. A tarefa de extração automática de relações entre químicos e doenças (*Chemical Disease Relation* - CDR) subdivide-se em duas tarefas:

1. Reconhecimento de entidades nomeadas (REN): o conjunto de dados é formado por resumos de artigos PubMed através dos quais pode-se localizar e classificar as entidades *Chemical* (químico) e *Disease* (doença).

2. Extração de relações (ER): o conjunto de dados é o mesmo que o utilizado na tarefa de REN e pode-se retornar uma lista de pares de entidades *Chemical-Disease* associados a um tipo de interação (*Chemical Induced Disease* - CID).

Neste estudo de caso, a ER é a principal tarefa de interesse. Para isto, utiliza-se a tarefa REN para auxiliar no treinamento da tarefa de ER durante o treinamento multitarefa. Os resultados dos modelos propostos serão comparados com o modelo base BRAN. Assim, este estudo de caso apresenta as seguintes características:

– Dados anotados limpos: os dados foram anotados e revisados manualmente por especialistas.

– Dados adicionais: anotados automaticamente (de forma a serem comparados ao modelo base BRAN).

– Classificação de relação binária: tarefa de extração de relação que considera apenas um único tipo de relação.

– Multi-tarefa: modelo treinado simultaneamente para as tarefas de REN e ER.

4.2.2 Conjunto de dados

O corpus foi construído a partir do conjunto de Dados Toxicogenômicos Comparativos (*Comparative Toxicogenomics Dataset* - CTD), com informação referente a interações entre *genes*, *químicos* e *doenças*. O conjunto de dados Biocreative V foi formado por 1500 resumos: 500 para treinamento, 500 para validação e 500 para teste. Foram acrescentados dados ruidosos (criados automaticamente) a partir do conjunto de dados CTD e artigos do PubMed. Os detalhes são apresentados na Tabela 4.4.

Tipo de dado	Documentos	Exemplos Positivos	Exemplos Negativos
Train	500	1038	4280
Valid	500	1012	4136
Test	500	1066	4270
CTD(+Data)	15,448	26,657	146,057

Tabela 4.4: Distribuição de dados para treinamento, validação, teste e dados adicionais partindo do conjunto de dados CTD.

As entidades (*químico* e *doença*) foram manualmente anotadas usando o identificador de conceitos *Medical Subject Headings*(MeSH)[1], um código único que permite identificar as entidades. A Figura 4.1 apresenta o formato de anotação do documento. A primeira linha mostra o título e a segunda, o resumo. Os separadores |t| e |a| são usados para diferenciar o código do documento do título e do resumo respectivamente. As linhas seguintes são denominadas menções. Cada menção é composta por seis atributos: PMID (PubMed ID), OFFSET INICIAL, OFFSET FINAL, MENÇÃO, TIPO DE MENÇÃO, IDENTIFICADOR. O OFFSET INICIAL e OFFSET FINAL indicam o índice do primeiro e último caractere da menção, respectivamente.

[1] `https://www.nlm.nih.gov/mesh/meshhome.html`

```
16274958|t|Recurrent dysphonia and acitretin.
16274958|a|We report the case of a woman complaining
of dysphonia while she was treated by acitretin. Her
symptoms totally regressed after drug withdrawal and
reappeared when acitretin was reintroduced. To our
knowledge, this is the first case of acitretin-induced
dysphonia. This effect may be related to the
pharmacological effect of this drug on mucous membranes.
16274958    10  19  dysphonia   Disease D055154
16274958    24  33  acitretin   Chemical    D017255
16274958    80  89  dysphonia   Disease D055154
16274958    115 124 acitretin   Chemical    D017255
16274958    199 208 acitretin   Chemical    D017255
16274958    271 280 acitretin   Chemical    D017255
16274958    289 298 dysphonia   Disease D055154
16274958    CID D017255 D055154
```

Figura 4.1: Um exemplo de anotação do corpus CDR.

A relação CID é anotada em nível de documento. Cada anotação de relação contém quatro atributos: PMID, TIPO DE RELAÇÃO, IDENTIFICADOR QUÍMICO, IDENTIFICADOR DOENÇA.

4.2.3 Pré-processamento

Nesta etapa, o texto foi tokenizado e os casos de hiperonímia foram tratados. Para isto, foi utilizado o vocabulário de hierarquia de *Medical Subject Headings* (MeSH)[3]. Assim, cada entidade localizada no vocabulário de hierarquia foi substituído pelo máximo superior no árvore de hierarquia.

4.2.4 Resultados

Na Tabela 4.5, as duas abordagens propostas são comparadas com os melhores resultados na literatura para o conjunto de dados. O modelo de Gu et al. [95] utilizou um modelo CNN, enquanto Zhou [96] usou uma rede LSTM. O modelo proposto por Verga et al. [80] foi baseado no módulo *Transformer* e Mandya et al. [97] fez uso de um híbrido entre CNN e LSTM. O trabalho de Nguyen et al. [98] explorou a utilização de *embeddings* de palavras baseado em caracteres sobre uma CNN. O modelo BRAN apresentou resultados para quatro configurações de treinamento. O modelo normal usou a versão "ensemble" de vários modelos treinados e acrescentou um conjunto de dados adicionais (+Data) para estes dois caso. O modelo proposto, quando comparado aos outros modelos, obteve os melhores resultados, tanto para o modelo UTRE como para NTCRE. Pode-se observar que a utilização de dados ruidosos (+Data) auxilia no aprendizado.

[3] `https://meshb.nlm.nih.gov/treeView`

Modelo	Precisão	Recall	F1-score
(CNN) Gu et al. (2016)	62,0	55,1	58,3
(LSTM) Zhou et al. (2016)	55,6	68,4	61,3
(CNN) Gu et al. (2017)	55,7	68,1	61,3
(BRAN) Verga et al. (2018)	55,6	70,8	62,1
+Data	64,0	69,2	66,2
(BRAN) Verga et al. (2018) (ensemble)	63,3	67,1	65,1
+Data	65,4	71,8	68,4
(CNN + LSTM) Nguyen et al. (2018)	71,1	72,6	71,8
(CNN + Emb) Mandya et al. (2018)	69,0	70,0	69,0
UTRE-CPD	71,4	74,5	**72,9**
+Data	75,6	71,6	**73,5**
NTCRE-CPD	73,3	74,9	**74,1**
+Data	80,3	71,5	**75,6**

Tabela 4.5: Resultados da avaliação do estudo de caso Biocreative V.

Na Tabela 4.6 são apresentados os resultados do treinamento para cada melhora realizada no modelo base BRAN.

Config.	**Mem.**	**n-gram**	**Trans.**	**CPD**	**Treino**	**Valid.**	**Teste**
1					76,6	71,0	68,0
2				x	75,8	72,9	72,5
3			x		75,5	72,8	72,5
4			x	x	78,5	73,1	72,7
5		x			77,4	70,9	69,8
6		x		x	76,9	72,0	71,8
7		x	x		76,1	73,4	72,7
UTRE-CPD		x	x	x	**78,7**	**73,8**	**72,9**
9	x			x	81,3	72,9	72,6
10	x		x	x	80,8	73,9	73,5
11	x	x		x	81,9	73,1	72,9
NTCRE-CPD	x	x	x	x	**81,9**	**74,8**	**74,1**

Tabela 4.6: Resultados F1-score para diferentes configurações do modelo proposto.

A coluna Mem. indica se é a primeira abordagem (sem memória) ou a segunda (com memória). A coluna n-gram indica se o modelo utilizou *embeddings* de n-gram de caracteres. A coluna Trans. indica se o modelo utilizou a função de transição proposta e a coluna CPD indica se o modelo

utilizou a condição de parada dinâmica. As melhores configurações escolhidas para comparar com o modelo base BRAN foram obtidas observando-se as maiores acurácias de validação da Tabela 4.6. Os resultados basearam-se na média de 30 repetições do experimento.

Dos resultados obtidos, pode-se observar que os modelos com memória conseguem uma acurácia consideravelmente maior quando se trata dos dados de treino, ultrapassando 80%. A função de transição e o módulo CPD são os módulos que mais melhoram desempenho do modelo (em mais de 3% da configuração 1 para as configurações 2 e 3). Os valores em negrito correspondem às maiores acurácias para cada abordagem (com e sem memória). Um fator relevante na configuração 11 e no NTCRE-CPD foi a utilização da função de transição, que causou uma melhora de 1,7% nos resultados de validação. Pode-se observar que cada novo módulo adicionado melhora o desempenho do modelo. Todos os testes realizados com memória utilizam o módulo CPD (considerado como parte do sistema de gerenciamento de memória).

Teste de significância: de forma a avaliar a significância estatística dos resultados, foi realizado o teste não paramétrico de Wilcoxon. Nesta análise, a hipótese considera que as distribuições de onde os resultados (F1-Score) dos dois sistemas foram amostrados são idênticas. Em outras palavras, não seria possível diferenciar qual dos dois modelos gerou esses resultados. Por convenção, a hipótese nula é rejeitada para valores de p (p-valor) menores ou iguais a 0,05. Na tabela 4.7 pode-se observar os resultados do teste de Wilcoxon:

Algoritmo 1	Algoritmo 2	p
UTRE-CPD	NTCRE-CPD	0,07652
UTRE-CPD	BRAN	0,00038
NTCRE-CPD	BRAN	0,00052*

Tabela 4.7: Teste de significância entre as configurações dos modelos propostos e o modelo BRAN.

A comparação de ambos os modelos propostos com o modelo BRAN conseguem rejeitar a hipótese nula. Por outro lado, a comparação entre os modelos propostos não consegue rejeitar a hipótese, dado que o p-value é 0,07652. Este valor é muito próximo do p-valor de rejeição, o que permite dar uma noção da significância estatística dos resultados.

4.2.5 Discussão de resultados do estudo de caso

De todos os experimentos realizados, a configuração que apresentou melhor desempenho foi NTCRE-CPD (segunda abordagem), conseguindo obter o maior F1-score e a maior precisão. Os diferentes experimentos realizados e apresentados na Tabela 4.6 permitiram quantificar as melhoras progressivas realizadas no modelo. O teste de significância estatística foi utilizado de forma a comparar os resultados com o modelo BRAN e como resultado a hipótese nula foi rejeitada. Assim, um ganho na acurácia foi obtido com todos as melhoras acrescentadas. Os experimentos realizados com o modelo base BRAN indicam uma média de mais de 50000 iterações para a convergência do modelo[1], valor drasticamente reduzido para 5000 iterações com o modelo proposto. Os resultados da tarefa REN são apresentados no Anexo B.1.

Para fins do modelo proposto, é interessante observar o funcionamento do módulo MCAM do modelo NTCRE. Como ocorre a realização do processo de escrita e leitura na memória? Ele realmente funciona? Particularmente, para este estudo de caso foi realizado uma visualização do MCAM com a finalidade de fornecer uma melhor compreensão do que acontece na rede neural.

