UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Rodrigo Ribeiro Caputo

Deep learning aplicado a dados estruturados como cubos OLAP

São João del-Rei

Fevereiro de 2020

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Rodrigo Ribeiro Caputo

Deep learning aplicado a dados estruturados como cubos OLAP

Dissertação apresentada como requisito para obtenção do título de mestre em Ciências no Curso de Mestrado do Programa de Pós Graduação em Ciência da Computação da UFSJ.

Orientador: Edimilson Batista dos Santos

Universidade Federal de São João del-Rei – UFSJ

Mestrado em Ciência da Computação

São João del-Rei

Fevereiro de 2020

Rodrigo Ribeiro Caputo

Deep Learning aplicado a dados estruturados como cubos OLAP

Dissertação apresentada como requisito para obtenção do título de mestre em Ciências no Curso de Mestrado do Programa de Pós Graduação em Ciência da Computação da UFSJ.

Trabalho aprovado. São João del-Rei, 17 de fevereiro de 2020:

Prof. Edimilson Batista dos Santos, Presidente
Orientador

Prof. Laurence Rodrigues do Amaral
Universidade Federal de Uberlandia

Prof. Leonardo Chaves Dutra da Rocha
Universidade Federal de Minas Gerais

São João del-Rei
17 de fevereiro de 2020

Agradecimentos

Agradeço a Deus pela minha sorte. Aos meus pais Marlise e Rosauro pelo exemplo de caráter e por me darem todas as condições para seguir meus sonhos. Ao meu irmão Bruno pela referência, companheirismo, e incentivo para estudar computação. Aos meus avós Antônio, Maria, Aparecida e Antônia *(in memoriam)* pela sabedoria que transmitem. À minha sobrinha Maria Cecília por trazer esperança no futuro. Aos meus amigos e todas as pessoas queridas, pela compreensão nos momentos em que estive ausente.

Agradecimentos especiais ao meu orientador professor Edimilson, pela paciência e pela incrível capacidade de me direcionar em todos os momentos destes trabalhos. Ao Sicoob Credivertentes pelo apoio desde o início do projeto, acreditando em mim, apoiando a ciência e sempre cooperando com todos à sua volta. Ao professor Leonardo pelas contribuições e por disponibilizar os recursos computacionais de alto desempenho para execução dos experimentos, juntamente com seus alunos do laboratório pelo auxílio nas configurações do ambiente.

Agradeço também todos os professores da Universidade Federal de São João del-Rei por todo o conhecimento compartilhado e a própria UFSJ por ter sido minha segunda casa por alguns anos. Estendo o agradecimento a todos os meus professores desde sempre, em especial ao meu professor de física do ensino médio Ronaldo, que me ensinou a acreditar no impossível.

Por fim, agradeço à rádio 89 pela trilha sonora nas várias madrugadas de estudos, aos meus colegas estudantes da computação e de todas as áreas do conhecimento que buscam, através da ciência, tornar nosso mundo um lugar melhor.

Resumo

O processo de Descoberta de Conhecimento de Bases de Dados (KDD) aplicado em dados estruturados é um desafio onde diversas abordagens conseguiram relativo sucesso, já há algum tempo. Muitas destas abordagens baseiam-se no Aprendizado de Máquina tradicional e seus paradigmas. Já o processamento de dados não-estruturados se popularizou apenas mais recentemente, com o surgimento do *Deep Learning*, uma abordagem baseada em Redes Neurais Artificiais (RNAs). Embora as RNAs sejam um método clássico do Aprendizado de Máquina, a evolução das técnicas de otimização e do poder computacional, democratizado principalmente com o uso das GPUs, impulsionaram suas aplicações em diversos domínios como processamento de imagens, áudio, linguagem natural, dentre outros. Este trabalho investiga como estas novas técnicas podem ser aplicadas a fim de melhorar os resultados em modelos gerados a partir de dados estruturados. Desta investigação, surgiu a proposta da *OlapNet*, uma arquitetura de Rede Neural Convolucional (CNN) que se baseia em operações implícitas de cubos de dados como entrada. Formalmente identificou-se que esta arquitetura é capaz de superar os resultados de um grupo específico de transformações realizadas sobre os dados, permitindo em partes a automação da etapa de transformação de dados no processo de KDD. Para verificar a proposta empiricamente, foi utilizada uma amostra de dados de uma base real contendo dados anonimizados do histórico de endividamento dos clientes de uma instituição financeira. A partir destes dados, foi modelado um problema de classificação preditiva para estimar a probabilidade de um cliente qualquer contratar novos créditos nos próximos três meses. Assim, foram utilizados métodos tradicionais de Aprendizado de Máquina e variações de CNNs, incluindo a arquitetura proposta neste trabalho. Os resultados mostraram que em quase todos os casos as CNNs superam os métodos tradicionais, indicando que os *Feature Maps* gerados a partir dos *kernels* convolucionais aprendidos pela rede são capazes de extrair características relevantes dos dados. Estes *kernels* não somente permitem extrair características, como reduzem a complexidade da rede ao delimitar uma vizinhança para cada pixel e diminuem a propensão de ocorrer *overfitting*. Dentre as CNNs analisadas nos cenários de testes, a *OlapNet* apresentou os melhores resultados, indicando que a arquitetura proposta é bastante promissora neste tipo de aplicação.

Palavras-chaves: Aprendizado Profundo; Aprendizado de Máquina; OLAP; Descoberta de Conhecimento em Base de Dados (KDD); Redes Neurais Artificiais; Redes Neurais Convolucionais

Abstract

The Knowledge Discovery in Database (KDD) process applied to structured data is a challenge in which several approaches have been relatively successful for some time. Many of these approaches are based on traditional machine learning and its paradigms. On the other hand, the processing of unstructured data has become more popular recently, with the emergence of Deep Learning, an approach based on Artificial Neural Networks (ANNs). Although ANNs is a classic method of Machine Learning, the evolution of optimization techniques and computational power, democratized mainly with the use of GPUs, increase their applications in several domains such as image processing, audio, natural language and others. This work investigates how these new techniques can be applied to improve the results in models generated from structured data. Through this investigation, OlapNet's proposal emerged, a Convolutional Neural Network (CNN) architecture that is based on implicit operations of data cubes as input. Formally, it was concluded that this architecture is able to overcome the results of a specific set of transformations carried out on the data, allowing in part an automation of the data transformation stage in the KDD process. To verify the proposal empirically, a sample of data from a real database was used containing debt data from anonymous clients of a financial institution. Based on this data, a classification problem was modeled to estimate the likelihood of a client getting new credits in the next three months. Therefore, traditional machine learning methods and variations of CNNs were used including a proposal for this work. The results showed that in almost all cases the CNNs overcame the traditional methods, indicating that the Feature Maps generated from the convolutional kernels learned by the network are able to extract relevant features from the data. These "kernels" not only allow the features to be extracted, but also as decrease the complexity of the network by delimiting the neighborhood for each pixel and decrease the propensity to occur overfitting. Among the CNNs analyzed in the test scenarios, the OlapNet presented the best results indicating that the proposed architecture is very promising in this type of application.

Key-words: Deep Learning; Machine Learning; OLAP; Knowledge Discovery in Database (KDD); Artificial Neural Networks; Convolutional Neural Networks

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ADAM	*Adaptive Moment Estimation*
AUC	Área sob a Curva ROC
BI	*Business Intelligence*
CNN	Redes Neurais Convolucionais
ETL	Extração, Transformação e Carga de dados
GD	Gradiente Descendente
GPU	Unidade de Processamento Gráfico
KDD	Descoberta de Conhecimento em Bases de dados *(Knowledge Discovery in Databases)*
KNN	Algoritmo *K-Nearest Neighbors*
OLAP	Processamento Analítico Online *(Online Analytical Processing)*
OLTP	Processamento de Transação Online *(Online Transaction Processing)*
MLP	Perceptron Multi-camadas *(MultiLayer Perceptron*
ReLU	Unidade Linear Retificada *(Rectified Linear Unit)*
RNA	Redes Neurais Artificiais
ROC	Curva Característica de Operação do Receptor
SGD	Gradiente Descendente Estocástico *(Stochastic Gradient Descent)*

Sumário

1 Introdução

O desenvolvimento de tecnologias da informação resultou na produção contínua de enorme quantidade de dados. Ao mesmo tempo, frequentemente é necessário realizar análises destes conteúdos de alguma forma, para os mais variados fins. Entretanto, tal atividade não é trivial, e faz-se necessário o uso de ferramentas de apoio à essa tarefa.

Um dos fatores dificultantes é a origem e características de cada informação. *Softwares* tradicionais geralmente fazem os registros dos seus dados em arquivos ou em Sistemas Gerenciadores de Banco de Dados que os organizam em tabelas ou em outros tipos de estruturas. Por esta razão, são chamados dados estruturados. Outro tipo de informação como textos em páginas da internet são mais difíceis de organizar, por isso são chamados semi-estruturados. Por fim, temos fotos, vídeos, sons, dentre outros tipos de conteúdo que são não-estruturados.

Independentemente do tipo de dado, para qualquer uma das três classificações existem necessidades de análises específicas. Para dados estruturados, de uma empresa por exemplo, uma abordagem bastante popular é utilizar ferramentas OLAP *(Online Analytical Processing)*. Como os dados estão organizados de maneira bem definida, existem operações OLAP específicas (ver Seção 2.1.1) que possibilitam ao usuário fazer análises descritivas e diagnósticas, além de obter *insigths* sobre o negócio. A desvantagem dessa abordagem é justamente por ser manual e possuir uma alta dependência do julgamento humano, dificultando automações e podendo gerar outros problemas.

Uma outra abordagem envolve o processo de Descoberta de Conhecimento em Bases de Dados - KDD (ver Seção 2.2) que, dentre outras, possui a etapa de Mineração de Dados. Nesta etapa geralmente utiliza-se um algoritmo de Aprendizado de Máquina adequado ao problema em questão para que ele seja capaz de identificar padrões existentes nos dados. Por ser um método computacional, este naturalmente pode ser automatizado, eliminando assim alguns dos problemas existentes no modelo de análise tradicional, utilizando manualmente ferramentas OLAP, por exemplo.

Como será visto na Seção 2.3, existem modelos estatísticos, baseados em funções de distância, em regras de associação, dentre outros. Cada um deles possuem vantagens, desvantagens e obviamente limitações. Este trabalho foca especificamente nos modelos baseados em Redes Neurais Artificiais (RNA).

Por algum tempo, o custo computacional inviabilizou a aplicação de RNAs em diversos domínios. Porém, a partir do aumento considerável do poder de processamento dos computadores modernos e das melhorias aplicadas nos métodos, foi possível a aplicação desta tecnologia em cenários até então impraticáveis. Isso possibilitou o surgimento de

redes neurais cada vez mais robustas, com estruturas mais complexas e camadas mais "profundas", o que resultou no surgimento do termo *Deep Learning*, ou Aprendizado Profundo.

Os principais resultados obtidos nesse novo paradigma envolvem dados semi-estruturados e principalmente não-estruturados, aplicando diferentes tipos de RNAs em domínios como visão computacional, reconhecimento automático de fala, processamento de linguagem natural, dentre outros. Em problemas de visão computacional, em específico, um tipo de RNA bastante popular é a Rede Neural Convolucional (CNN), que se difere das RNAs tradicionais onde todos os neurônios de uma camada estão totalmente conectados à camada posterior. Ela possui apenas um subgrupo de neurônios interconectados, definidos pelo *kernel*, a fim de realizar a operação de convolução, o que tende a torná-la mais rápida de se ajustar e mais generalista.

1.1 Objetivos

O objetivo deste trabalho é investigar o uso de CNNs em bases de dados estruturados, criando uma organização nestes dados baseada em cubos OLAP. Com isso, a dimensionalidade matricial presente nas imagens (2D ou 3D) seria aplicada no próprio cubo, ou em um cubo derivado. A hipótese é que, se visões resultantes de operações OLAP são úteis para analistas humanos, então elas podem ser úteis para as RNAs também.

Para isso, é realizado um estudo de caso real de um problema de análise preditiva da propensão dos clientes de uma instituição financeira contratarem crédito. Neste caso, deseja-se criar um modelo que indique a probabilidade de cada cliente contratar crédito em um futuro próximo. A partir destas probabilidades, é criado um Rank da maior para a menor, a fim de realizar ações comerciais posteriores.

Os dados utilizados consistem do histórico financeiro de cada cliente nos últimos 12 meses em relação às Modalidades de crédito e Prazo de pagamento. Por se tratar de dados estruturados, busca-se uma abordagem adequada para extrair o conhecimento desejado. Ao final do processo, espera-se que a solução não apenas resolva o problema investigado neste experimento, mas também contribua para a realização de tarefas futuras com características similares.