Funcionamento do módulo de controle de acesso à memória

Considera-se que cada elemento da sequência compartilha os pesos que processam o vetor de interface do controlador do MCAM. De forma a visualizar o funcionamento deste mecanismo, foram coletados tensores de representação durante uma inferência. O modelo NTCRE foi configurado para realizar no máximo três iterações durante a inferência. Foram escolhidos dois elementos da sequência de entrada; o primeiro foi classificado como Doença e o outro como Null. A Figura 4.2 apresenta o funcionamento do mecanismo para os dois casos.

[1] https://github.com/patverga/bran

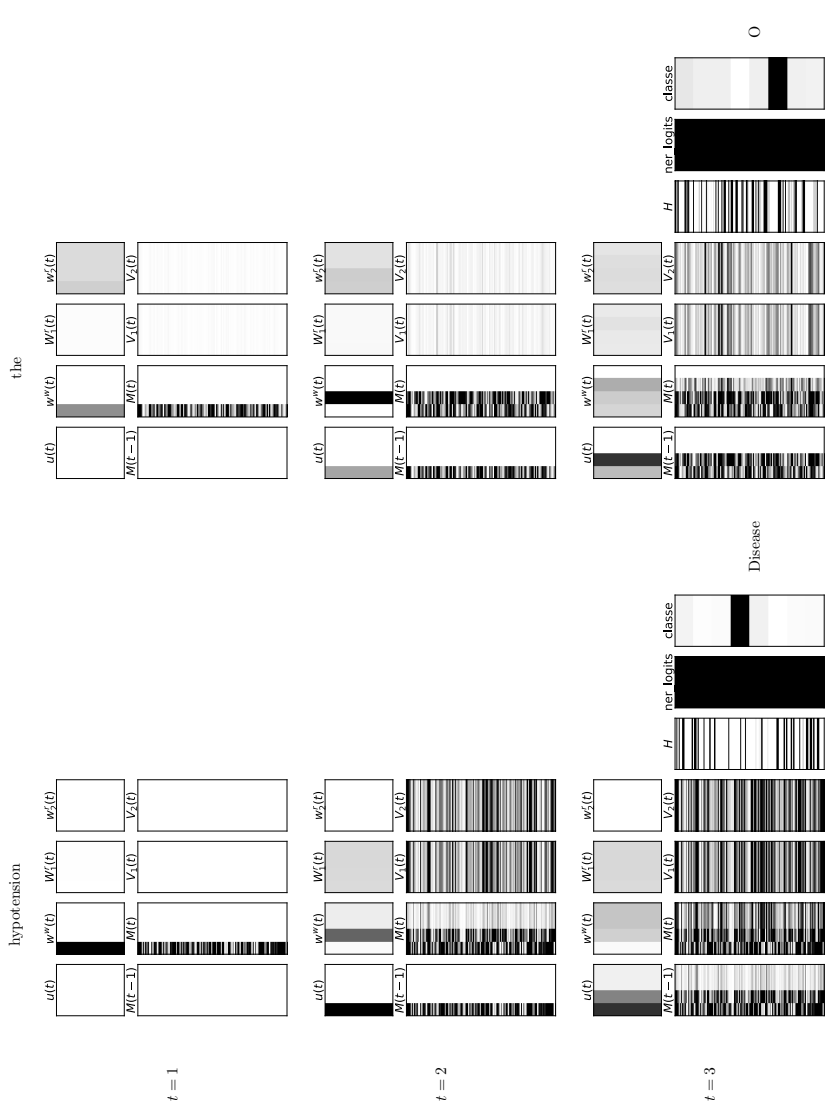


Figura 4.2: Funcionamento do MCAM: inferência de 2 elementos de uma sequência.

- $u(t)$: Lista de posições de memória utilizadas na iteração t.
- $M(t)$: Estado da memória externa na iteração t.
- $M(t-1)$: Estado da memória externa na iteração anterior $t-1$.
- $w^w(t)$: Peso de escrita na iteração t.
- $w_1^r(t)$: Peso de Leitura 1 na iteração t.
- $w_2^r(t)$: Peso de Leitura 2 na iteração t.
- $V_1(t)$: Primeiro valor lido da memória na iteração t..
- $V_2(t)$: Segundo valor lido da memória na iteração t..
- H: Saída do módulo *Transformer*.
- ner_logits e classes: Energia e saída de Probabilidades respectivamente.

No primeiro caso, o MCAM prevê a classe de entidade Doença (*Disease*) para a palavra *hypotension*. Pode-se observar que, na primeira iteração ($t = 1$), o MCAM realiza uma escrita em memória usando alocação dinâmica paralela, dado que escreve em uma região específica da memória (observe o vetor de escrita $w^w(t)$ e a transição da memória de $M(t-1)$ a $M(t)$). Na segunda e terceira iterações, o MCAM realiza escritas usando um endereçamento baseado em conteúdo. Ele consegue preencher a maior parte da memória e os vetores de leitura coletam a informação necessária para o transporte para a camada de encoder.

No segundo caso, a palavra *the* deve ser categorizada como Null (O). Pode-se observar que o MCAM realiza três escritas usando a alocação dinâmica paralela, visto que preenche apenas as três primeiras posições da memória de forma precisa. Os valores de leitura permitem extrair os valores necessários para classificar a palavra corretamente.

O processo foi descrito para dois elementos da sequência. O funcionamento do MCAM para a sequência completa pode ser observado na Figura 4.3. Na parte superior, observam-se 4 blocos de sequência em 3 etapas ($t = 1, 2, 3$). Os dois elementos analisados previamente são representados por duas linhas vermelhas.

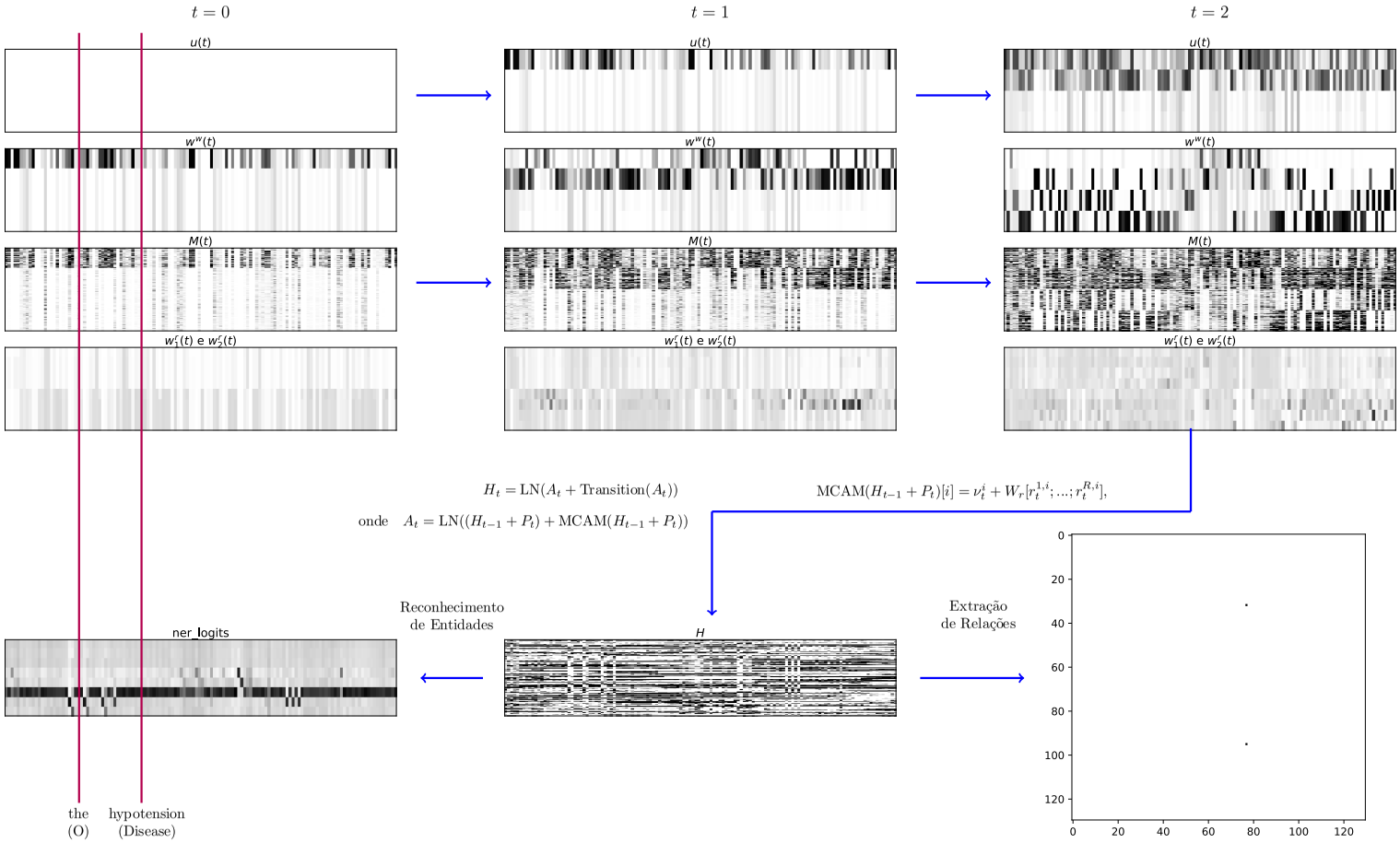


Figura 4.3: Funcionamento do MCAM: Inferência de 2 elementos de uma sequência.

4.3 Extração de relações em resumos de artigos médicos: conjunto de dados baseado em dados Toxicogenómicos Comparativos e PubMed

O conjunto de dados, proposto por Verga [80], foi construído a partir do conjunto de dados Toxicogenómicos Comparativos (*Comparative Toxicogenomics Database - CTD*) e artigos médicos de PubMed. A base de dados CTD contém informação de interações entre genes, substâncias químicas e doenças. Este conjunto de dados foi construído mediante o seguinte procedimento: cada relação da base de dados CTD está associada a um par de entidades e, em cada artigo de PubMed onde o par de entidades foi observado, a relação foi associada automaticamente. Este tipo de anotação automática (dados ruidosos) é chamada de supervisionamento à distância. Da mesma forma que no estudo de caso Biocreative, as entidades e relações foram definidas em nível de documento. Os dados podem ser baixados do repositório Github [1].

4.3.1 Descrição da tarefa

Assim como no primeiro estudo de caso, a tarefa de extração automática de relações consiste em duas subtarefas:

1. Reconhecimento de entidades nomeadas (REN): localização e classificação de entidades: *Chemical* e *gene*, *Disease*, *Species*, *ProteinMutation*, *DNAMutation* e *SNP*.

2. Extração de relações entre entidades: o conjunto de dados é o mesmo que na tarefa de REN, podendo retornar uma lista de pares de entidades de *químico-doença* ou *gene-doença* ou *químico-gene* associados a algum tipo de interação. Nesta tarefa, as relações referenciadas são: *marker/mechanism, therapeutic marker/mechanism , therapeutic , increase_expression , increase_metabolic_proc , decrease_expression , increase_activity , affects_response, decrease_activity, affects_transport, increase_reaction , decrease_reaction , decrease_metabolic_proc.*

Assim, este estudo de caso apresenta as seguintes caraterísticas:

- Dados anotados ruidosos: uso de supervisionamento à distância.
- Classificação de múltiplas relações: existem 14 classes de relações.
- Multi-tarefa: o modelo é treinado para as tarefas de REN e ER.
- Dados desbalanceados: a distribuição de exemplos por relação é altamente desbalanceada.

[1] `https://github.com/patverga/bran`

– Análise para longas sequências: analisa-se o desempenho do modelo para diferentes comprimentos de texto.

4.3.2 Conjunto de dados

O conjunto de dados contém 68.400 resumos de artigos médicos, dentre os quais foram coletados 166.474 exemplos positivos. As relações podem existir entre pares Químico/Doença, Químico/Gene e Gene/Doença. Os detalhes são apresentados na Tabela 4.8.

Tipo	Documentos	Exemplos Positivos	Exemplos Negativos
Químico/Doença	64.139	93.940	571.932
Químico/Gene	34.883	63.463	360.100
Gene/Doença	32.286	9.071	266.461

Tabela 4.8: Distribuição de exemplos positivos/negativos por tipo de relação

Os detalhes da distribuição de dados para treinamento, validação e teste são apresentados na Tabela 4.9.

	Treinamento	Validação	Teste
Químico/Doença			
marker/mechanism	41,562	5,126	5167
therapeutic	24,151	2,929	3,059
Gene/Doença			
marker/mechanism	5,930	825	819
therapeutic	560	77	75
Químico/Gene			
increase_expression	15,581	1,958	2,137
increase_metabolic_proc	5,986	740	638
decrease_expression	5,870	698	783
increase_activity	4,154	467	497
affects_response	3,834	475	508
decrease_activity	3,124	396	434
affects_transport	3,009	333	361
increase_reaction	2,881	367	353
decrease_reaction	2,221	247	269
decrease_metabolic_proc	798	100	120

Tabela 4.9: Distribuição dos dados para treinamento, validação e teste.

Cada resumo tem como limite máximo 500 *tokens*. O conjunto de dados CTD contém mais de 100 tipo de relações organizadas de forma hierárquica. As relações foram mapeadas para seu superior hierárquico, resultando em 14 tipos de relações.

4.3.3 Resultados

Foi utilizado uma função de custo ponderada, descrita na Eq. (4-2).

	Proposed Model			BRAN (2018)		
	Precisão	Recall	F1-score	Precisão	Recall	F1-score
Químico/Doença						
marker/mechanism	89,6	83,7	**86,5± 0,2**	46,2	57,9	51,3
therapeutic	83,7	79,4	**81,5± 0,3**	55,7	67,1	60,8
Gene/Doença						
marker/mechanism	89,1	74,7	**81,3± 0,4**	42,2	44,4	43,0
therapeutic	63,3	26,4	**37,3± 0,3**	52,6	10,1	15,8
Químico/Gene						
increase_expression	57,0	69,7	**62,7± 0,6**	39,7	48,0	43,3
increase_metabolic_proc	50,5	53,8	**52,1± 0,2**	26,3	35,5	29,9
decrease_expression	49,1	40,5	**44,4± 0,9**	34,4	32,9	33,4
increase_activity	50,1	34,8	**41,1± 0,7**	24,5	24,7	24,4
affects_response	74,0	41,8	**53,4± 1,3**	40,9	35,5	37,4
decrease_activity	59,1	42,0	**49,0± 0,6**	30,8	19,4	23,5
affects_transport	50,8	36,6	**42,5± 1,4**	28,7	23,8	25,8
increase_reaction	30,3	5,6	**9,5± 1,8**	12,8	5,6	7,4
decrease_reaction	25,8	8,7	**13,0± 0,9**	12,3	5,7	7,4
decrease_metabolic_proc	40,0	6,7	**11,5± 4,2**	28,9	7,0	11,0

Tabela 4.10: Resultados da avaliação de extração de relações no conjunto de dados CTD (dados de teste).

O modelo proposto prevê com melhor desempenho as relações com maior quantidade de exemplos. Devido à elevada quantidade de exemplos para treinamento, validação e teste, foi utilizada apenas a melhor configuração do primeiro estudo de caso (NTCRE-CPD) para a análise. Os resultados apresentados são compostos da média e do desvio padrão para 10 experimentos. Cada experimento necessita de 12hs a 14hs para treinar 60000 iterações em uma máquina Intel (R) Xeon (R) CPU E5-2660 2.00Ghz com uma placa gráfica V100 de 16GB (arquitetura Volta). Dada a grande quantidade de exemplos no

conjunto de dados CTD, uma abordagem interessante é analisar se o modelo consegue aprender relações de longas dependências entre entidades e não apenas as de curta dependência. Na Figura 4.4 é apresentado um histograma de distância entre entidades que formam uma relação. Pode-se observar que existe uma grande quantidade de relações de curta dependência, com uma média de 90 tokens de separação.

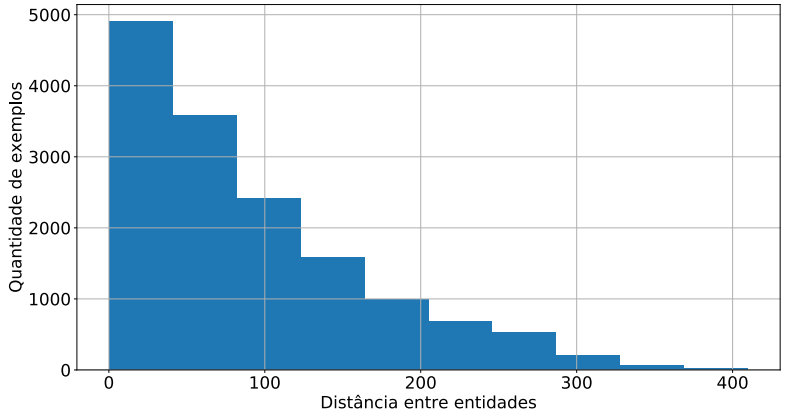


Figura 4.4: Distribuição do conjunto de dados CTD em função da distância entre pares de entidades que formam a relação.

A Figura 4.5, apresenta o desempenho (F1-score) do modelo considerando distâncias máximas de [50,100,200,300,500] entre entidades que formam a relação. Pode-se observar que o modelo consegue manter um bom F1-score para diferentes distâncias e diferentes grupos de classes de relações.

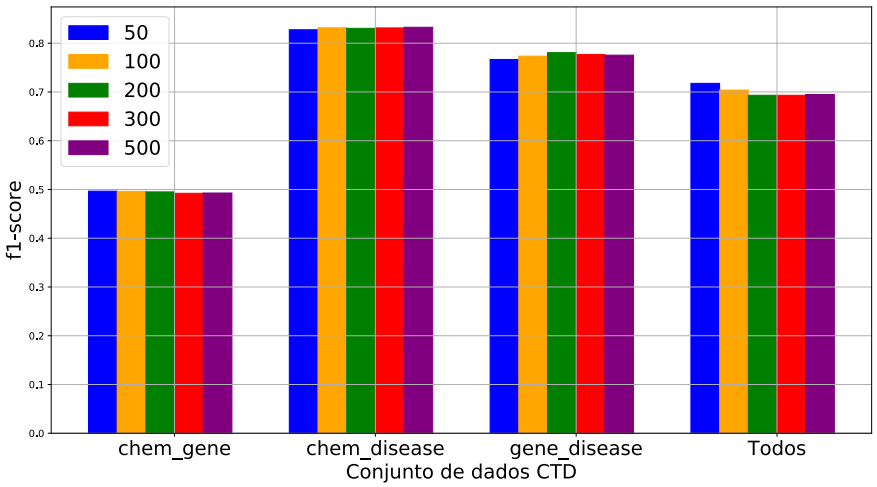


Figura 4.5: Desempenho do modelo no conjunto de dados CTD em função da distância entre pares de entidades que formam a relação.

4.3.4
Discussão de resultados do estudo de caso

Este estudo de caso pode ser considerado como uma extensão do primeiro, dado que também considera dados biomédicos. No entanto, o número de entidades e relações é maior neste estudo. Tem-se observado ao longo dos experimentos uma ampla diferença entre os resultados do modelo BRAN e os do modelo proposto, especialmente nas classes com maior quantidade de exemplos, sendo os grupos Chemical/Disease e Gene/Disease os grupos com melhor F1-score. Na Figura 4.6 são apresentados os históricos da média F1-score de validação para as 14 classes de relações.

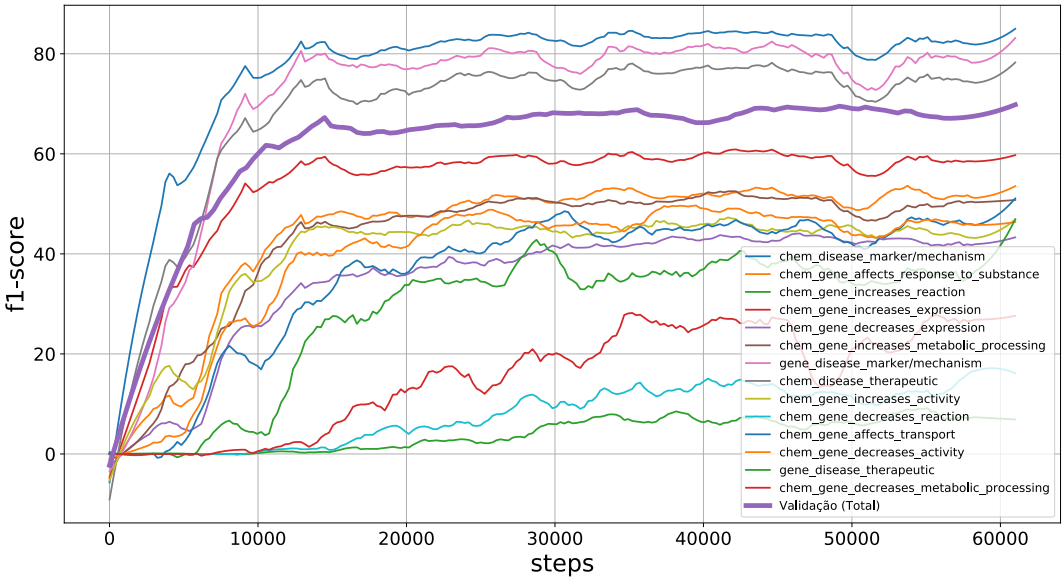


Figura 4.6: Resultados F1-score de validação para as 14 classes e o F1-score total.