1.1.1 Contribuições Esperadas

Após os estudos dos temas correlatos, implementação e execução dos experimentos propostos, espera-se as seguintes contribuições:

- Construir uma metodologia baseada em *Deep Learning* para a extração de conhecimento em dados estruturados;

- Propor uma arquitetura genérica de RNA para problemas como o investigado neste trabalho;
- Compreender o incremento na qualidade da classificação a partir da análise automática de diferentes visões resultantes das operações sobre um cubo OLAP;
- Gerar valor para o negócio, a partir da criação de um modelo preditivo eficiente.

1.2 Organização do Trabalho

A seguir, no Capítulo 2, será apresentado o referencial teórico sobre os conceitos utilizados neste trabalho. Inicialmente na Seção 2.1 são abordados os Cubos de Dados e o processo de Data Warehousing, seguido pela definição do processo de Descoberta de Conhecimento em Bases de Dados (KDD), em 2.2. Na seção seguinte (2.3), são definidos os problemas de aprendizado de máquina, sendo indicados cinco métodos tradicionalmente utilizados em tarefas de classificação. Entraremos então na Seção 2.4, que tratará em detalhes das Redes Neurais Artificiais, desde o funcionamento básico da estrutura de decisão, até as mais modernas arquiteturas de redes profundas.

Em seguida, no Capítulo 3 são apresentados os Trabalhos Relacionados envolvendo dados financeiros, extração de *features*, representação e processamento de séries temporais, cubos OLAP, Redes Neurais Convolucionais, outras RNAs modernas, maximização de AUC, dentre outros.

No Capítulo 4 é apresentada a proposta de modelagem, sendo a primeira etapa a idealização de um cubo OLAP e o devido mapeamento das possíveis funções OLAP a serem aplicadas. Finalmente é definida a *OlapNet* e o conceito utilizado na sua construção. Além disso, são escolhidos os critérios de avaliação dos resultados dos experimentos.

Já no Capítulo 5 a base de dados utilizada é apresentada em detalhes, desde o processo de extração das amostras, passando pela sumarização e normalização da base e todas as transformações de dados aplicadas. Ao final são mostrados os resultados comparativos em uma tabela de resultados, juntamente com o comparativo dos gráficos das AUCs de cada método testado.

Finalmente, no Capítulo 6 são feitas as conclusões sobre os resultados obtidos e o desenvolvimento do trabalho como um todo. Além da análise dos desempenhos dos métodos em si, é ponderado sobre a viabilidade de investir neste tipo de abordagem para o problema do estudo de caso.

2 Referencial teórico

Este capítulo fornece alguns conceitos que embasam este trabalho. Na Seção 2.1 é definido o que são Cubos de Dados, ou Cubos OLAP, abordando os Data Warehouses e as principais operações OLAP. Em seguida, na Seção 2.2, são definidas as etapas do processo de Descoberta de Conhecimento em Bases de Dados (KDD) e seus objetivos principais. Adentrando no Aprendizado de Máquina, na Seção 2.3, cada um dos métodos clássicos utilizados são apresentados. Já a Seção 2.4 trata especificamente da classe de métodos baseados em Redes Neurais Artificiais, desde os princípios básicos até redes mais avançadas como as redes Convolucionais, base fundamental da rede neural proposta. São apresentados também alguns detalhes importantes dos otimizadores e técnicas de regularização dos dados, finalizando com algumas considerações adicionais.

2.1 Data Warehouse e Cubos de Dados

Aplicações de suporte a tomada de decisão surgiram a partir os anos 1970, mas foi nos anos 1990 que elas se tornaram bastante populares. Neste período, Howard Dressner, analista do grupo Gartner cunhou o termo *Business Intelligence* (BI) que, na prática, é atualmente utilizado para descrever aplicações analíticas em geral (WATSON; WIXOM, 2007).

Sistemas de banco de dados tradicionais, em especial os bancos relacionais, são utilizados em diversas aplicações de Processamento de Transação Online *(Online Transaction Processing)* ou OLTP, onde frequentemente são realizadas inserções, atualizações e exclusões nos dados. Por outro lado, aplicações analíticas conhecidas como Processamento Analítico Online *(Online Analytical Processing)* ou OLAP focam em prover velocidades de leitura e capacidades de consulta superiores ao `SQL` tradicional, além de uma série de princípios propostos por Codd (1993).

Estes requisitos motivaram o desenvolvimento de tecnologias específicas, conhecidas como *Data Warehouse*. A partir de processos de ETL (Extração, Transformação e Carga), os dados de diversas origens são coletados para um ambiente apartado, onde são criadas bases que servirão às aplicações de BI. O processo de *Data Warehousing* como um todo pode ser visto na Figura 1.

Dentre as aplicações de BI, as ferramentas OLAP em especial permitem a realização de consultas com uma série de operações capazes de apresentar análises com informações ricas para o negócio. Para suportar eficientemente estas operações, uma estrutura especifica é utilizada, chamada de Cubo de Dados ou simplesmente Cubo OLAP.

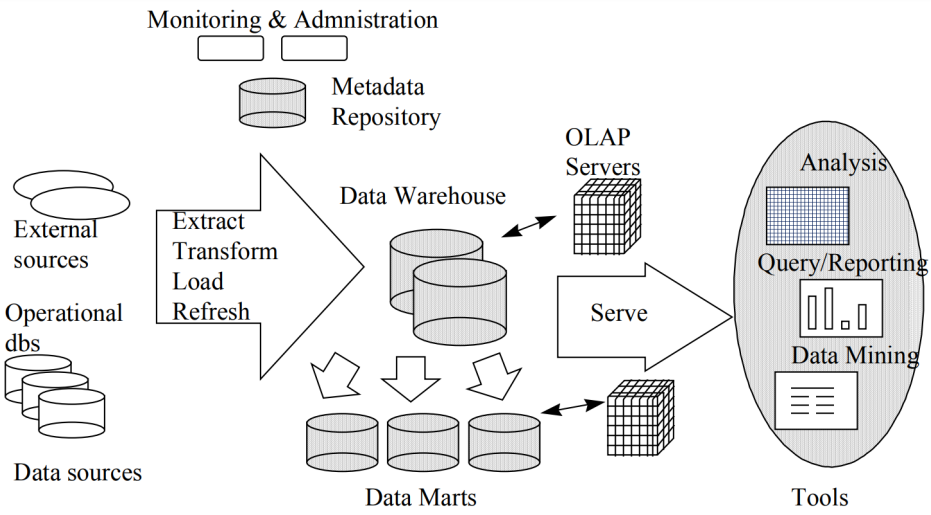


Figura 1 – Arquitetura do processo de *Data Warehousing*.
Fonte: (CHAUDHURI; DAYAL, 1997)

Os cubos de dados são produzidos a partir de tabelas de banco de dados ou de planilhas-padrão. Uma planilha-padrão é uma matriz bidimensional. Um exemplo seria uma planilha de vendas, regionais por produto para determinado período, onde os produtos são mostrados como linhas, com as receitas de vendas para cada região compreendendo as colunas, como vista na Figura 2a. Ao acrescentar uma dimensão de tempo, como os trimestres fiscais de uma organização, seria produzida uma matriz tridimensional, representada na Figura 2b. Devido a essa terceira dimensão chama-se de cubo de dados, ou hipercubo, se houverem mais dimensões.

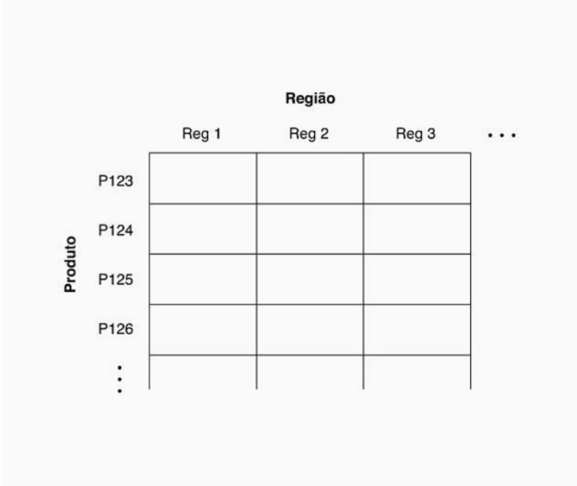
2.1.1 Operações OLAP

A partir dos Cubos de Dados, ferramentas OLAP oferecem suporte a diversas operações particulares, permitindo a realização de consultas complexas facilmente. Dentre as operações mais comuns estão:

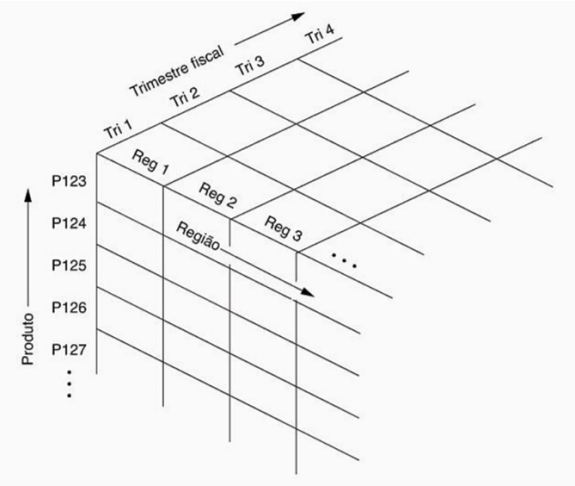
- **Roll-up**: Os dados são resumidos com generalização cada vez maior, como na Figura 2c. Neste exemplo, subcategorias dos produtos foram agrupadas, propiciando uma visão mais alto nível e facilitando a análise em uma empresa com muitos produtos. Para isso, em geral utiliza-se funções de agregação como SOMA, CONTAGEM, MÁXIMO, MÍNIMO, dentre outras.

- **Drill-down**: O complemento da função Roll-up. Fornece níveis de detalhes cada vez maiores, como em Figura 2d.

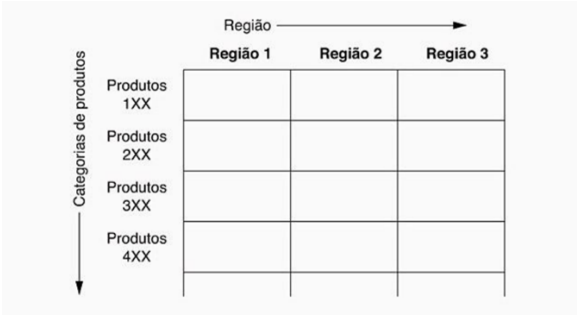
- **Rotação** (ou *Giro*): Possibilita a construção de tabelas cruzadas, através do pivotamento de uma dimensão.



(a) Exemplo de matriz bidimensional



(b) Exemplo de cubo de dados tridimensional



(c) Operação Roll-up

(d) Operação Drill-down

Figura 2 – Exemplos de Cubos de Dados e operações OLAP.
Fonte: (ELMASRI; NAVATHE, 2010)

- **Slice** e **Dice**: Operações de projeção, realizadas nas dimensões.
- **Ordenação**: Organiza os dados seguindo algum valor ordinal.
- **Seleção**: Os dados são filtrados por valor ou intervalo.
- **Atributos derivados (calculados)**: Atributos são calculados por operações sobre outros valores armazenados e derivados.

Todas estas operações fornecem grande flexibilidades aos analistas de negócio, facilitando a construção de diferentes visões rapidamente. Desta forma, as informações ficam mais claras e é possível obter *insights* valiosos, apoiando as organizações em seus processos de tomada de decisão. Entretanto, nem todas as necessidades de análise são supridas com estes recursos e neste ponto, um processo relevante pode apoiar: o KDD.

2.2 KDD: Descoberta de Conhecimento em Bases de Dados

Com a digitalização de diversos processos e, em especial, com o advento da web 2.0, as capacidades computacionais de coletar e acumular dados chegaram a um ritmo dramático. Extrair informações úteis (conhecimento) em meio a esse gigantesco volume de dados, é o processo chamado de KDD (*Knowledge Discovery in Databases*).

Um dos desafios básicos é o de lidar com altos níveis de detalhes, o que gera muitos dados e dificulta o entendimento, em comparação com visões mais resumidas, que podem omitir informações relevantes. Outras formas de apresentação podem ser mais abstratas, como uma análise descritiva, ou com maior aplicação prática, como um modelo preditivo por exemplo.

Como apresentado na Figura 3, Fayyad, Piatetsky-Shapiro e Smyth (1996) divide o processo de KDD em etapas de múltiplas iterações, onde todas podem e devem ser repetidas e refinadas, sendo elas:

- **Seleção**: A partir do entendimento do domínio de aplicação, definir quais dados são (ou podem ser) relevantes, criando uma base de dados de acordo com o objetivo da análise.
- **Pré-processamento**: Realizar a limpeza dos dados, tratando ruídos, informações faltantes e outros problemas que podem comprometer a qualidade das informações a serem geradas, fornecendo dados melhores para a etapa seguinte.
- **Transformação**: Selecionar e/ou criar atributos que melhor representem os dados, dependendo da tarefa objetivo. Usualmente, busca-se reduzir a dimensionalidade dos dados, facilitando a tarefa do método de mineração e buscando evitar o problema de *over-fitting*, ou sobre-ajuste do modelo.
- **Mineração de Dados**: Aplicar métodos de acordo com a tarefa, podendo ser de sumarização, classificação, regressão, agrupamento, dente outras, a fim de identificar os padrões existentes.
- **Interpretação** e **Avaliação**: Interpretar os resultados e analisá-los de acordo com métricas de avaliação adequadas. Revisar todo o processo buscando melhorias e refinamentos nas etapas anteriores.