Os modelos com baixa quantidade de exemplos apresentam um Recall muito baixo, o que indica a presença de muitos falsos negativos (isto é, uma baixa detecção). A classe decrease_reaction apresenta um recall baixo e uma alta variância, tendo algumas vezes chegado a zero em F1-score. Analisado o comportamento do modelo para identificar relações com diferentes comprimentos entre entidades, concluiu-se que o modelo consegue manter o desempenho do aprendizado em todas as distâncias analisadas. Da mesma forma que no primeiro estudo de caso, a frequência de treinamento foi reduzida para a tarefa REN de forma a dar prioridade ao treinamento da tarefa de ER. Os resultados da tarefa REN são apresentados no Anexo C.1. Nestes dois

estudos de caso, pode-se observar um melhor desempenho do modelo proposto quando comparado ao do modelo BRAN.

4.4 Extração e classificação de relações semânticas em artigos científicos (SemEval 2018)

De forma geral, uma tendência emergente na área de PLN é a sua utilização na área de ciências e humanidades. Vários trabalhos têm procurado melhorar o acesso a este tipo de literatura e responder a informações que requerem maior capacidade quando comparadas aos tradicionais motores de busca: por exemplo, o reconhecimento e classificação de conceitos e a extração de relações que os conectam. Este estudo de caso trata da extração de relações semânticas de um domínio específico, partindo de um corpus de artigos científicos. Para tanto, é necessário encontrar relações que forneçam informações como: "um novo método proposto para uma tarefa", ou "os resultados de diferentes experimentos são comparados entre si". Estas relações permitem detectar trabalhos de pesquisa que lidam com o mesmo problema ou rastrear a evolução dos resultados em uma determinada tarefa.

4.4.1 Descrição da tarefa

A tarefa consiste na identificação e classificação de seis relações semânticas para um domínio específico. As relações são específicas para o domínio da ciência e podem ser encontradas no resumo de artigos científicos. Nesta tarefa, as relações são definidas em nível de sentença. Deste modo, não é necessário processar o resumo completo para realizar o reconhecimento.

A competição SemEval 2018 Task 7 disponibiliza dois conjuntos de dados para a competição de extração de relações: a primeira com anotações de entidades de forma manual e a segunda de forma automática. A abordagem utilizada neste trabalho integra os dois conjuntos de dados de forma a se ter uma maior quantidade de dados para o treinamento do modelo; posteriormente, avaliam-se as tarefas separadamente. Assim, este estudo apresenta as seguintes caraterísticas:

- Dados anotados limpos e ruidosos.
- Classificação de múltiplas relações: 6 classes de relações.
- Não existem classes de entidades, apenas as relações entre elas.
- Única tarefa: o modelo será treinado com uma única tarefa – extração de relações.

– Pré-treinamento: este estudo de caso apresentou melhores resultados quando incluiu um pré-treinamento não supervisionado.

4.4.2 Conjunto de dados

O conjunto de dados está formado por anotações de 6 relações semânticas: *usage, result, model, part_whole, topic* e *comparison*. Esta tarefa considera dois corpus com anotações realizadas manualmente (*Subtask 1.1* - Dados limpos) e de forma automática (*Subtask 1.2* - Dados ruidosos). Cada corpus está formado por 350 resumos para treinamento e 150 resumos para teste. Os resumos contêm cerca de 100 palavras em média. As anotações são descritas em dois arquivos, apresentados na Figura 4.7 e Figura 4.8.

```
<text id="N03-2017"> <title>Word Alignment with Cohesion Constraint
</title> <abstract> We present a <entity id="N03-2017.1">syntax-based
constraint</entity> for <entity id="N03-2017.2">word alignment</entity> ,
known as the <entity id="N03-2017.3">cohesion constraint</entity> . It
requires disjoint <entity id="N03-2017.4">English phrases</entity> to be
mapped to non-overlapping intervals in the <entity id="N03-2017.5">French
sentence</entity> . We evaluate the utility of this <entity id=
"N03-2017.6">constraint</entity> in two different algorithms. The results
show that it can provide a significant improvement in <entity id=
"N03-2017.7">alignment quality</entity> . </abstract>
```

Figura 4.7: Exemplo de anotação do corpus (*Task7, Subtask 1.1*).

O primeiro arquivo, na Figura 4.7, apresenta um resumo e a localização das suas entidades. Cada resumo e entidade é rotulado por um *id*. O conteúdo dos rótulos `<title>` e `<abstract>` são concatenados formando o resumo. O segundo arquivo apresenta as relações entre as entidades rotuladas. O formato é apresentado na Figura 4.8.

```
USAGE(N03-2036.1,N03-2036.2)
PART_WHOLE(N03-2036.12,N03-2036.13)
USAGE(N03-2036.15,N03-2036.16,REVERSE)
USAGE(N03-3010.1,N03-3010.2)
PART_WHOLE(N03-4010.1,N03-4010.2,REVERSE)
```

Figura 4.8: Lista de relações entre entidades (*Task7, Subtask 1.1*).

Cada linha indica uma relação do tipo: TIPO DE RELAÇÃO (ENTIDADE_ID_ 1 , ENTIDADE_ID_2). Após a leitura das anotações, foi realizada uma análise sobre a distribuição das classes no conjunto de dados.

4.4.3 Balanceamento de classes

A distribuição de classes para as 6 relações é apresentada na Tabela 4.11. A coluna "Reversa" indica a direção da relação (esquerda → direita = Sim).

Relação	Subtarefa		Reversa		Total
	1.1	1.2	Sim	Não	
USAGE	483	464	615	332	947
MODEL_FEATURE	326	172	346	152	498
RESULT	72	121	135	58	193
TOPIC	18	240	235	23	258
PART_WHOLE	233	192	273	152	425
COMPARE	95	41	136	-	136

Tabela 4.11: Número de exemplos para cada relação.

Algumas relações podem ser simétricas ou assimétricas. As relações USAGE, MODEL_FEATURE, RESULT, TOPIC, PAT_WHOLE, NULL são assimétricas e COMPARE é a única relação simétrica. Esta informação é relevante dado que, para o caso de COMPARE, pode-se acrescentar anotações de relação invertendo a direção da relação (a→b, b→a). Pode-se observar na coluna "Total" que a distribuição dos dados por relação é completamente desbalanceada. Por isso, foi utilizada a função de custo ponderada para cada classe descrita na Eq.4-2.

4.4.4 Pré-treinamento

De forma a aproveitar as características aprendidas de forma não supervisionada, dois conjuntos de dados de domínio similar foram utilizados para o treinamento dos vetores que inicializam os *embeddings* de palavras:

1. ACL Anthology Reference Corpus [1]: coleção de publicações de linguística computacional com um total de 22,878 resumos de artigos.

2. SemEval 2018 Task7 (Task 1.1 e Task 1.2)[2]: conjunto de 1000 resumos de artigos científicos.

Os corpus foram pré-processados e treinados conjuntamente, utilizando-se a biblioteca gensim[1], com o algoritmo word2vec e vetores de dimensão 300. De

[1] https://acl-arc.comp.nus.edu.sg/
[2] https://lipn.univ-paris13.fr/~gabor/semeval2018task7/
[1] https://radimrehurek.com/gensim/

forma a comparar resultados, também foram utilizados vetores pré-treinados de repositórios públicos:

1. Google News word2vec [3]: Corpus de aprox. 100 bilhões de palavras, vocabulário de 3 milhões de palavras, vetor de dimensão 300.

2. NLPL repository [4] [99]: Repositório de vetores de palavras pré-treinados. Os vetores escolhidos foram treinados com textos da Wikipédia com o método word2vec com vetores de dimensão 300.

4.4.5 Resultados

O modelo proposto foi comparado com os melhores modelos da competição de SemEval2018 para as subtarefas 1.1. e 1.2. O modelo ETH-DS3Lab [100] utilizou um híbrido entre um modelo CNN e uma LSTM e o modelo UWNLP [101] foi baseado em um modelo LSTM. Os modelos SIRIUS-LTG-UiO [102] e Talla[103] utilizaram modelos baseados em CNN. Os resultados para ambas astarefas são apresentados na Tabela 4.12. Para cada modelo indica-se, entre parênteses, as posições da competição SemEval2018 nas subtarefas 1.1 e 1.2 respectivamente:

Modelo	Subtarefa	
	1.1	1.2
ETH-DS3Lab (1ero/1ero)	81,7	90,4
UWNLP (2do lugar/-)	78,9	-
SIRIUS-LTG-UiO (3ro lugar/3ro lugar)	76,7	83,2
Talla (4to/2do lugar)	76,7	83,2
UTRE-CPD	**72,5 ± 0,7**	**80,0**
NTCRE-CPD	**75,7 ± 0,3**	**83,6**
NTCRE-CPD (ensemble)	**77,3**	**85,4**

Tabela 4.12: Tabela de Resultados F1-score para a tarefa 1.1 e 1.2. de SemEval 2018.

Neste estudo de caso foram utilizadas as melhores configurações das abordagens propostas UTRE-CPD e NTCRE-CPD. Pode-se observar que ambas configurações conseguem obter bons resultados para as subtarefas 1.1. e 1.2. , sendo que a NTCRE-CPD apresenta maior acurácia em ambas as tarefas. A versão assemble do modelo NTCRE-CPD tem um ganho na acurácia final.

[3] `https://code.google.com/archive/p/word2vec/`
[4] `http://vectors.nlpl.eu/repository/`

A versão assemble foi baseado na média de 10 experimentos e a técnica de voto majoritário[1]. Um consenso entre todas as técnicas da tabela 4.12 (incluindo o modelo proposto nesta tese) é a utilização de *embeddings* pré-treinados a partir do corpus ACL. Caso este processo não seja realizado, os resultados serão muito inferiores. Os melhores modelos da competição realizaram outros processos linguísticos que aumentaram suas bases de conhecimento. Por exemplo, o modelo UWNLP utilizou um modelo estado da arte em rotulação de eventos científicos sob o conjunto de dados ACL com a finalidade de criar *embeddings* de conceitos pré-treinados. Para o modelo ETH-DS3Lab, as entidades dos dados de teste foram observadas e combinadas aos dados de treinamento e a qualidade destas novas anotações foi avaliada. Neste modelo também foram propostas técnicas para inverter a direção da relação, aumentando a quantidade de anotações. Lamentavelmente, os códigos dos modelos não são públicos, nem os detalhes de implementação das transformações dos dados. Nestes casos, torna-se inviável efetuar uma avaliação objetiva dos resultados, visto que os modelos estão sendo treinados com diferentes conjuntos de dados.

O modelo UTRE-CPD apresenta o segundo melhor resultado e um fator importante entre as duas abordagens é o tempo de processamento, que será discutido na parte final deste capítulo. Estes modelos foram analisados com quatro inicializações de pesos. Os resultados são apresentados na Tabela 4.13. O símbolo — indica uma inicialização de *embeddings* aleatórios.

Configuração	F1-score(subtarefa 1.1/1.2)			
	—	GoogleNews	NLPL	ACL+SemEval2018
UTRE-CPD	61.4/64.8	71.5/78.9	68.6/76.1	72.5/80.0
NTCRE-CPD	63./65.3	73.6/79.1	69.9/73.3	75.7/83.6

Tabela 4.13: Resultados do modelo proposto utilizando diferentes inicializações de embeddings.