A parte final pode envolver a visualização dos padrões ou modelos obtidos, dependendo do objetivo da análise. O conhecimento obtido poderá então ser reportado aos interessados ou, se for o caso, incorporado ao domínio de aplicação original, aperfeiçoando seus processos.

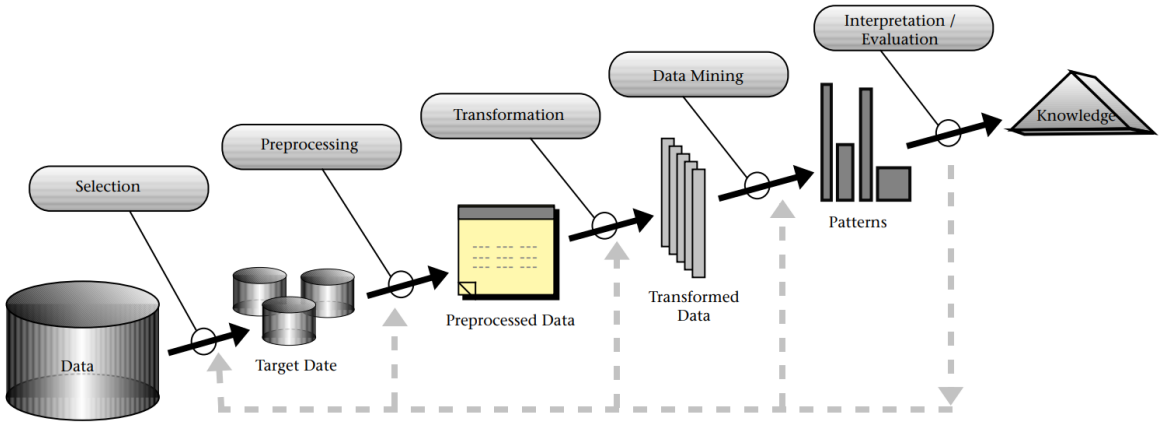


Figura 3 – Visão geral das etapas que compõe o processo de KDD.
Fonte: (FAYYAD, 1996)

O autor argumenta que os métodos mais tradicionais envolvem muitas análises e interpretação manuais, o que torna o processo lento, custoso e altamente subjetivo. Um exemplo disso são as aplicações OLAP, bastante populares como já mencionado. Neste ponto, dentro do processo de KDD, a mineração de dados possui alta capacidade de automação, em especial os métodos de aprendizado de máquina.

2.3 Aprendizado de Máquina

O objetivo do aprendizado de máquina é construir modelos computacionais que possam adaptar-se e aprender a partir da experiência contida nos dados. Formalmente, Mitchell (1997) define com o Teorema 2.3.1.

Teorema 2.3.1. *Um programa de computador aprende com a experiência E em relação a alguma classe de tarefas T e medida de desempenho P, se seu desempenho em T, medido por P, melhorar com a experiência E.*

Desta forma, ao longo do tempo surgiram diversas propostas de algoritmos que atendem a este objetivo. Estes métodos usualmente são classificados em Aprendizado Supervisionado, Aprendizado Semi-supervisionado, Aprendizado Não-supervisionado e Aprendizado por Reforço. Neste trabalho investigaremos especificamente um problema de Aprendizado Supervisionado.

Um método de Aprendizado de Máquina Supervisionado usualmente possui como entrada um conjunto de N exemplos[1] de treinamento $\{(x_1, y_1), ..., (x_n, y_n)\}$ rotulados com

[1] também chamados de casos, ou instâncias

os valores y de uma função f desconhecida $y = f(x)$, onde os valores x_i são vetores da forma $< x_{i1}, x_{i2}, ..., x_{iM} >$ cujos componentes são valores discretos ou contínuos relacionados aos atributos $A = \{A_1, A_2, ..., A_M\}$, ou seja, x_{ij} denota o valor do atributo A_j do exemplo i.

Dado esse conjunto de exemplos de treinamento, o algoritmo induz uma hipótese h que deve aproximar a verdadeira função f, tal que dados os valores x de um novo exemplo, h prediz um valor y correspondente. No caso dos valores y pertencerem a um conjunto discreto de N_{C_l} classes, isto é, $y \in \{C_1, ..., C_{N_{C_l}}\}$, a tarefa de aprendizado é chamada de **classificação**. No entanto, se $y \in \mathbb{R}$, essa tarefa é denominada de **regressão** (LEE, 2005).

Assim como feito em (YEH; LIEN, 2009), neste trabalho foram realizados experimentos envolvendo um problema de classificação em uma base de dados real. Os métodos utilizados foram: KNN, Regressão Logística, Análise Discriminante, Naïve Bayes, Árvore de Decisão e Redes Neurais Artificiais. A seguir são apresentadas as suas principais características.

2.3.1 KNN

O algoritmo de vizinhos mais próximos (*K-Nearest Neighbors* ou simplesmente *K-NN*) é um dos mais simples de todos os algoritmos de aprendizagem de máquina. Para realizar sua classificação, ele baseia-se nos exemplos mais "próximos"da instância em questão. Esta proximidade é definida em termos de uma função de distância. Desta forma, o método analisa as classes dos K vizinhos mais próximos e atribui a classe mais comum a eles. É considerado um algoritmo de aprendizado preguiçoso, pois posterga a criação do seu modelo até o momento da classificação. Apesar de se mostrar competitivo com as demais estratégias, o *K-NN* tem algumas desvantagens como consumir muita memória (para armazenar os exemplos, visto que não existe uma simples função de classificação), além de ser muito sensível à função de distância e à escolha do valor para K, como visto na Figura 4.

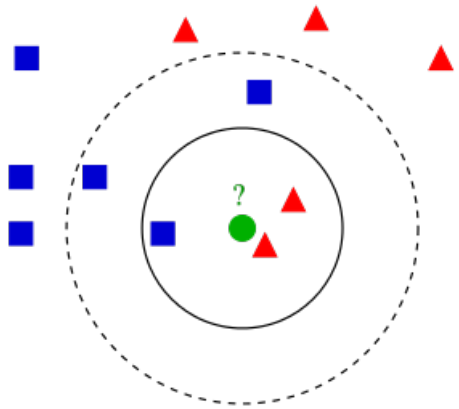


Figura 4 – Visualização gráfica do funcionamento do KNN.
Fonte: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg

2.3.2 Regressão Logística

Regressão Logística pode ser considerado um modelo especial de regressão linear. A técnica produz, a partir de um conjunto de observações, um modelo que permite a predição de valores categóricos, frequentemente binários, a partir de uma série de variáveis explicativas (atributos). O modelo específica uma função linear que aproxima o resultado da variável de classe à média dos valores observados. A maior vantagem da regressão logística é que obtém-se uma fórmula probabilística simples de classificação. Por outro lado, esta abordagem não é capaz de resolver problemas não-linearmente separáveis, como será visto na Seção 2.4.2.

2.3.3 Análise Discriminante

Também conhecida como regra de Fisher, a Análise Discriminante é uma alternativa à Regressão Logística. Ela assume que as variáveis independentes (atributos) seguem uma distribuição normal com média e parâmetros de covariância comuns. O objetivo da regra de Fisher é maximizar a distância entre grupos diferentes e minimizar a distância entre os próprios grupos. Desta forma, as vantagens e desvantagens da Análise Discriminante se assemelham com as da Regressão Logística.

2.3.4 Naive Bayes

Naive Bayes é uma técnica simples para a construção de classificadores, baseado na teoria de *Bayes*. Não é um único algoritmo para treinar tais classificadores, mas uma família de algoritmos baseados em um princípio comum: todos os classificadores Bayesianos "Naive" assumem que o valor de um atributo particular é independente do valor de qualquer outro atributo, dada a variável de classe. Tal consideração nem sempre retrata fielmente a realidade, porém, desta forma o Naive Bayes consegue ser eficiente computacionalmente e historicamente apresenta resultados comparáveis a outros classificadores.

2.3.5 Árvore de Decisão

Árvores de Decisão são estruturas onde cada nó interno representa um teste em um atributo, cada ramo representa uma saída de cada teste e os nós folha representam as classes. Uma Árvore de Decisão pode ser criada a partir de um algoritmo que automatiza a geração da árvore, como por exemplo o ID3. Esses algoritmos se baseam na observação de todos os atributos das instâncias de treinamento e suas respectivas classes, buscando minimizar a impureza, ou seja, a variabilidade das classificações em relação aos valores indicados. Elas possuem a vantagem de resultar em simples regras de classificação e por consequência permite fácil interpretabilidade do modelo criado. Elas são capazes de criar regras que representam modelos não-lineares, entretanto as árvores geradas a partir destes

métodos podem variar bastante de acordo com os dados observados, além de sofrer com *overfitting* em alguns casos.

A Figura 5 mostra inicialmente uma árvore de decisão que realiza inferências a partir de duas variáveis: *start* e *age*. Baseado em seus valores ela indica se ocorreu ou não determinada característica, através das classes *present* ou *absent*, respectivamente.

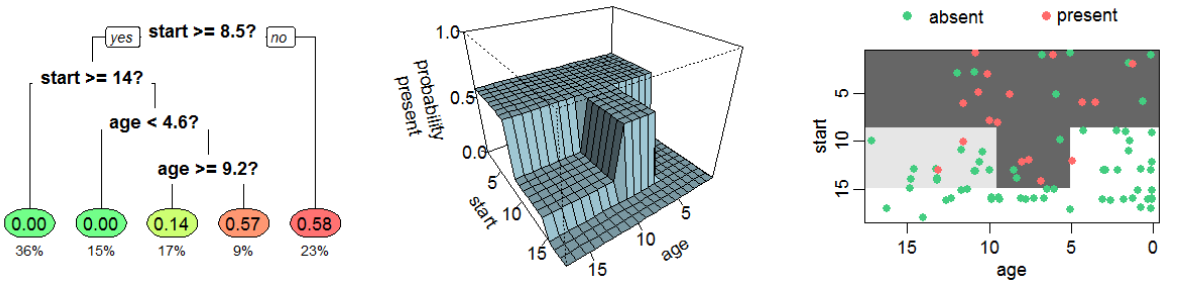


Figura 5 – Exemplo de Árvore de Decisão com sua superfície de decisão em 3D e 2D, respectivamente.
Fonte: https://en.wikipedia.org/wiki/Decision_tree_learning#/media/File:Cart_tree_kyphosis.png

2.4 Redes Neurais Artificiais

Uma Rede Neural Artificial (RNA) é um modelo matemático inspirado na interconexão dos neurônios de um cérebro biológico. Essa rede é construída a partir de um denso conjunto de unidades básicas, em especial o *Perceptron*. Estas unidades possuem pesos entre suas conexões que podem ser automaticamente "aprendidos"em um processo de *treinamento*. Para isso, um método como o *backpropagation* ajusta os parâmetros da rede buscando otimizar uma *função objetivo* que avaliará a qualidade do modelo.

A RNA é um método prático, popular, e de uso geral para o aprendizado a partir de dados. Mitchell (1997) cita algumas características de problemas onde esta abordagem é apropriada, são elas:

- Instâncias são representadas por vários pares de atributo-valor.
- A saída da função objetivo pode ser valores discretos, reais ou um vetor deles.
- Os exemplos de treinamento podem conter ruídos.
- Rápida avaliação depois que o treinamento é concluído.
- Problemas que permitam um longo período de treinamento e que a interpretabilidade do modelo não é um requisito.

2.4.1 Perceptron

As primeiras abordagens envolvendo um neurônio (ou um cérebro) artificial surgiram com McCulloch e Pitts (1943) e a ideia de aprendizado introduzida por Rosenblatt (1958) ao propor o *Perceptron* construíram os princípios que são utilizados até hoje nas RNAs.

Um Perceptron, ilustrado na Figura 6, recebe um vetor de valores reais de entrada, calcula uma combinação linear destes valores e produz o valor 1 se este resultado for maior que um *threshold* (limiar) e -1 caso contrário.