4.4.6 Discussão de resultados do estudo de caso

Diferentemente dos dois primeiros estudos de caso, o reduzido conjunto de dados obrigou a utilização de vetores pré-treinados para inicializar os pesos. Isto torna os *embeddings* essenciais para a acurácia final do modelo. Por essa razão, os vetores pré-treinados foram obtidos de diversas fontes, sendo que o melhor desempenho foi gerado pelo treinamento do conjunto de artigos do repositório 'ACL+SemEval2018 '.

[1]Técnica *hard voting*: https://towardsdatascience.com/ensemble-learning-in-machine-learning-getting-started-4ed85eb38e00

Este estudo de caso permitiu testar a capacidade do modelo para detecção de relações a curtas distâncias, já que as frases são definidas em nível de frase. Grande parte dos modelos na literatura não utilizou os resumos em frases para facilitar o treinamento. No entanto, o modelo proposto utilizou resumos completos. A tarefa foi mais complexa para o modelo UTRE-CPD, dado que o mecanismo *Attention* obriga um relacionamento com todos os elementos da sequência. Mesmo assim, os resultados foram bons. O modelo NTCRE-CPD introduz um nível de esparsidade, dado que o MCAM decide que informação deve ser passada para cada elemento da sequência. Neste caso, o modelo NTCRE-CPD também consegue o melhor resultado com uma baixa variância.

4.5 Analise de Desempenho

Nesta seção analisa-se o desempenho dos mecanismos e configurações de modelos apresentados.

4.5.1 Análise de desempenho do mecanismo de alocação dinâmica paralela

O mecanismo de alocação dinâmica descrito em 3.3 é projetado para o processamento sequencial. A sua implementação requer operações de ordem e indexação, sendo a operação de ordem a de maior custo computacional. Para um tamanho grande de memória, diversos algoritmos de ordem (*quick sort*, *bubble sort*, etc.) são uma alternativa ótima para reduzir a complexidade computacional da operação. Também pode-se considerar alternativas paralelas como os sistemas distribuídos. O MCAM do modelo proposto NTCRE apresenta as seguintes características:

1. Projetado para longas sequências (100-1000 elementos).
2. O módulo *transformer*, utilizado para criar a sequência de vetores de interface, caracteriza-se por um número reduzido de iterações.
3. Cada elemento da sequência dispõe de um MCAM contendo um módulo de memória externa.

Sequências longas indicam múltiplas operações de ordem simultânea a cada iteração do módulo transformer. Dado que o número de iterações é reduzido, são poucas as escritas realizadas na memória através das quais se pode considerar um tamanho pequeno de memória. A memória total é muito maior e depende do tamanho da sequência. Outra característica importante é que a maioria dos modelos de aprendizado profundo são executados em unidades de

processamento gráfico (GPUs), as quais apresentam uma arquitetura *Single Instruction Multiple Thread* em cada *Stream Multiprocessor*, o que as impede de executar diferentes sequências de ordem em cada *thread* de execução. Assim, a GPU perde a vantagem do paralelismo. Contudo, a alocação dinâmica paralela apresenta uma complexidade quadrática em respeito ao tamanho da memória atribuída a cada elemento e linear em respeito ao número de elementos. Em contrapartida, as operações são altamente paralelizáveis. Visto que para o modelo proposto o tamanho de memória é pequeno (N=4), o mecanismo consegue obter um bom desempenho e torna viável a utilização do MCAM para longas sequências. De forma a comparar computacionalmente os mecanismos, foram analisados os dois parâmetros que definem a complexidade: sequências de $[1, 10, 50, 100, 200, 500, 1000]$ elementos e memórias de $[4, 16, 32]$ palavras. A Tabela 4.14 apresenta os resultados de 100 repetições de cada experimento. Os experimentos foram rodados em uma máquina do google colab [1] com GPU Tesla T4.

Tamanho de	Tamanho da sequência (# tokens)						
Memória	1	10	50	100	200	500	1000
Alocação Dinâmica (seg)							
4	2,47	3,16	4,69	6,32	10,08	22,16	41,39
16	2,39	3,50	5,54	7,55	12,31	26,28	49,70
32	2,33	3,71	6,71	8,97	14,90	32,07	60,93
Alocação Dinâmica Paralela (seg)							
4	0,67	0,64	0,63	0,70	0,75	0,74	0,87
16	0,69	0,65	0,74	0,77	0,90	1,34	2,05
32	0,68	0,65	0,89	1,00	1,45	2,93	5,03
Speedup							
4	3.68	4.89	7.42	8.97	13.51	30.05	**47.83**
16	3.49	5.39	7.53	9.89	13.86	19.61	**24.16**
32	3.45	5.69	7.58	8.52	10.28	10.94	**12.12**

Tabela 4.14: Comparação de desempenho do mecanismo de alocação dinâmica e dinâmica paralela.

Speedup é a relação representada pela divisão entre tempos de processamento do mecanismo de alocação dinâmica e de dinâmica paralela. Pode-se observar que a aceleração (*SpeedUp*) aumenta para longas sequências devido ao grau de paralelismo do mecanismo. Para longas sequências, também se pode

[1] site de Google colab: `https://colab.research.google.com`

observar que o modelo é altamente escalável. Isto é mais perceptível graficamente (Figura 4.9):

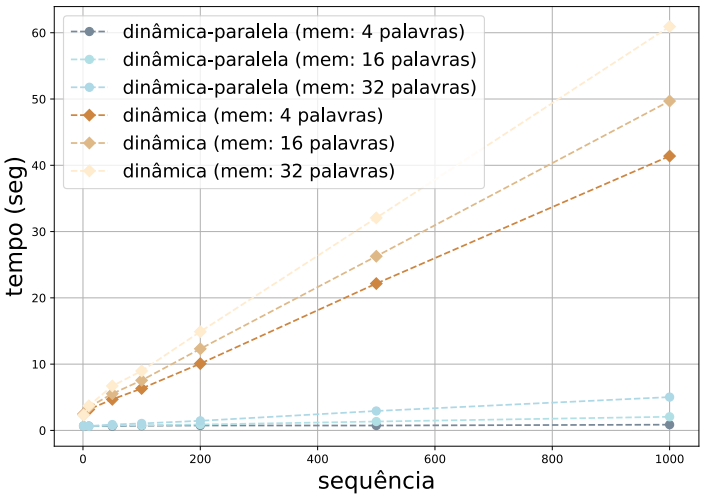


Figura 4.9: Tempo de processamento: mecanismos de alocação dinâmica e dinâmica paralela. O mecanismo original (linhas com marcador ⋄) aumenta o tempo de processamento com uma inclinação muito mais elevada do que o faz o mecanismo proposto (linhas com marcador ∘).

Pode-se observar que, com o aumento da sequência, o custo computacional em ambos os mecanismos aumenta linearmente, enquanto o mecanismo proposto o incrementa apenas um pouco. Um caso crítico é o de um tamanho de memória igual a 4 (valor usado nos três estudos de caso), no qual praticamente não existe variação. O código deste mecanismo é detalhado no Anexo A.1.

4.5.2 Análise de desempenho dos mecanismos propostos.

No Capítulo 3 foram apresentados dois mecanismos que aprimoram o desempenho do modelo proposto: o módulo de controle de acesso a memória (MCAM) e o módulo condição de parada dinâmica (CPD). A integração de cada mecanismo acrescenta custo computacional ao sistema. Nesta seção, são analisados os tempos de processamento de cada modelo para os conjuntos de dados SemEval 2018 e BioCreative V. Os dois se diferenciam pelo comprimento do texto, sendo Biocreative V o conjunto de textos de maior comprimento (Figura 4.10). O conjunto SemEval2018 tem uma média de 96 tokens por resumo e a base de dados Biocreative, 398 tokens. A rede neural processa a sequência completa de tokens em paralelo.

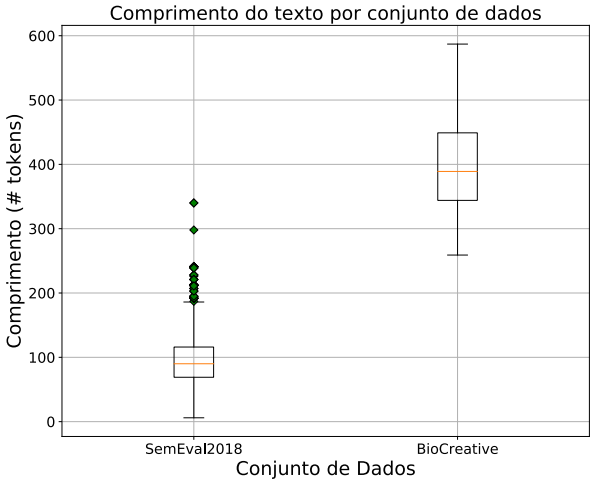


Figura 4.10: Comprimento de Texto dos conjuntos de dados SemEval2018 e BiocreativeV.

Durante o treinamento dos estudos de caso prévios, o tempo de processamento de cada experimento foi medido. Na Tabela 4.15 é apresentada, para cada configuração, a média do tempo despendido para a rede neural realizar uma iteração. Pode-se observar a escalabilidade do modelo, já que, ao se aumentar o tamanho da sequência por um fator próximo de 4 ($\approx$ 398/96), o tempo de processamento também varia na mesma escala. Para fins de comparação do tempo de processamento, foi considerada a configuração NTCRE com e sem CPD.

Modeo	SemEval2018		BioCreative	
	Média	σ	Média	σ
UTRE	0.109	0.011	0.488	0.015
NTCRE	0.141	0.014	0.623	0.015
UTRE-CPD	0.146	0.014	0.570	0.013
NTCRE-CPD	0.232	0.025	0.802	0.055

Tabela 4.15: Tabela de tempo de processamento por iteração (seg/iteração) para cada configuração.

O tempo por iteração na base BioCreative apresenta um desvio padrão maior que o da a base SemEval, algo que também acontece no comprimento dos textos para cada conjunto. Na Figura 4.11, observa-se uma distribuição de tempo similar para os dois conjuntos de dados. Em ambos os casos, o modelo UTRE é mais simples e rápido. O MCAM (NTCRE) e o módulo CPD (UTRE-CPD) acrescentam tempos de processamento semelhantes. No entanto, os dois módulos em conjunto (NTCRE-CPD) aumentam o custo computacional ainda

mais. Isto pode ser devido ao incremento da complexidade computacional para o cálculo do gradiente.