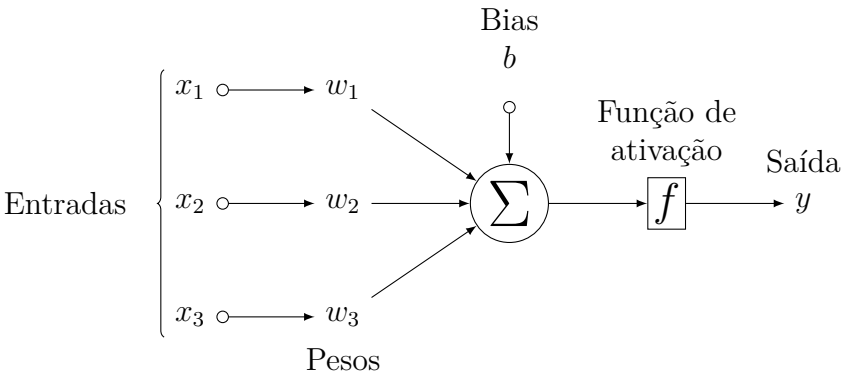


Figura 6 – Neurônio Artificial

Matematicamente, dado as entradas x_1 até x_n, a saída $o(x_1, ..., x_n)$ será computada como na Equação 2.1, onde w_i é um valor real constante, ou peso (*weight*), que determina a contribuição da entrada x_i para a saída do *Perceptron* (MITCHELL, 1997).

$$o(x_1, ..., x_n) = \begin{cases} 1 & \text{se } w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n > 0 \\ -1 & \text{caso contrário} \end{cases} \tag{2.1}$$

O peso w_0 (oposto do *threshold*) é conhecido como *Bias*, pois introduz um viés à classificação. Para simplificação, adiciona-se uma constante $x_0 = 1$, permitindo reescrever a inequação da Equação 2.1 como $\sum_{i=0}^{n} w_i x_i > 0$, ou em notação de vetores, $\vec{w} \cdot \vec{x} > 0$. Chegamos assim à Equação 2.2 que evidencia o papel da função de ativação ϕ definida na Equação 2.3.

$$o(\vec{x}) = \phi(\vec{w} \cdot \vec{x}) \tag{2.2}$$

onde

$$\phi(y) = \begin{cases} 1 & \text{se } y > 0 \\ -1 & \text{caso contrário} \end{cases} \tag{2.3}$$

O aprendizado de um *Perceptron* envolve ajustar os valores dos pesos $w_0, ..., w_n$. Assim, o espaço de hipóteses H considerado no aprendizado é o conjunto de todos os vetores de pesos com valores reais.

$$H = \{\vec{w} | \vec{w} \in \mathbb{R}^{n+1}\}$$

Regra de Treinamento do *Perceptron*

Uma forma de encontrar um vetor $\vec{w}$ adequado aos dados é iniciá-lo com valores aleatórios e iterativamente aplicar cada exemplo de treinamento ao *Perceptron*, alterando os pesos sempre que houver erros de classificação. Esse processo será repetido quantas vezes necessárias até que todos os exemplos sejam corretamente classificados. Para isso, os pesos serão modificados em cada iteração de acordo com a *Regra de Treinamento do Perceptron*, que recebe um peso w_i associado com a entrada x_i como na Equação 2.4 (MITCHELL, 1997).

$$w_i \leftarrow w_i + \Delta w_i \tag{2.4}$$

onde

$$\Delta w_i = \eta(t - o)x_i \tag{2.5}$$

tal que t é o valor correto *(target)* da classificação daquele exemplo, o é o valor de saída *(output)* gerado pelo *Perceptron*, e η uma constante positiva chamada *taxa de aprendizado (learning rate)*.

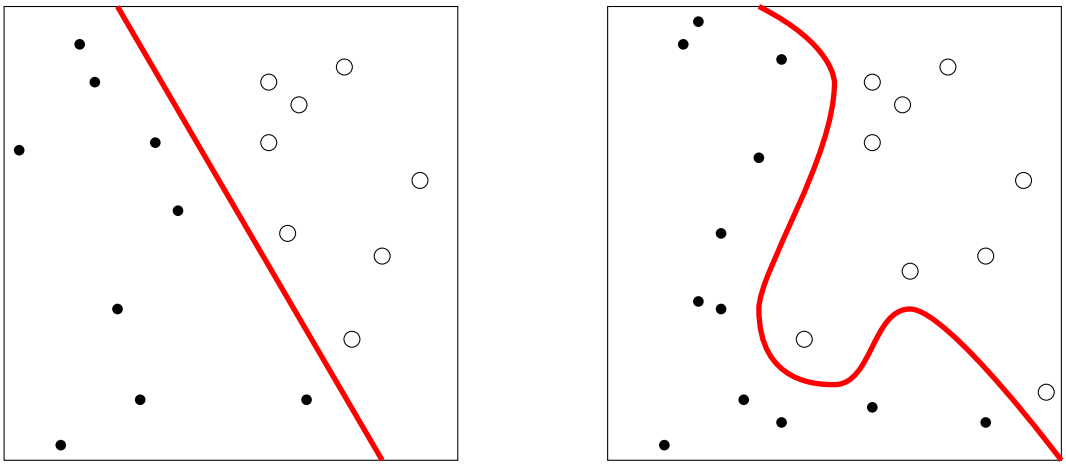
Sendo o valor de η suficientemente pequeno, esse processo irá convergir em um número finito de passos, classificando todos os exemplos corretamente, desde que os dados sejam de um *problema linearmente separável* (MINSKY; PAPERT, 1969).

2.4.2 Problemas linearmente separáveis

O *Perceptron* pode ser visto como um hiperplano de superfície de decisão no espaço n-dimensional, ou seja, um classificador linear. Para alguns problemas, como o ilustrado na Figura 7a, essa habilidade pode ser suficiente para encontrar uma solução aceitável. Entretanto, a maioria dos problemas do mundo real possuem maior complexidade, sendo necessário uma função não-linear para resolvê-lo, como visto na Figura 7b.

Um exemplo de função que ficou bastante conhecida por ser não-linearmente separável é a função booleana **XOR** (a OU exclusivo). Ela será VERDADEIRO (1) se e somente se $x_1 \neq x_2$, sendo FALSO (0) caso contrário. Graficamente a função **XOR** produz o comportamento visto na Figura 8.

Felizmente, o *Perceptron* é capaz de representar outras funções booleanas primitivas, como **AND** (E), **OR** (OU), **NAND** (¬**AND**, não-E) e **NOR** (¬**OR**, não-OU), que permitem representar qualquer outra expressão lógica, incluindo a **XOR**, através de combinações destas funções básicas. Desta forma, os *Perceptrons* podem ser agrupados criando assim uma Rede, como será apresentado na Seção 2.4.4.



(a) Linearmente Separável (b) Não-linearmente Separável

Figura 7 – Problema Linearmente Separável vs Não-linearmente Separável
Fonte: (PENG, 2013), adaptado

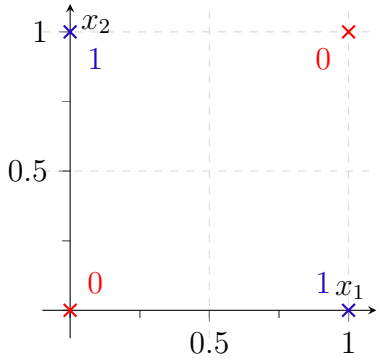


Figura 8 – Gráfico da função booleana **XOR** (OU exclusivo)

2.4.3 Gradiente Descendente

Um método conhecido como Regra Delta é utilizado para encontrar os pesos $\vec{w}$ que melhor se ajustam ao conjunto de treinamento. Desta forma, diferentemente da Regra de Treinamento do *Perceptron* apresentada na Seção 2.4.1, a Regra Delta pode ser utilizada em problemas não-linearmente separáveis. Neste caso, ela converge para a melhor aproximação do conceito objetivo através de uma busca no espaço de hipóteses H, orientada pelo *Gradiente Descendente*.

O gradiente de uma função é um vetor que indica o sentido e direção que obtém-se o maior incremento, dado um ponto especificado. Assim, utiliza-se uma função de custo E para mensurar o erro de treinamento, a fim de minimizá-lo. Por esta razão, o processo de aprendizado segue o sentido oposto do gradiente, ou seja, descendente, como ilustrado[2] na Figura 9.

[2] Adaptado de https://gist.github.com/felipessalvatore/c2e1c09dfcb8710b847e2457620f8204

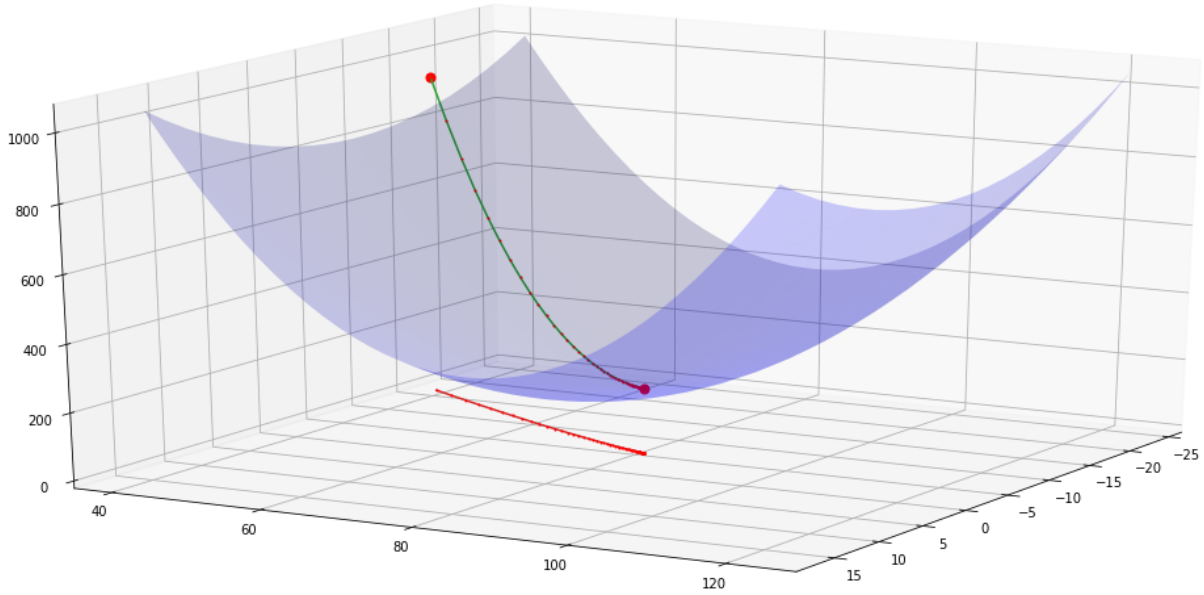


Figura 9 – Ilustração da busca baseada no Gradiente descendente. Os eixos horizontais representam os componentes de $\vec{w}$, enquanto o eixo vertical representa $E(\vec{w})$, ou seja, o erro calculado pela função de custo, de acordo com $\vec{w}$

Embora existam muitas maneiras de definir esta função E, um critério comum e conveniente é dado pela Equação 2.6.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \tag{2.6}$$

onde D é o conjunto de treinamento, t_d é o valor correto *(target)* da classificação do exemplo d e o_d é o valor de saída *(output)* gerado pelo *Perceptron* para o exemplo d. Em outras palavras, $E(\vec{w})$ é a metade do quadrado da diferença entre o saída objetivo t_d e a saída gerada o_d, somando-se todos os exemplos de treinamento.

Para simplificar, foi omitido de E o próprio conjunto de treinamento, além de que será considerado apenas a unidade linear, ou seja, a primeira parte do *Perceptron* antes da função de ativação, com a saída o dada pela Equação 2.7.

$$o(\vec{x}) = \vec{w} \cdot \vec{x} \tag{2.7}$$

Para encontrar o gradiente de E com respeito ao vetor de pesos $\vec{w}$, denotado por $\nabla E(\vec{w})$, temos de calcular a sua derivada, como na Equação 2.8.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n} \right] \tag{2.8}$$

Assim, a regra de treinamento para o gradiente descendente é dada pela Equação 2.9.

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w} \tag{2.9}$$

onde

$$\Delta \vec{w} = -\eta \nabla E(\vec{w}) \tag{2.10}$$

sendo η a taxa de aprendizado *(learning rate)*, e o sinal negativo para indicar a direção, que é para minimizar E. Na sua forma decomposta por componente, temos que

$$w_i \leftarrow w_i + \Delta w_i \tag{2.11}$$

onde

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \tag{2.12}$$

a fim de alterar cada componente w_i de $\vec{w}$ seguindo a proporção de $\frac{\partial E}{\partial w_i}$. Substituindo E pela Equação 2.6, temos como na Equação 2.13

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id}) \tag{2.13}$$

onde x_{id} representa o valor de entrada do componente x_i para o exemplo de treinamento d.

Finalmente, substituindo a Equação 2.13 na Equação 2.12 chegamos à regra de atualização pelo Gradiente Descendente (GD), dada pela Equação 2.14.

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id} \tag{2.14}$$

Em resumo, o algoritmo GD para treinamento de unidades lineares segue a seguintes etapas:

1. Inicie o vetor de pesos w com valores aleatórios
2. Aplique a unidade linear para todos os exemplos de treinamento
3. Calcule Δw_i para cada peso de acordo com a Equação 2.14
4. Atualize cada peso w_i adicionando Δw_i
5. Repita o processo

Como a superfície H possui apenas um mínimo local, o algoritmo do Gradiente Descendente irá convergir para um vetor de pesos com o erro mínimo, desde que os exemplos de treinamento sejam linearmente separáveis e a taxa de aprendizado η utilizada seja suficientemente pequena. Caso η seja muito grande, o gradiente descendente corre o risco de ultrapassar o mínimo da superfície e não encontrá-lo. Por esta razão, uma modificação comum é reduzir gradualmente o valor de η conforme os passos do método são executados.

2.4.3.1 Gradiente Descendente Estocástico

Um dos problemas práticos do GD, é a demora para convergir ao ponto mínimo, pois podem ser necessárias muitas iterações. Outro problema é a possível existência de vários mínimos locais na superfície de busca, o que pode impedir o método de encontrar o mínimo global. Para tentar amenizar estes problemas, existem várias estratégias de melhoria, sendo uma delas o Gradiente Descendente Estocástico (*Stochastic Gradient Descent - SGD*), de Robbins e Monro (1951).

Nesta proposta, ao invés de calcular o gradiente que verdadeiramente representa o comportamento da função de erro $E(\vec{w})$, através da utilização de todos os exemplos de treinamento D, utiliza-se uma abordagem aproximada. Esta aproximação consiste em atualizar os pesos w_i incrementalmente, seguindo o cálculo para cada exemplo individual. Assim, a regra de treinamento equivalente à Equação 2.14, é dada pela Equação 2.15

$$\Delta w_i = \eta(t - o)x_i \tag{2.15}$$

onde t, o e x_i são o valor correto *(target)*, o valor de saída *(output)* e o i-ésimo valor de entrada para o exemplo de treinamento em questão.

O SGD altera os pesos de acordo com a função de erro, iterando sobre cada exemplo de treinamento, ao contrário do método original que calcula a soma do erro de todos os exemplos e assim possui maior custo computacional. Por outro lado, o Gradiente Descendente pode utilizar um η maior, enquanto sua versão estocástica deve utilizar valores menores para compensar a imprecisão causada pela aproximação. Curiosamente, esta imprecisão traz benefícios permitindo ao SGD escapar de mínimos locais.

Otimizações como as feitas no SGD são essenciais para a construção de redes mais complexas. Sem métodos eficientes, redes como as Convolucionais vistas na Seção 2.4.8 seriam impraticáveis em muitos contextos. A Seção 2.4.9 apresenta um comparativo de diversos otimizadores existentes e as principais melhorias incorporadas, sendo capazes de otimizar os pesos das redes mais rapidamente.

2.4.4 Perceptron Multi-camadas (MLP)

Redes MLP *(MultiLayer Perceptron)* consistem em um conjunto de *Perceptrons* (ou neurônios) organizados em camadas de forma que as saídas das unidades de uma camada serão as entradas para os neurônios da camada posterior, criando assim um Grafo Direcionado Acíclico. A Figura 10 apresenta um MLP com três dados de entrada x_1, x_2 e x_3 que se conectam a uma camada oculta de três neurônios. Por fim, as saídas da camada oculta se conectam a um neurônio de saída, responsável por finalmente realizar a classificação.

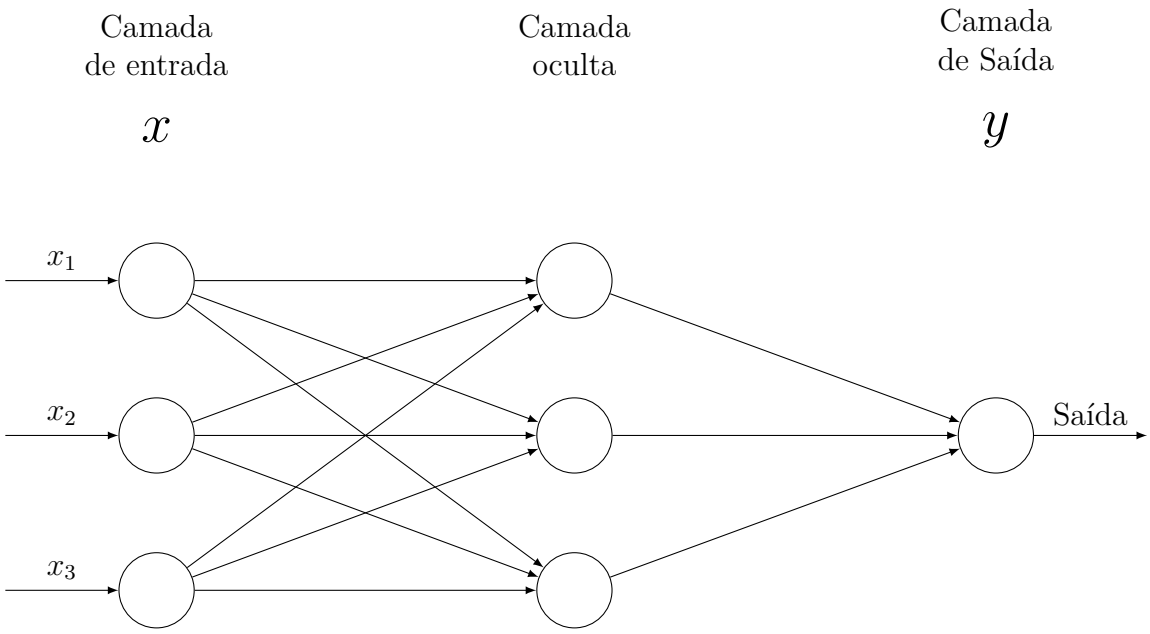


Figura 10 – Perceptron Multi-camadas (MLP)

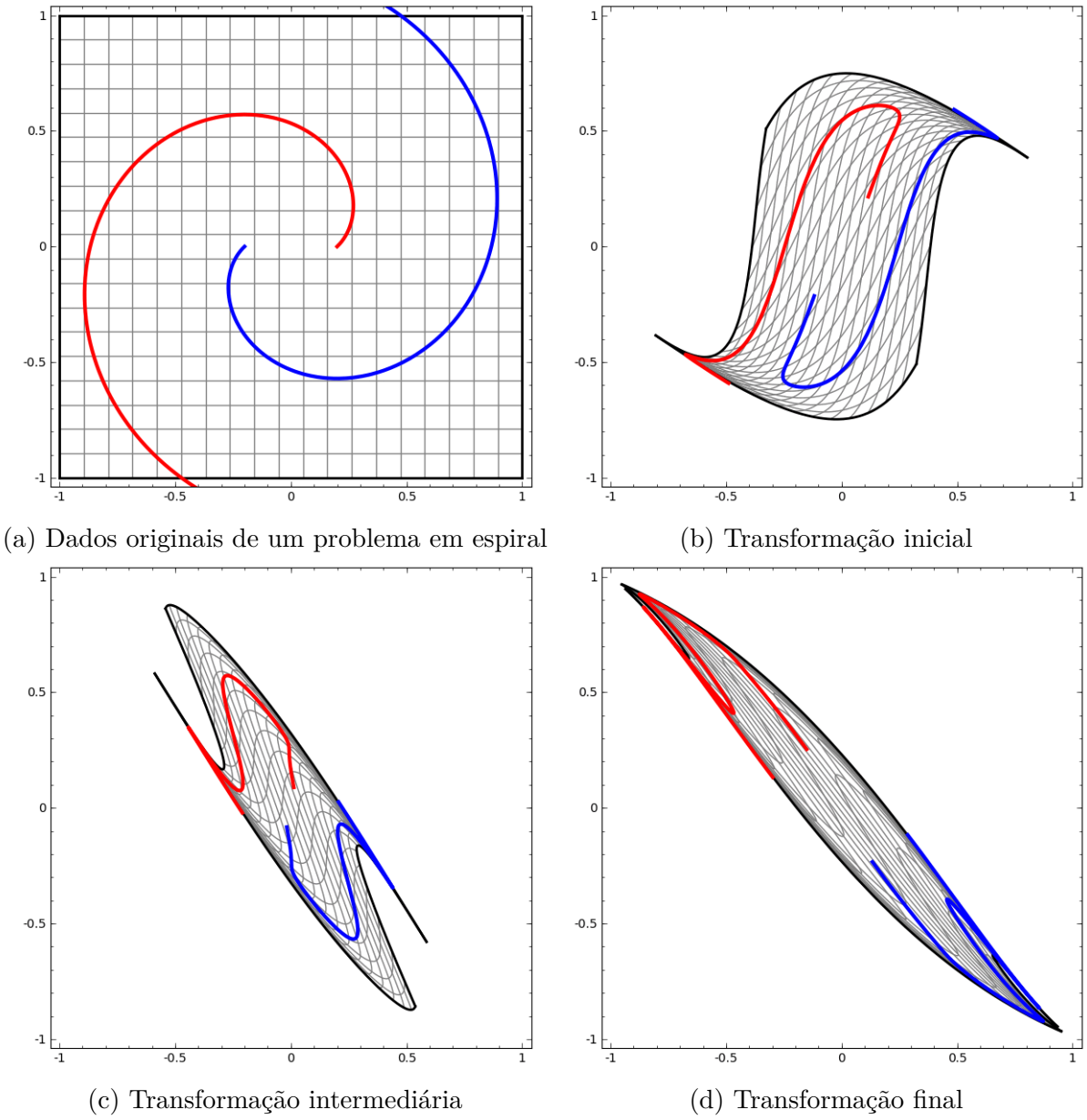
Importante ressaltar que o componente de agregação (Σ) é uma unidade linear e simplesmente cascatear unidades lineares não produzirá funções capazes de resolver problemas não-lineares. Para isso, é preciso aplicar Funções de Ativação não-lineares.

A função ϕ apresentada na Equação 2.3, por exemplo, é não-linear e por consequência permite construir uma rede capaz de resolver estes problemas mais complexos. Entretanto, como visto anteriormente na Seção 2.4.3, métodos baseados no gradiente descendente necessitam de funções deriváveis, o que não é o caso da função ϕ. A Seção 2.4.5 apresenta as principais Funções de Ativação utilizadas na prática.

Nestas condições, Hornik, Stinchcombe e White (1989) provou que uma rede MLP com pelo menos uma camada oculta e alguma função de ativação não-linear pode aproximar qualquer função em um espaço dimensionalmente finito. Todavia, essa camada pode ser inviavelmente grande e falhar em aprender e generalizar corretamente o problema investigado (GOODFELLOW, 2016).

Usualmente as RNAs, incluindo a MLP, podem ter várias camadas ocultas, ou profundas (por isso o tempo *deep*). Em cada camada, a rede transforma os dados, criando assim novas representações. A Figura 11 exibe um problema de classificação de dois espirais. Conforme a rede é ajustada, os dados são representados de maneira a facilitar a classificação, de tal forma que na saída da última camada oculta, os dados deverão estar transformados em um problema linearmente separável.