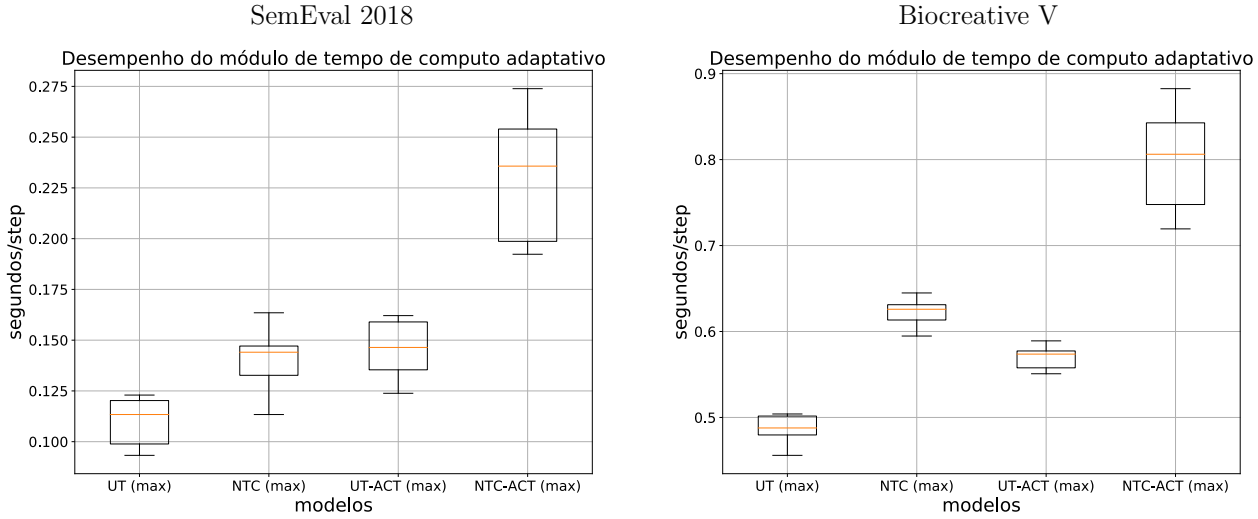


Figura 4.11: Tempo de processamento para as configurações das abordagens UTRE e NTCRE.

Apesar de o modelo NTCRE-CPD ter apresentado os resultados mais relevantes nos estudos de caso, o seu treinamento é $\approx 50\%$ mais custoso computacionalmente do que o do UTRE-CPD. Isto torna os dois modelos atraentes para trabalhar em conjunto ou realizar pré-treinamentos de *embeddings*. Um modelo nesta linha é o modelo BERT [104] que, de modo geral, é um mecanismo *transformer* que realiza um pré-treinamento, utilizando a técnica *Masked LM* para realizar a tarefa de modelamento de linguagem sobre os dados de treinamento. Estes conjuntos de técnicas abrem um universo de novas estratégias para o aprendizado automático, no qual o limite é decidido somente pelo poder computacional.

5
Conclusões e Trabalhos Futuros

Este trabalho apresentou a proposta de um modelo híbrido que sintetiza as características dos mecanismos baseados em *Transformers*, módulo de condição de parada dinâmica e mecanismos de acesso à memória. Este modelo é capaz de aprender a extração de conhecimento de forma mais eficiente, ao obter resultados superiores, ou equivalentes, aos modelos do estado da arte em competições BioCreativeV, CTD e SemEval2018 (modelos que não utilizaram bases de conhecimento adicionais), com um número menor de iterações. Ele também é capaz de oferecer escalabilidade, com o aumento da complexidade do problema, devido ao alto nível de paralelismo presente em sua arquitetura.

O modelo NTCRE aqui proposto diferencia-se dos demais modelos *Transformer* principalmente por utilizar recursos de controle de acesso à memória externa, integrando os dois mecanismos para sintetizar os benefícios de ambas características em um novo modelo. O modelo *Transformer* pode aprimorar o nível de raciocínio, aprendendo a manipular o controle de acesso à memória. O MCAM pode aprender muito mais rapidamente devido a sua habilidade de compartilhar neurônios do controlador com todos os elementos da sequência, aprendendo com um número maior de exemplos a cada iteração.

De forma concisa, pode-se mencionar os benefícios de cada estrutura interna da rede neural proposta:

- As camadas *BatchNormalization* na entrada da rede, na função de transição e na projeção de tarefas aceleraram a convergência quando comparadas ao modelo base BRAN.
- O módulo de condição de parada dinâmica permitiu aprimorar o desempenho dos modelos e reduzir a variância do modelo.
- A integração entre o mecanismo *Transformer* e o MCAM foi realizada na saída do módulo *Multi-Head Attention*, permitindo à rede neural um raciocínio mais sofisticado sobre o vetor de *Attention* antes de este ser utilizado.
- De forma a tornar possível esta integração, foi proposto o módulo de alocação dinâmica paralela, que permite reduzir drasticamente o tempo de cálculo de endereços de memória dinâmica, atingindo execuções até 40 vezes mais rápidas.

– O MCAM permite ao módulo *Attention* ter um grau de esparsidade, dado que o controlador decide que informação deve ser lida para cada elemento da sequência.

A primeira abordagem (UTRE-CPD) contém parte do sistema proposto, sendo o objetivo final a segunda abordagem (NTCRE-CPD). Este modelo muito mais robusto consegue o melhor desempenho entre todas as configurações e ao mesmo tempo eleva o custo computacional em até 100% se comparado ao modelo UTRE, ou 50% se comparado ao modelo UTRE-CPD. O número de iterações necessárias para a convergência é muito menor em comparação ao modelo base BRAN.

Os trabalhos futuros consistem em dar continuidade ao processo de pesquisa, aprimoramento, desenvolvimento e avaliação experimental do modelo NTCRE+CPD, conforme descrito nas próximas seções.

5.1 Multilinguagem

A utilização de *embeddings* de palavras permite aplicar abordagens capazes de transferir conhecimento aprendido entre diferentes linguagens. Isto ocasiona um aumento na robustez dos modelos. Outro ponto importante é a grande disponibilidade de dados no idioma inglês, o que pode ser aproveitado para aplicações em outras linguagens como francês, português, espanhol, etc.

5.2 Hiperparâmetros

Como consequência da sintetização de diferentes técnicas em um único modelo mais robusto, o número de hiperparâmetros aumenta, assim como a dificuldade de sintonizar o modelo. É necessário definir procedimentos que levem a uma configuração adequada para o treinamento, como, por exemplo, os tipos de regularização: de pesos, de saturação, de condição de parada dinâmica e *dropout* (o modelo apresenta camadas *dropout* na parte intermediária e na saída da rede). Também é necessário sintetizar estes parâmetros de forma a facilitar a escolha daqueles mais adequados para cada treinamento.

5.3 Inicialização de pesos

No modelo atual de NTCRE+CPD, a inicialização de pesos, principalmente os *embeddings* de palavras, é uma tarefa crítica para se obter um bom aprendizado. Estes parâmetros podem ser inicializados com tarefas não supervisionadas (word2vec, GloVe, etc). Uma abordagem similar é utilizar a técnica

Mask LM, que permite a inicialização do modelo completo e a aprendizagem das características principais do corpus de texto, realizando a tarefa de modelamento de linguagem. O modelo proposto neste trabalho pertence ao grupo de modelos baseados em *Transformers* capazes de realizar este tipo de treinamento, o que os tornaria muito mais eficientes na solução de problemas.

5.4 Aceleração utilizando sistemas distribuídos

Para grandes conjuntos de dados escalares, o sistema é um fator importante. É necessário utilizar múltiplas GPUs acompanhados de técnicas de *warmup* para aumentar a taxa de aprendizado em conjunto com o tamanho do batch, sendo este paralelizado por meio de múltiplas GPUs.

5.5 Aumentar a esparsidade do módulo *Attention*

Um exemplo prático foi o terceiro estudo de caso, no qual as relações entre entidades encontravam-se apenas em nível de frase. O modelo processa o texto completo e a operação de *Attention* relaciona todos os elementos da sequência, o que pode dificultar o treinamento em relação a treinar apenas a frase em que se encontram as entidades relacionadas. O MCAM dá um passo nessa direção, controlando o fluxo de informações que passa para cada elemento da sequência. No entanto, esta característica pode ser aprimorada com novas abordagens de esparsidade.

Referências bibliográficas

[1] FANG, H.. **Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem**. In: CYBER TECHNOLOGY IN AUTOMATION, CONTROL, AND INTELLIGENT SYSTEMS (CYBER), 2015 IEEE INTERNATIONAL CONFERENCE ON, p. 820–824. IEEE, 2015.

[2] TAYLOR, C.. **Structured vs. Unstructured Data**. `https://www.datamation.com/big-data/structured-vs-unstructured-data.html`, 2018. [Online; accessed 13-May-2018].

[3] GRIMES, S.. **Unstructured data and the 80 percent rule**. Carabridge Bridgepoints, p. 10, 2008.

[4] ZHAI, C.; MASSUNG, S.. **Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining**. Morgan & Claypool, 2016.

[5] LUONG, M.-T.; SUTSKEVER, I.; LE, Q. V.; VINYALS, O. ; ZAREMBA, W.. **Addressing the rare word problem in neural machine translation**. arXiv preprint arXiv:1410.8206, 2014.

[6] SOCHER, R.; PENNINGTON, J.; HUANG, E. H.; NG, A. Y. ; MANNING, C. D.. **Semi-supervised recursive autoencoders for predicting sentiment distributions**. In: PROCEEDINGS OF THE CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, p. 151–161. Association for Computational Linguistics, 2011.

[7] WEN, T.-H.; GASIC, M.; MRKSIC, N.; SU, P.-H.; VANDYKE, D. ; YOUNG, S.. **Semantically conditioned lstm-based natural language generation for spoken dialogue systems**. arXiv preprint arXiv:1508.01745, 2015.

[8] MAYNARD, D.; BONTCHEVA, K. ; AUGENSTEIN, I.. **Natural language processing for the semantic web**. Synthesis Lectures on the Semantic Web: Theory and Technology, 6(2):1–194, 2016.

[9] RODRIGUES, M.; TEIXEIRA, A. ; OTHERS. **Advanced applications of natural language processing for performing information extraction**. Springer, 2015.

[10] MINTZ, M.; BILLS, S.; SNOW, R. ; JURAFSKY, D.. **Distant supervision for relation extraction without labeled data**. In: PROCEEDINGS OF THE JOINT CONFERENCE OF THE 47TH ANNUAL MEETING OF THE ACL AND THE 4TH INTERNATIONAL JOINT CONFERENCE ON NATURAL LANGUAGE PROCESSING OF THE AFNLP: VOLUME 2-VOLUME 2, p. 1003–1011. Association for Computational Linguistics, 2009.

[11] VERGA, P.; BELANGER, D.; STRUBELL, E.; ROTH, B. ; MCCALLUM, A.. **Multilingual relation extraction using compositional universal schema**. arXiv preprint arXiv:1511.06396, 2015.

[12] RIEDEL, S.; YAO, L.; MCCALLUM, A. ; MARLIN, B. M.. **Relation extraction with matrix factorization and universal schemas**. In: PROCEEDINGS OF THE 2013 CONFERENCE OF THE NORTH AMERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS: HUMAN LANGUAGE TECHNOLOGIES, p. 74–84, 2013.