A principal vantagem destas transformações em relação a alguns processos de



(a) Dados originais de um problema em espiral

(b) Transformação inicial

(c) Transformação intermediária

(d) Transformação final

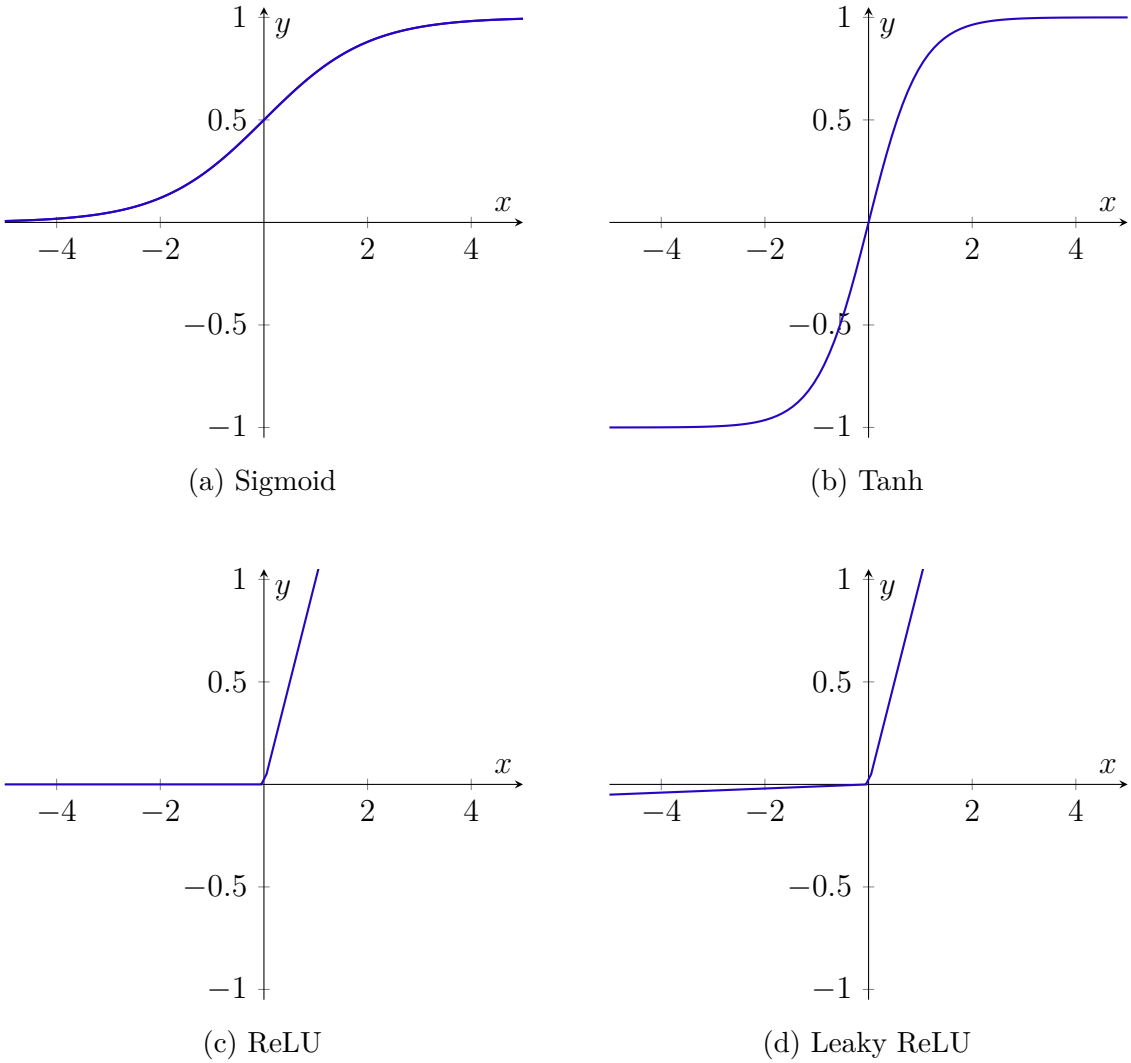
Figura 11 – Transformações de dados tornando-os linearmente separáveis.
Fonte: (OLAH, 2014)

extração de características mais tradicionais, é que os parâmetros podem ser aprendidos automaticamente a partir dos dados, tornando a tarefa menos dependente da intervenção humana. Em contrapartida, é muito mais difícil interpretar os critérios utilizados pela rede na classificação.

2.4.5 Funções de Ativação

Diferentes funções de ativação são utilizadas em diversas arquiteturas de RNAs, e cada uma delas possui suas vantagens e desvantagens. Embora algumas características possam ser conhecidas previamente, como o impacto das suas funções derivadas durante o processo de otimização, é ainda um desafio não trivial escolher a função mais adequada

de acordo com o cenário, sendo muita das vezes necessários testes empíricos para cada problema. A seguir são apresentadas as principais funções de ativação, com os gráficos comparativos de cada comportamento apresentado[3] na Figura 12.



(a) Sigmoid (b) Tanh

(c) ReLU (d) Leaky ReLU

Figura 12 – Comparativo das principais Funções de Ativação

2.4.5.1 Sigmoid

A função Sigmoid, também conhecida como função logística, delimita sua entrada ao intervalo $(0; 1)$. É uma função relativamente simples e possui as propriedades desejadas: é não-linear, continuamente diferenciável, monótona (não-decrescente), e sua saída está em um intervalo de saída fixo. Por outro lado, valores de entrada muito grandes ou muito pequenos tendem a gerar variações quase nulas na saída, o que causa o problema do *vanishing* do gradiente. Além disso, a saída não é centralizada no zero, o que dificulta o processo de otimização. Ela é vista na Figura 12a, e é dada pela Equação 2.16.

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.16}$$

[3] Adaptado de http://null.zbr.pt/?p=2259

2.4.5.2 Tanh

A função Tanh delimita sua entrada ao intervalo $(-1; 1)$ e é semelhante à Sigmoid. Entretanto, ela possui a vantagem de ser centralizada no zero e pelo fato de sua derivada ter valores mais significativos em comparação à derivada da função Sigmoid, a otimização tende a ser mais rápida. Ainda assim, o *vanishing* do gradiente é um problema que ocorre com este tipo de função. Seu comportamento gráfico é visto na Figura 12b, e sua definição é dada pela Equação 2.17.

$$\text{Tanh}(x) = \frac{e^{-x} - e^{x}}{e^{-x} + e^{x}} \tag{2.17}$$

2.4.5.3 ReLU

A Unidade Linear Retificada *(Rectified Linear Unit)*, ou apenas ReLU, é semelhante à função linear, onde $f(x) = x$, porém ela delimita sua entrada ao intervalo $[0; +\infty)$. Isso evita o *vanishing* do gradiente, que é quando o módulo do vetor gradiente decai a valores tendendo a 0. Isso pode ocorrer quando se utiliza as funções Sigmoid e Tanh, pois elas limitam os valores a intervalos pequenos, e o módulo do vetor gradiente fica "fraco"principalmente nas camadas mais profundas. Além disso, ela trás a vantagem de ser computacionalmente mais barata que a Sigmoid e Tanh. Porém, esta função deve ser utilizada apenas nas camadas internas da RNA. Outra questão importante é que se houverem ativações com valores negativos, a ReLU produzirá um gradiente igual a 0, impossibilitando que estes neurônios sejam ajustados, e assim os "matando", o que é conhecido como *problema da ReLU*. Além disso, como a imagem está no intervalo $[0; +\infty)$, os valores de ativação podem ser extremamente altos causando o que é conhecido como *explosão* do gradiente. A sua visão gráfica está ilustrada na Figura 12c, e ela é dada pela Equação 2.18.

$$\text{ReLU}(x) = \max(0; x) \tag{2.18}$$

2.4.5.4 Leaky ReLU

A Leaky ReLU é uma variante da função ReLU, onde ao invés de assumir 0 para $x < 0$, permite-se um pequeno valor não-nulo dado por αx, onde α será um gradiente constante (normalmente $\alpha = 0.01$). Desta forma, a Leaky ReLU delimita sua entrada ao intervalo $(-\infty; +\infty)$ e resolve o *problema da ReLU*. Entretanto, como possui forte linearidade, não se sai muito bem em problemas de classificação muito complexos, apresentando desempenho inferior até mesmo em relação à Sigmoid e Tanh e os seus benefícios ainda não são muito claros (CS231N, 2017). O gráfico desta função é visto em Figura 12d e é dada pela Equação 2.19.

$$\text{LeakyReLU}(x) = \max(\alpha x; x) \tag{2.19}$$

2.4.5.5 Softmax

A Softmax é uma função particular de ativação, baseada na Sigmoid, utilizada para calcular a distribuição de probabilidades em relação a n eventos. Ela transforma as saídas para cada classe em valores entre 0 e 1 e os divide pela soma das saídas. Isso essencialmente dá a probabilidade de a entrada estar em uma determinada classe. Pode ser definido como na Equação 2.20, onde j é uma classe qualquer e K é a quantidade total de classes do problema.

$$\text{Softmax}(x)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.20}$$

Exemplo Digamos, por exemplo, que temos as saídas como $[1.2, 0.9, 0.75]$. Quando aplicamos a função Softmax, obteríamos $[0.42, 0.31, 0.27]$. Então, agora podemos usá-los como probabilidades de que o valor seja de cada classe.

A função Softmax é idealmente usada na camada de saída do classificador, onde realmente estamos tentando gerar as probabilidades para definir a classe de cada entrada (DSA, 2019).

Qual função de ativação usar?

A função de ativação adequada dependerá do problema a ser resolvido e é possível que para uma mesma arquitetura seja necessário utilizar mais de uma função em diferentes partes da RNA. Na maioria das vezes, a função ReLU funcionará como um aproximador geral, por isso se a natureza do problema a ser aprendido é desconhecida, então é sugerido que se inicie utilizando ativações ReLU (SHARMA, 2017). Além disso, na prática a ReLU permite uma convergência na otimização até 6 vezes mais rápida se comparada à Tanh (KRIZHEVSKY, 2012).

2.4.6 Backpropagation

O *Backpropagation* é um algoritmo capaz de aprender os pesos de uma RNA multi-camadas através da retro-propagação dos erros. Embora concebido por vários pesquisadores anteriormente, como Parker (1985), LeCun (1986) e até trabalhos mais remotos como (WERBOS, 1974), a sua ampla compreensão foi atingida a partir de (RUMELHART, 1986).

Considerando que a rede pode conter mais de uma saída, a função de custo E será redefinida da Equação 2.6 para a Equação 2.21.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in C} (t_{kd} - o_{kd})^2 \tag{2.21}$$

onde C é o conjunto de classes de saídas da rede, t_{kd} são os valores corretos *(target)* e o_{kd} os valores de saída *(output)* para a k-ésima classe de saída e exemplo de treinamento d.

O algoritmo busca em um grande espaço de hipóteses definido por todos os possíveis valores de peso em todas as unidades da rede. Assim como para uma única unidade, como visto na Seção 2.4.3, o Gradiente Descendente pode ser utilizado na busca da hipótese que minimize o valor da função E. As redes Multi-camadas como as MLPs permitem a existência de múltiplos mínimos locais em sua superfície de hipóteses. Infelizmente, o GD convergirá para algum mínimo local, não necessariamente o erro mínimo global.

O Algoritmo 1 descreve em detalhes o funcionamento do Backpropagation. Esta é uma versão incremental (ou estocástica), baseada no GD. Nela, a entrada de uma unidade i na unidade j é denotada por x_{ji}, e o peso da unidade i para a unidade j é denotada w_{ji}. Já δ_n indica o erro associado com a unidade n.

O erro δ_k é calculado para cada unidade de saída k por $(t_k - o_k)$, multiplicado pelo fator $o_k(1 - o_k)$, que é a derivada da função de ativação Sigmoid. Já δ_h não possui t_k *(targets)* diretamente disponíveis para indicar o erro da camada oculta. Assim, os erros das unidades h são calculados pela soma dos erros de cada unidade de saída influenciada por h, ponderando-se cada δ_k por w_{kh}, ou seja, o peso da unidade oculta h para a saída k.

Uma série de condições de parada podem ser usadas, como por exemplo um número fixo de iterações, ou até atingir determinada taxa de erro no conjunto de exemplos de treinamento, ou ainda até atingir determinada taxa de erro em um conjunto de validação, à parte do conjunto de treinamento. Esta escolha é particularmente importante pois, se parada antecipadamente, a otimização pode não atingir níveis suficientemente baixos de erro, ou se parada tardiamente, a rede poderá ficar sobre-ajustada aos dados e não classificar bem exemplos desconhecidos. Estes problemas são conhecidos respectivamente como *underfitting* e *overfitting* (MITCHELL, 1997). O problema do *overfitting* será abordado em mais detalhes na Seção 2.4.10.

2.4.7 Funções de Custo

Como apresentado nas Seções 2.4.3 e 2.4.6, a escolha da função de custo (também conhecida como função objetivo ou função de perda) é fundamental para que o processo de otimização convirja adequadamente. Ela deve ser continuamente diferenciável e a função mais apropriada pode variar de acordo com o problema em questão.

Frameworks como o PyTorch (PASZKE, 2017) já fornecem muitas funções padrão e devidamente documentadas[4], além de permitir aos usuários a implementação de funções personalizadas. A seguir, são indicadas as principais delas:

[4] https://pytorch.org/docs/stable/nn.html

Algoritmo 1: Backpropagation

Data:

- *Exemplos_de_Treinamento*: Conjunto de pares $\langle \vec{x}, \vec{t} \rangle$, onde $\vec{x}$ é o vetor com os valores de entrada, e $\vec{t}$ é o vetor de saída com os valores de objetivo *(target)*
- η: é a taxa de aprendizado, ou seja, o tamanho do passo no espaço de busca
- $n_{entrada}$: o número de unidades de entrada da rede
- n_{saida}: número de unidades na camada oculta
- n_{oculta}: número de unidades de saída

Criar uma rede MLP com $n_{entrada}$ entradas, n_{oculta} unidades ocultas e n_{saida} saídas;
Inicializar todos os pesos com um número aleatório pequeno;
while *Condição de parada não for satisfeita* **do**
  foreach $\langle \vec{x}, \vec{t} \rangle$ *em Exemplos_de_Treinamento* **do**

    Propagar a entrada através da rede, computando as saídas:

    1. Entrar com $\vec{x}$ na rede e computar a saída o_u de cada unidade u;

    Retro-propagar os erros através da rede:

    2. Para cada unidade de saída k da rede, calcular seu erro δ_k

$$\delta_k \leftarrow o_k(1 - o_h)(t_k - o_k)$$

    3. Para cada unidade oculta h, calcular seu erro δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in saidas} w_{kh}\delta_k$$

    4. Atualizar cada peso w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

    onde

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

  end
end

- ***NLLLoss***: *Negative Log Likelihood Loss*, apropriada para problemas de classificação, particularmente útil em bases desbalanceadas, analisa o *Log de Probabilidades* de cada classe.
- ***CrossEntropyLoss***: Entropia Cruzada, é semelhante à *NLLLoss*, mas sem a necessidade de calcular o *Log de Probabilidades* previamente. É detalhada a seguir na Seção 2.4.7.1.
- ***L1Loss***: *Mean Absolute Error*, mede o erro médio absoluto entre a saída da rede e

o valor objetivo *(target)*. Adequada para problemas de regressão.

- ***MSELoss***: *Mean Squared Error*, semelhante à *L1Loss*, porém mede o erro médio quadrático. Também adequada para problemas de regressão.
- ***HingeEmbeddingLoss***: Usada para mensurar a similaridade entre dois tensores. Tipicamente utilizada no Aprendizado Semi-Supervisionado.

2.4.7.1 Entropia Cruzada

Como o problema investigado nos experimentos deste trabalho trata-se de uma classificação binária, a função de perda escolhida foi a Entropia Cruzada. Em sua versão para classificação binária, ela pode ser calculada por 2.22.

$$-\sum_{d \in D} t_d \log(o_d) + (1 - t_d) \log(1 - o_d) \tag{2.22}$$

onde o é a saída da rede e t é o valor objetivo *(target)*.

Para problemas multi-classes, pode-se utilizar a variação[5] 2.23.

$$-\sum_{d \in D} \sum_{k \in C} t_{kd} \log(o_{kd}) \tag{2.23}$$

2.4.8 Redes Neurais Convolucionais

As redes MLPs permitiram a construção de várias soluções de qualidade para diversos problemas. Assim, estas tecnologias foram sendo empregadas ao longo do tempo em diversos domínios de aplicação, sendo em muitos casos o núcleo responsável pelo reconhecimento de padrões. Entretanto, esta arquitetura de rede se mostrou muito sensível a pequenas mudanças no dados.

Especificamente nos problemas de Processamento de Imagens, deslocamentos de posição, distorções, dentre outras, afetavam muito os resultados dos modelos. Fukushima (1980) então propôs o *Neocognitron*, uma rede de células organizadas hierarquicamente que permite a identificação de padrões mais complexos. Esta organização teve importância fundamental para as redes que surgiram posteriormente.

Na tarefa de reconhecimento de dígitos manuscritos, trabalhos como Denker, Gardner, Graf, Henderson, Howard, Hubbard, Jackel, Baird e Guyon (1989) conseguiram notável sucesso pois, apesar de resultados iniciais *"medíocres"* (sic), atingiram os melhores resultados aprimorando o pré-processamento dos dados. Foram utilizados 49 diferentes extratores de *features* (características) que geravam *Features Maps* similares aos usados por Watanabe (1985), biologicamente inspirados em estudos como Hubel e Wiesel (1962).

[5] https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

Cada *Feature Map* indica se (e onde) determinado padrão ocorre nos dados, ou no caso, na imagem. Logo, eles podem ser formulados como um problema de casamento de padrões. Estes padrões são expressados como uma convolução usando um *kernel* bastante pequeno, visto que é verificada a ocorrência em qualquer local da imagem, de acordo com o valor do *pixel* e de seus vizinhos.

2.4.8.1 Convolução

A operação de convolução pode ser interpretada como uma medida de semelhança entre dois sinais. Para isso, em cada convolução $*$ realizada sobre a imagem I, aplica-se um *kernel* K de mesma dimensão. A Figura 13[6] mostra um exemplo de convolução 2D, onde o cálculo da saída será dado pela Equação 2.24.

$$I(x,y) * K(x,y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(m,n) \cdot K(x-m, y-n) \tag{2.24}$$

onde M e N são os tamanhos do *kernel*, que é centralizado no índice (0, 0).

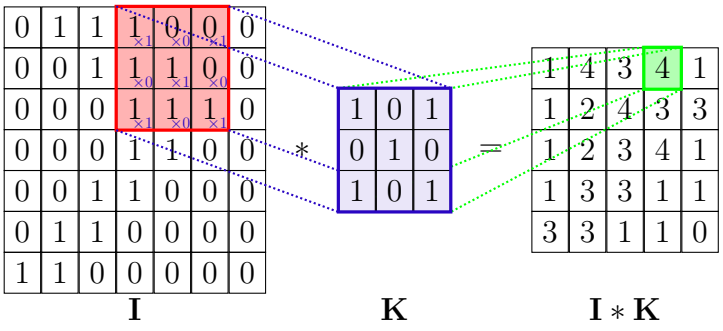


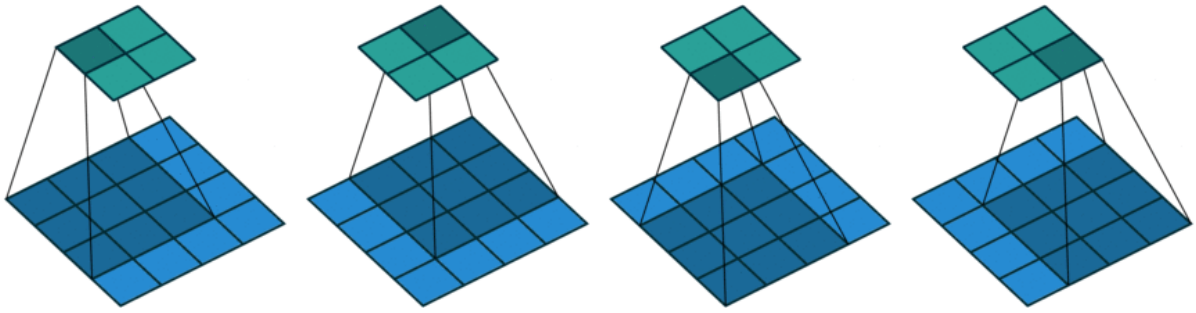
Figura 13 – Exemplo de Convolução 2D com *Kernel* 3x3

Estas operações são aplicadas em todos os componentes da entrada, criando assim Camadas Convolucionais. Estas camadas possuem três parâmetros importantes:

- F: *Field of View*, ou Campo de Visão, define o tamanho de K, dado por M e N.
- S: *Stride*, ou Tamanho do Deslocamento, define se K percorrerá a imagem deslocando de 1 em 1 pixel, 2 em 2, etc.
- P: *Padding*, ou Preenchimento, permite adicionar pixels vazios na borda de I, sendo particularmente útil para manter o tamanho original de I ao final da convolução.

Graficamente, o processo iterativo do cálculo das camadas convolucionais funciona como na Figura 14. Nela é possível visualizar uma imagem de tamanho 4x4, sendo aplicada uma convolução com *kernel* de tamanho 3x3, *stride* $= 1$ e *padding* $= 0$, gerando uma saída de tamanho 2x2.

[6] Fonte: https://github.com/PetarV-/TikZ



(a) Pixel $\langle 0,0 \rangle$ (b) Pixel $\langle 0,1 \rangle$ (c) Pixel $\langle 1,0 \rangle$ (d) Pixel $\langle 1,1 \rangle$

Figura 14 – Iterações de uma convolução 2D

Como observado, o tamanho da saída varia de acordo com o tamanho da entrada e os parâmetros da camada convolucional. É de fundamental importância considerar estas questões para construir uma arquitetura de rede válida. A resolução de saída de uma camada pode ser obtida pela Equação 2.25.

$$X_{saida} = \frac{X_{entrada} - F + 2P}{S} + 1 \tag{2.25}$$

LeCun, Boser, Denker, Henderson, Howard, Hubbard e Jackel (1989) e o relatório técnico LeCun (1989) foram trabalhos percursores das CNNs de extrema importância, pois os autores construíram RNAs com camadas convolucionais de tal forma que os valores dos *kernels* eram ajustados durante o processo de treinamento da rede. Isso permitiu reconhecer com grande sucesso dígitos manuscritos, sem grandes esforços de pré-processamento como eram necessários até então.

2.4.8.2 Pooling

As operações de *Pooling* são operações matriciais semelhantes às convoluções, porém são utilizadas em geral para agregar valores, reduzindo a variância espacial e a dimensionalidade das entradas. Desta forma, não existem parâmetros da rede a ser ajustados.

O Pooling pode ser definido baseado na função de Máximo, Mínimo, Média, dentre outras. A Figura 15[7] apresenta um exemplo de *max pooling*, com $F = 2x2$ e $S = 2$.

Estas operações, tanto o pooling quanto a convolução, compõe o núcleo das CNNs. Um material bastante completo acerca do tema foi apresentado por Dumoulin e Visin (2016) focando na aritmética convolucional aplicada nas RNAs.

2.4.8.3 Arquiteturas

As arquiteturas convolucionais possuem camadas que se diferem das tradicionais camadas de redes MLPs. Enquanto nas MLPs cada neurônio está conectado a todos

[7] Fonte: https://github.com/MartinThoma/LaTeX-examples

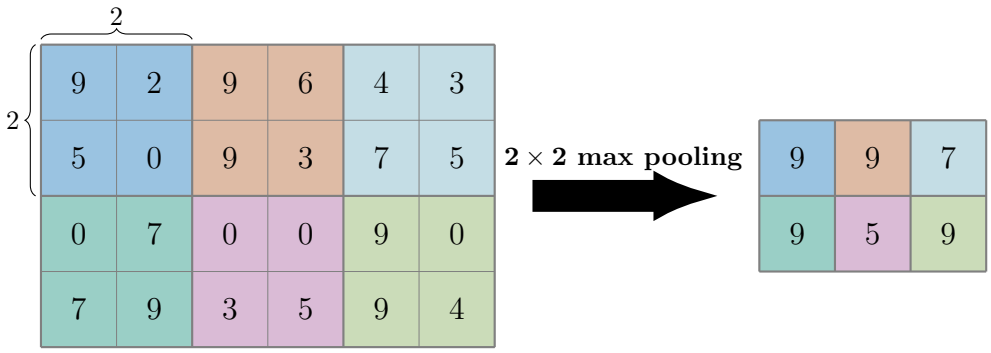


Figura 15 – Exemplo de Max-pooling

os neurônios das camadas anterior e posterior, as Redes Neurais Convolucionais (CNN) possuem algumas camadas específicas. Estas camadas são multidimensionais, geralmente em 2D e 3D.

O já citado LeCun, Boser, Denker, Henderson, Howard, Hubbard e Jackel (1989), além de descrever a implementação do processo de otimização, propôs uma arquitetura chamada *LeNet*, semelhante à vista na Figura 16. Esta arquitetura possui duas camadas convolucionais intercaladas com *Pooling* antes de seguir para uma sub-rede de camadas *fully-connected* (completamente conectadas), também chamadas de camadas lineares, ou simplesmente MLP, apresentado na Seção 2.4.4.

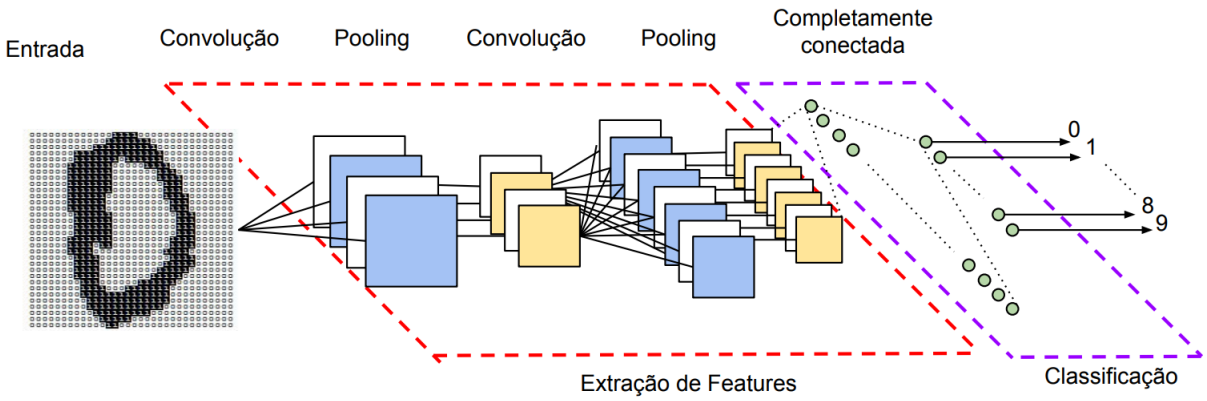


Figura 16 – Exemplo de arquitetura de uma Rede Neural Convolucional
Fonte: (VARGAS, 2016)

Os autores da *LeNet* argumentam que o princípio básico do seu design foi reduzir o número de parâmetros livres ajustáveis pelo *backpropagation*, mas sem reduzir muito o seu poder de representação. Diversos trabalhos afirmam que esta estratégia permite melhor generalização porque resulta em uma arquitetura de rede com menor entropia, além de reduzir a dimensionalidade de Vapnik-Chervonenkis (BAUM; HAUSSLER, 1989).