[13] LECUN, Y.; BENGIO, Y. ; HINTON, G.. **Deep learning**. nature, 521(7553):436, 2015.

[14] BAHDANAU, D.; CHO, K. ; BENGIO, Y.. **Neural machine translation by jointly learning to align and translate**. CoRR, abs/1409.0473, 2014.

[15] VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł. ; POLOSUKHIN, I.. **Attention is all you need**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, p. 5998–6008, 2017.

[16] GRAVES, A.. **Deep learning 7. attention and memory in deep learning (deepmind)**.

[17] BANZHAF, W.; NORDIN, P.; KELLER, R. E. ; FRANCONE, F. D.. **Genetic programming: an introduction**, volumen 1. Morgan Kaufmann San Francisco, 1998.

[18] BRAMEIER, M. F.; BANZHAF, W.. **Linear genetic programming**. Springer Science & Business Media, 2007.

[19] GRAVES, A.; WAYNE, G. ; DANIHELKA, I.. **Neural turing machines**. arXiv preprint arXiv:1410.5401, 2014.

[20] REED, S.; DE FREITAS, N.. **Neural programmer-interpreters**. arXiv preprint arXiv:1511.06279, 2015.

[21] KAISER, Ł.; SUTSKEVER, I.. **Neural gpus learn algorithms**. arXiv preprint arXiv:1511.08228, 2015.

[22] GRAVES, A.; WAYNE, G.; REYNOLDS, M.; HARLEY, T.; DANIHELKA, I.; GRABSKA-BARWIŃSKA, A.; COLMENAREJO, S. G.; GREFENSTETTE, E.; RAMALHO, T.; AGAPIOU, J. ; OTHERS. **Hybrid computing using a neural network with dynamic external memory**. Nature, 538(7626):471, 2016.

[23] RILOFF, E.; LORENZEN, J.. **Extraction-based text categorization: Generating domain-specific role relationships automatically**. In: NATURAL LANGUAGE INFORMATION RETRIEVAL, p. 167–196. Springer, 1999.

[24] COWIE, J.; WILKS, Y.. **Information extraction**. Handbook of Natural Language Processing, 56:57, 2000.

[25] MOENS, M.-F.. **Information extraction: algorithms and prospects in a retrieval context**, volumen 21. Springer Science & Business Media, 2006.

[26] DAS, S.; CAKMAK, U. M.. **Hands-On Automated Machine Learning: A beginner's guide to building automated machine learning systems using AutoML and Python**. Packt Publishing Ltd, 2018.

[27] MANNING, C.; RAGHAVAN, P. ; SCHÜTZE, H.. **Introduction to information retrieval**. Natural Language Engineering, 16(1):100–103, 2010.

[28] SCHÜTZE, H.; MANNING, C. D. ; RAGHAVAN, P.. **Introduction to information retrieval**. In: PROCEEDINGS OF THE INTERNATIONAL COMMUNICATION OF ASSOCIATION FOR COMPUTING MACHINERY CONFERENCE, p. 260, 2008.

[29] ZITOUNI, I.. **Natural language processing of semitic languages**. Springer, 2014.

[30] RATINOV, L.; ROTH, D.. **Design challenges and misconceptions in named entity recognition**. In: PROCEEDINGS OF THE THIRTEENTH CONFERENCE ON COMPUTATIONAL NATURAL LANGUAGE LEARNING, p. 147–155. Association for Computational Linguistics, 2009.

[31] RAMSHAW, L. A.; MARCUS, M. P.. **Text chunking using transformation-based learning**. In: NATURAL LANGUAGE PROCESSING USING VERY LARGE CORPORA, p. 157–176. Springer, 1999.

[32] KHALID, M. A.; JIJKOUN, V. ; DE RIJKE, M.. **The impact of named entity normalization on information retrieval for question answering**. In: EUROPEAN CONFERENCE ON INFORMATION RETRIEVAL, p. 705–710. Springer, 2008.

[33] TORAL, A.; NOGUERA, E.; LLOPIS, F. ; MUNOZ, R.. **Improving question answering using named entity recognition**. In: INTERNATIONAL CONFERENCE ON APPLICATION OF NATURAL LANGUAGE TO INFORMATION SYSTEMS, p. 181–191. Springer, 2005.

[34] BABYCH, B.; HARTLEY, A.. **Improving machine translation quality with automatic named entity recognition**. In: PROCEEDINGS OF THE 7TH INTERNATIONAL EAMT WORKSHOP ON MT AND OTHER LANGUAGE TECHNOLOGY TOOLS, IMPROVING MT THROUGH OTHER LANGUAGE TECHNOLOGY TOOLS: RESOURCES AND TOOLS FOR BUILDING MT, p. 1–8. Association for Computational Linguistics, 2003.

[35] RUSSELL, S. J.; NORVIG, P.. **Artificial intelligence: a modern approach. malaysia**. Pearson Education Limited. Rycroft-Malone, J.(2004). The PARIHS framework—A framework for guiding the implementation of evidenceǴ based practice. Journal of nursing care quality, 19(4):297–304, 2016.

[36] CHOLLET, F.. **Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek**. MITP-Verlags GmbH & Co. KG, 2018.

[37] BANGALE, T.. **What is artificial intelligence?** Online; accessed: 13-May-2018.

[38] FEIGENBAUM, E.. **Expert systems in the 1980s, state of the art report on machine intelligence**. Per-gamon-Infotcch, 1981.

[39] GOODFELLOW, I.; BENGIO, Y. ; COURVILLE, A.. **Deep learning**. MIT press, 2016.

[40] CORTES, C.; VAPNIK, V.. **Support-vector networks**. Machine learning, 20(3):273–297, 1995.

[41] HO, T. K.. **Random decision forests**. In: PROCEEDINGS OF 3RD INTERNATIONAL CONFERENCE ON DOCUMENT ANALYSIS AND RECOGNITION, volumen 1, p. 278–282. IEEE, 1995.

[42] KŮRKOVÁ, V.. **Kolmogorov's theorem and multilayer neural networks**. Neural Networks, 5(3):501 – 506, 1992.

[43] BENGIO, Y.; COURVILLE, A. ; VINCENT, P.. **Representation learning: A review and new perspectives**. IEEE transactions on pattern analysis and machine intelligence, 35(8):1798–1828, 2013.

[44] WANG, W.; ZHENG, V. W.; YU, H. ; MIAO, C.. **A survey of zero-shot learning: Settings, methods, and applications**. ACM Transactions on Intelligent Systems and Technology (TIST), 10(2):13, 2019.

[45] BART, E.; ULLMAN, S.. **Cross-generalization: Learning novel classes from a single example by feature replacement**. In: 2005 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'05), volumen 1, p. 672–679. IEEE, 2005.

[46] CARUANA, R.. **Multitask learning. autonomous agents and multi-agent systems**. 1998.

[47] HINTON, G. E.; SALAKHUTDINOV, R. R.. **Reducing the dimensionality of data with neural networks**. science, 313(5786):504–507, 2006.

[48] CHOI, E.; BAHADORI, M. T.; SEARLES, E.; COFFEY, C.; THOMPSON, M.; BOST, J.; TEJEDOR-SOJO, J. ; SUN, J.. **Multi-layer representation learning for medical concepts**. In: PROCEEDINGS OF THE 22ND ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 1495–1504. ACM, 2016.

[49] CHEN, H.; PEROZZI, B.; HU, Y. ; SKIENA, S.. **Harp: Hierarchical representation learning for networks**. In: THIRTY-SECOND AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2018.

[50] SIMPSON, D.. **Phrenology and the neurosciences: contributions of fj gall and jg spurzheim**. ANZ journal of surgery, 75(6):475–482, 2005.

[51] FLOURENS, P.. **Recherches expérimentales sur les propriétés et les fonctions du système nerveux dans les animaux vertébrés**. Ballière, 1842.

[52] MIKOLOV, T.; CHEN, K.; CORRADO, G. S. ; DEAN, J.. **Efficient estimation of word representations in vector space**. CoRR, abs/1301.3781, 2013.

[53] PENNINGTON, J.; SOCHER, R. ; MANNING, C.. **Glove: Global vectors for word representation**. In: PROCEEDINGS OF THE 2014 CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING (EMNLP), p. 1532–1543, 2014.

[54] PETERS, M. E.; NEUMANN, M.; IYYER, M.; GARDNER, M.; CLARK, C.; LEE, K. ; ZETTLEMOYER, L.. **Deep contextualized word representations**. arXiv preprint arXiv:1802.05365, 2018.

[55] TANG, D.; WEI, F.; YANG, N.; ZHOU, M.; LIU, T. ; QIN, B.. **Learning sentiment-specific word embedding for twitter sentiment classification**. In: PROCEEDINGS OF THE 52ND ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (VOLUME 1: LONG PAPERS), volumen 1, p. 1555–1565, 2014.

[56] LIN, Y.; SHEN, S.; LIU, Z.; LUAN, H. ; SUN, M.. **Neural relation extraction with selective attention over instances**. In: PROCEEDINGS OF THE 54TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (VOLUME 1: LONG PAPERS), volumen 1, p. 2124–2133, 2016.

[57] KUSNER, M.; SUN, Y.; KOLKIN, N. ; WEINBERGER, K.. **From word embeddings to document distances**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 957–966, 2015.

[58] JOLLIFFE, I.. **Principal component analysis**. Technometrics, 45(3):276, 2003.

[59] SCHÖLKOPF, B.; SMOLA, A. ; MÜLLER, K.-R.. **Kernel principal component analysis**. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETWORKS, p. 583–588. Springer, 1997.

[60] HYVÄRINEN, A.; KARHUNEN, J. ; OJA, E.. **Independent component analysis**, volumen 46. John Wiley & Sons, 2004.

[61] MAATEN, L. V. D.; HINTON, G.. **Visualizing data using t-sne**. Journal of machine learning research, 9(Nov):2579–2605, 2008.

[62] HARTMANN, N.; FONSECA, E.; SHULBY, C.; TREVISO, M.; RODRIGUES, J. ; ALUISIO, S.. **Portuguese word embeddings: evaluating on word analogies and natural language tasks**. arXiv preprint arXiv:1708.06025, 2017.

[63] GARG, N.; SCHIEBINGER, L.; JURAFSKY, D. ; ZOU, J.. **Word embeddings quantify 100 years of gender and ethnic stereotypes**. Proceedings of the National Academy of Sciences, 115(16):E3635–E3644, 2018.

[64] KOZLOWSKI, A. C.; TADDY, M. ; EVANS, J. A.. **The geometry of culture: Analyzing meaning through word embeddings**. arXiv preprint arXiv:1803.09288, 2018.

[65] ZOU, J.; SCHIEBINGER, L.. **Ai can be sexist and racist—it's time to make it fair**, 2018.

[66] THORPE, S. J.; FABRE-THORPE, M.. **Seeking categories in the brain**. Science, 291(5502):260–263, 2001.

[67] MATHIEU, M. F.; ZHAO, J. J.; ZHAO, J.; RAMESH, A.; SPRECHMANN, P. ; LECUN, Y.. **Disentangling factors of variation in deep representation using adversarial training**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, p. 5040–5048, 2016.

[68] HADSELL, R.; CHOPRA, S. ; LECUN, Y.. **Dimensionality reduction by learning an invariant mapping**. In: 2006 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'06), volumen 2, p. 1735–1742. IEEE, 2006.

[69] DENTON, E. L.; OTHERS. **Unsupervised learning of disentangled representations from video**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, p. 4414–4423, 2017.

[70] HARSH JHA, A.; ANAND, S.; SINGH, M. ; VEERAVASARAPU, V.. **Disentangling factors of variation with cycle-consistent variational auto-encoders**. In: PROCEEDINGS OF THE EUROPEAN CONFERENCE ON COMPUTER VISION (ECCV), p. 805–820, 2018.

[71] RADFORD, A.; METZ, L. ; CHINTALA, S.. **Unsupervised representation learning with deep convolutional generative adversarial networks**. CoRR, abs/1511.06434, 2015.

[72] SIEGELMANN, H. T.; SONTAG, E. D.. **On the computational power of neural nets**. Journal of computer and system sciences, 50(1):132–150, 1995.

[73] HYÖTYNIEMI, H.. **Turing machines are recurrent neural networks**. Proceedings of step, 96, 1996.

[74] HOCHREITER, S.; BENGIO, Y.; FRASCONI, P.; SCHMIDHUBER, J. ; OTHERS. **Gradient flow in recurrent nets: the difficulty of learning long-term dependencies**, 2001.

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

[76] CHO, K.; VAN MERRIENBOER, B.; ÇAGLAR GÜLÇEHRE; BAHDANAU, D.; BOUGARES, F.; SCHWENK, H. ; BENGIO, Y.. **Learning phrase representations using rnn encoder-decoder for statistical machine translation**. ArXiv, abs/1406.1078, 2014.

[77] XU, K.; BA, J.; KIROS, R.; CHO, K.; COURVILLE, A.; SALAKHUDINOV, R.; ZEMEL, R. ; BENGIO, Y.. **Show, attend and tell: Neural image caption generation with visual attention**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 2048–2057, 2015.

[78] LUONG, T.; PHAM, H. ; MANNING, C. D.. **Effective approaches to attention-based neural machine translation**. 2015.

[79] CHILD, R.; GRAY, S.; RADFORD, A. ; SUTSKEVER, I.. **Generating long sequences with sparse transformers**. ArXiv, abs/1904.10509, 2019.

[80] VERGA, P.; STRUBELL, E. ; MCCALLUM, A.. **Simultaneously self-attending to all mentions for full-abstract biological relation extraction**. 2018.

[81] GEHRING, J.; AULI, M.; GRANGIER, D.; YARATS, D. ; DAUPHIN, Y. N.. **Convolutional sequence to sequence learning**. In: PROCEEDINGS OF THE 34TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING-VOLUME 70, p. 1243–1252. JMLR. org, 2017.

[82] NGUYEN, D. Q.; VERSPOOR, K. M.. **Convolutional neural networks for chemical-disease relation extraction are improved with character-based word embeddings**. 2018.

[83] HUANG, Y. Y.; WANG, W. Y.. **Deep residual learning for weakly-supervised relation extraction**. 2017.

[84] ZENG, W.; LIN, Y.; LIU, Z. ; SUN, M.. **Incorporating relation paths in neural relation extraction**. 2016.

[85] JIANG, X.; WANG, Q.; LI, P. ; WANG, B.. **Relation extraction with multi-instance multi-label convolutional neural networks**. In: PROCEEDINGS OF COLING 2016, THE 26TH INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS: TECHNICAL PAPERS, p. 1471–1480, 2016.

[86] MIWA, M.; BANSAL, M.. **End-to-end relation extraction using lstms on sequences and tree structures**. ArXiv, abs/1601.00770, 2016.

[87] ZHANG, M.; ZHANG, Y. ; FU, G.. **End-to-end neural relation extraction with global optimization**. In: PROCEEDINGS OF THE 2017 CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, p. 1730–1740, 2017.

[88] KATIYAR, A.; CARDIE, C.. **Going out on a limb: Joint extraction of entity mentions and relations without dependency trees**. In: PROCEEDINGS OF THE 55TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (VOLUME 1: LONG PAPERS), p. 917–928, 2017.

[89] AMMAR, W.; PETERS, M.; BHAGAVATULA, C. ; POWER, R.. **The ai2 system at semeval-2017 task 10 (scienceie): semi-supervised end-to-end entity and relation extraction**. In: PROCEEDINGS OF THE 11TH INTERNATIONAL WORKSHOP ON SEMANTIC EVALUATION (SEMEVAL-2017), p. 592–596, 2017.

[90] CHIKKA, V. R.; KARLAPALEM, K.. **A hybrid deep learning approach for medical relation extraction**. ArXiv, abs/1806.11189, 2018.

[91] DEHGHANI, M.; GOUWS, S.; VINYALS, O.; USZKOREIT, J. ; KAISER, L.. **Universal transformers**. ArXiv, abs/1807.03819, 2019.

[92] IOFFE, S.; SZEGEDY, C.. **Batch normalization: Accelerating deep network training by reducing internal covariate shift**. ArXiv, abs/1502.03167, 2015.

[93] SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A. ; CHEN, L.-C.. **Mobilenetv2: Inverted residuals and linear bottlenecks**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 4510–4520, 2018.

[94] YANG, H.; LEE, H.. **Research trend visualization by mesh terms from pubmed**. International journal of environmental research and public health, 15(6):1113, 2018.

[95] GU, J.; SUN, F.; QIAN, L. ; ZHOU, G.. **Chemical-induced disease relation extraction via convolutional neural network**. Database, 2017, 2017.

[96] ZHOU, H.; DENG, H.; CHEN, L.; YANG, Y.; JIA, C. ; HUANG, D.. **Exploiting syntactic and semantics information for chemical–disease relation extraction**. Database, 2016, 2016.

[97] MANDYA, A.; BOLLEGALA, D.; COENEN, F. ; ATKINSON, K.. **Combining long short term memory and convolutional neural network for cross-sentence n-ary relation extraction**. ArXiv, abs/1811.00845, 2018.

[98] NGUYEN, D. Q.; VERSPOOR, K.. **End-to-end neural relation extraction using deep biaffine attention**. In: EUROPEAN CONFERENCE ON INFORMATION RETRIEVAL, p. 729–738. Springer, 2019.

[99] LUO, Y.; TANG, J.; YAN, J.; XU, C. ; CHEN, Z.. **Pre-trained multi-view word embedding using two-side neural network**. In: TWENTY-EIGHTH AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2014.

[100] ROTSZTEJN, J.; HOLLENSTEIN, N. ; ZHANG, C.. **Eth-ds3lab at semeval-2018 task 7: Effectively combining recurrent and convolutional neural networks for relation classification and extraction**. 2018.

[101] LUAN, Y.; OSTENDORF, M. ; HAJISHIRZI, H.. **The uwnlp system at semeval-2018 task 7: Neural relation extraction model with selectively incorporated concept embeddings**. In: PROCEEDINGS

OF THE 12TH INTERNATIONAL WORKSHOP ON SEMANTIC EVALUATION, p. 788–792, 2018.

[102] NOORALAHZADEH, F.; ØVRELID, L. ; LØNNING, J. T.. **Sirius-ltg-uio at semeval-2018 task 7: Convolutional neural networks with shortest dependency paths for semantic relation extraction and classification in scientific papers**. 2018.

[103] PRATAP, B.; SHANK, D.; OSITELU, O. ; GALBRAITH, B.. **Talla at semeval-2018 task 7: Hybrid loss optimization for relation classification using convolutional neural networks**. In: PROCEEDINGS OF THE 12TH INTERNATIONAL WORKSHOP ON SEMANTIC EVALUATION, p. 863–867, 2018.

[104] DEVLIN, J.; CHANG, M.-W.; LEE, K. ; TOUTANOVA, K.. **Bert: Pre-training of deep bidirectional transformers for language understanding**. 2019.

A
Definição do Modelo

A.1
Cálculo do vetor de alocação dinâmica paralela

```
def write_allocation_weights(self, usage):
  """ Alocacao Dinamica Paralela """

  # epsilon evita posicoes de memoria iguais
  eps_sorted = tf.reshape(\
  tf.range(self._memory_size)*_EPSILON,\
  [1,1,self._memory_size])
  usage_sorted = eps_sorted + (1 - _EPSILON) * usage

  # criacao da mascara binaria g
  greather = tf.expand_dims(usage_sorted,3) -
  tf.expand_dims(usage_sorted,2)
  greather = tf.cast(greather>0,tf.float32)

  # produto acumulado de valores de uso de memoria previos
  prod = (1-greather) + greather*tf.expand_dims(usage,2)
  phi = tf.reduce_prod(prod, axis=3)

  # Calculo do vetor de alocacao dinamica paralela
  return tf.expand_dims((1 - usage)*phi*self.mask, axis=2)
```

Algoritmo A.1: Cálculo do vetor de alocação dinâmica paralela

B
Resultados do estudo de caso 1.

B.1
Tabela de resultados da tarefa REN para todas as configurações

Junto à redução da frequência de aprendizado da tarefa REN, pode-se observar que a adição do módulo TCA reforça ainda mais a tarefa com maior frequência (tarefa ER), reduzindo o f1-score da tarefa REN. Após acrescentamos dados ruidosos, o f1-score aumenta novamente.

Modelo	Classe	Precisão	Recall	F1-score
UTRE	Disease	72,06	67,75	69,82 ± 0,46
	Chemical	75,16	73,05	78,58 ± 0,50
NTCRE-CPD	Disease	70,65	67,42	**68,99 ± 0,47**
	Chemical	88,31	68,63	72,23 ± 0,39
UTRE-CPD	Disease	69,70	67,94	68,80 ± 0,60
	Chemical	88,24	68,36	77,00 ± 0,41
NTCRE-CPD	Disease	71,42	67,7	**69,31 ± 0,60**
(+ Data)	Chemical	88,74	68,01	76,94 ± 0,39
UTRE-CPD	Disease	70,16	68,26	69,19 ± 0,52
(+ Data)	Chemical	88,87	68,16	77,14 ± 0,35

Tabela B.1: Tabela de resultados f1-score, precisão, recall da tarefa REN do primeiro estudo de caso.

C
Resultados do estudo de caso 2.

C.1
Tabela de resultados da tarefa REN para todas as configurações

	Precisão	Recall	F1-score
Chemical	92,93	93,86	93,39
Disease	84,40	85,99	85,19
Species	98,32	98,70	98,51
Gene	87,89	89,39	88,63
Protein	88,63	90,40	89,50
DNAMutation	93,50	91,59	92,53
SNP	98,03	85,14	91,13

Tabela C.1: Tabela de resultados f1-score, precisão, recall da tarefa REN do segundo estudo de caso.