A arquitetura da *LeNet* pode ser interpretada em dois estágios. No primeiro, onde estão as camadas convolucionais, a rede realiza a extração de *features* (características)

ajustando os parâmetros dos *kernels* para gerar os *features maps*. Em seguida, uma rede MLP tradicional realiza a classificação daquela entrada.

Desde então as CNNs foram objeto de estudo em alguns trabalhos, principalmente com a evolução do poder computacional e o surgimento de *frameworks* com implementações cada vez mais eficientes. Mas foi a partir dos anos 2010 que seus desenvolvimentos se intensificaram.

Um fator que contribuiu para esse desenvolvimento foi a competição ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (RUSSAKOVSKY, 2015) que reunia bases com milhões de imagens e 1000 categorias.

Na edição de 2012 da ILSVRC, surgiu a AlexNet (KRIZHEVSKY, 2012), uma importante arquitetura que conquistou resultados inéditos e orientou o desenvolvimento das CNNs modernas. No ano seguinte ela foi aprimorada pela ZF Net (ZEILER; FERGUS, 2014).

Em 2014 surgiu a GoogLeNet, introduzindo o módulo *Inception*, que concatena filtros convolucionais e de pooling e reduz em aproximadamente 15 vezes o número de parâmetros da rede em relação à AlexNet (SZEGEDY, 2015). A Figura 17 ilustra o funcionamento do módulo *Inception*.

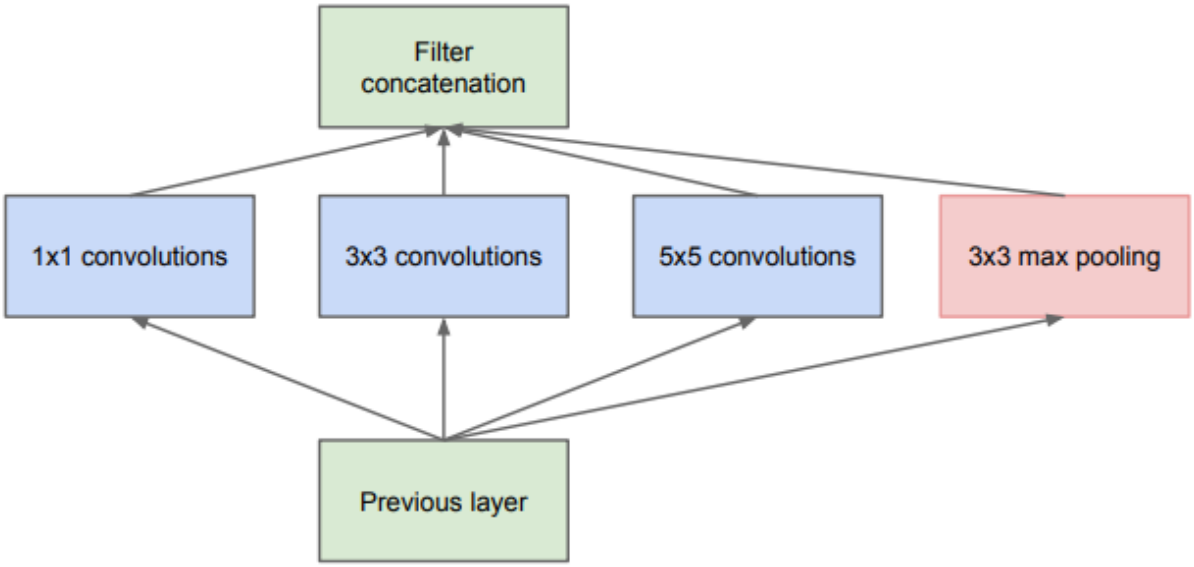


Figura 17 – Módulo Inception
Fonte: (SZEGEDY, 2015)

Além da GoogLeNet, ainda em 2014 surgiu a VGGNet que mostrou que o número de camadas ocultas é um fator crítico para uma boa performance (SIMONYAN; ZISSERMAN, 2014). Já em 2015 surgiram as ResNets (HE, 2016) que inserem conexões de atalho entre as camadas, permitindo ao gradiente atravessá-las e amenizando o problema do *vanishing* do gradiente. O ideia posteriormente foi então generalizada pela DenseNet (HUANG, 2017).

2.4.9 Otimizadores

Além dos já mencionados métodos Gradiente Descendente (Seção 2.4.3) e Gradiente Descendente Estocástico (Seção 2.4.3.1), existem várias outras propostas de otimizadores baseados no gradiente. Ruder (2016) apresenta uma visão geral dos principais algoritmos existentes, destacando as vantagens e desvantagens de cada um.

Gradiente Descente (GD)

Garante a convergência para o mínimo global em superfícies de erro convexas e converge para algum mínimo local para superfícies não-convexas. Em contra partida, é muito lento, se comparado com métodos mais modernos pois, a cada iteração, ele revê todas as instâncias do conjunto de treinamento, consumindo muita memória e se tornando impraticável para grandes volumes de dados. Essas características impedem o aprendizado *online*.

Gradiente Descendente Estocástico (SGD)

O SGD computa as atualizações a cada instância, permitindo que as iterações ocorram muito mais rápido se comparado ao GD tradicional. Desta forma, permite o aprendizado *online*. A desvantagem é que, por não seguir o *gradiente verdadeiro*, produz alta variância nas suas atualizações.

Gradiente Descente Mini-batch

Introduz o conceito de *Batches*, que consistem em subconjuntos do conjunto de treinamento completo. Realiza as atualizações analisando cada *Mini-batch* de n exemplos. Permite reduzir a variância nas atualizações em relação ao *SGD* e explorar a natureza de operações de multiplicação de matrizes aumentando o *speedup* geral. Se $n = 1$, então tem-se o SGD e se n é igual ao número de instâncias de treinamento total, tem-se o GD tradicional. Por consequência, o valor de n acaba sendo um hiper-parâmetro adicional.

Momentum

Proposto por Qian (1999), este otimizador é inspirado no princípio da física chamado *Momentum* (momento de inércia, impulso ou ainda, força cinética) que impulsiona a descida do gradiente através de um fator γ, usualmente $\gamma = 0.9$. Este comportamento permite potencializar as atualizações na mesma direção e reduzir as atualizações que a desviam, acelerando a convergência e permitindo escapar de mínimos locais. Goh (2017) apresenta um ótimo material interativo para a compreensão deste princípio.

AdaGrad (Adaptive Gradient)

Enquanto todos os métodos anteriores definem a taxa de aprendizado η igual para todos os parâmetros da rede, Duchi, Hazan e Singer (2011) propõe que η seja adaptativo, baseado na raiz quadrada da soma dos quadrados dos gradientes de iterações anteriores. Assim, ele diminui a criticidade da escolha de η e se sai bem em dados esparsos. Entretanto, em alguns casos ele pode reduzir muito o valor de *eta*, o que o torna infinitesimalmente pequeno e, na prática, não conseguirá convergir.

Adadelta e RMSprop

Para corrigir o problema do AdaGrad que armazena todo o histórico dos gradientes, Zeiler (2012) propõe o Adadelta, que define uma janela de restrição. Com esta janela, o método calcula a média acumulada do quadrado dos últimos gradientes e ainda adiciona um parâmetro γ semelhante ao termo do *Momentum*. Independentemente, Hinton, Srivastava e Swersky (2012) desenvolveram um método bastante similar, nomeado de RMSprop.

2.4.9.1 Adam

Adaptive Moment Estimation (Adam) (KINGMA; BA, 2014) é mais um método de otimização baseado em gradiente, evoluído a partir dos otimizadores citados anteriormente. De forma semelhante ao Adadelta e ao RMSprop, o Adam mantém a média dos gradientes anteriores m_t, além da média do quadrado dos gradientes anteriores, dada por v_t, como visto em 2.26 e 2.27, respectivamente.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.26}$$

e

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.27}$$

onde m_t e v_t são iniciados com vetores nulos, criando um viés próximo a 0 nos passos iniciais e especialmente quando a taxa de decaimento, calculada por 2.28 e 2.29, diminui (ou seja, β_1 e β_2 estão próximos a 1).

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.28}$$

e

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.29}$$

Os autores sugerem os valores 0.9 para β_1 e 0.999 para β_2, gerando em ambos os casos um decaimento exponencial. Assim, a regra de atualização do Adam é dada por 2.30.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{2.30}$$

sugerindo que $\eta = 0.001$ e $\epsilon = 10^{-8}$. Kingma e Ba (2014) verificou empiricamente que o desempenho do Adam é ligeiramente superior aos outros algoritmos de otimização citados.

Questões práticas

Diante de tantas opções de otimizadores, mesmo que os seus desempenhos sejam equivalentes, é necessário escolher um deles. Embora existam outros algoritmos além dos citados aqui, e novas propostas estejam surgindo com relativa regularidade, em geral o Adam pode ser a melhor escolha (RUDER, 2016). Por esta razão, foi o otimizador escolhido para a realização dos nossos experimentos.

Uma característica desejável (e em alguns casos indispensável) de qualquer otimizador é a capacidade de ser executado em paralelo e, em alguns casos, de forma distribuída. Os principais *frameworks* como o TensorFlow (ABADI, 2015) e o Pytorch (PASZKE, 2017) fornecem implementações eficientes que exploram bem estes paradigmas de programação.

Outro fator de interesse é a possibilidade de se executar estes métodos em Unidades de Processamento Gráfico - GPUs. Visto que a maioria dos cálculos envolvidos nestes algoritmos são operações básicas de adição e multiplicação de matrizes, este tipo de *hardware* consegue executar grande volume de computação paralela rapidamente, se comparado aos processadores de uso geral (CPUs). Esta alternativa permitiu a redução no tempo de espera para o treinamento de várias arquiteturas, tornando estas tecnologias muito mais acessíveis.

Como já mencionado, a taxa de aprendizado η deve ser suficientemente pequena pois, caso ela seja demasiadamente alta, a otimização pode divergir do objetivo. Assim, usualmente define-se um valor pequeno para garantir a convergência, mesmo que isso exija um número maior de iterações.

Geralmente são necessárias várias iterações sobre os mesmos dados a fim de convergir corretamente a rede. Neste contexto, criou-se o conceito de épocas *(epoches)*. Cada época consiste de um ciclo de iteração sobre todas as instâncias de treinamento. Entretanto, um número excessivo de épocas pode causar problemas como visto a seguir na Seção 2.4.10.

2.4.10 Regularização

Como citado na Seção 2.4.4, as RNAs possuem um poder de representação muito grande, o que é essencial em inúmeros problemas. Todavia, como discutido na Seção 2.4.6, um dos maiores desafios dessa tecnologia é evitar o sobre-ajuste aos dados, ou seja, o *overfitting*. Uma rede com *overfitting* pode atingir erros mínimos para o conjunto de treinamento, entretanto, por não conseguir generalizar o problema, não se sai bem com dados desconhecidos, tendo pouca utilidade prática.

A fim de evitar a ocorrência deste fenômeno, existem técnicas conhecidas como Regularização (GOODFELLOW, 2016). Embora o *overfitting* não seja um problema exclusivo das RNAs, são apresentadas algumas estratégias utilizadas especificamente neste contexto. Além de serem técnicas relevantes e de reconhecido sucesso, todas foram

utilizadas nos experimentos aqui realizados, sendo elas a *Early Stopping*, Regularização *L2*, *Dropout* e *Batch Normalization*.

2.4.10.1 Early Stopping

Uma das maneiras mais simples de evitar o *overfitting* é o *Early Stopping* (Antecipar a Parada), sendo uma estratégia de regularização implícita (LIN, 2016). O *Early Stopping* consiste em isolar um conjunto de validação com exemplos de dados não presentes no conjunto de treino.

Assim, durante o otimização da rede, deve-se calcular o erro obtido nesse conjunto de validação, desconhecido do otimizador. Caso o erro no conjunto de validação esteja crescendo consideravelmente, mesmo com a sua diminuição no conjunto de treino, é um forte indício de que a rede está sobre-ajustando e deve-se interromper o processo.

Prechelt (1998) discute sobre os problemas práticos da *Early Stopping*, visto que ocasionalmente o erro da validação pode crescer em determinado momento, mas se reduzir em iterações posteriores. Na prática, o *Early Stopping* baseado no número de épocas, por exemplo, monitorando o erro do conjunto de validação, é uma estratégia frequentemente usada (YAO, 2007).

2.4.10.2 Regularização L2

A Regularização L2 é um método estatístico que adiciona um termo de regularização à Função de Custo, com o objetivo de mensurar e minimizar a complexidade do modelo. É uma técnica aplicável a outras classes de algoritmos (além das RNAs) e formalizado como um problema de otimização.

Roughgarden e Valiant (2016) argumenta que a regularização L2 é um método prático de implementação do conceito da *Navalha de Occam* que diz ser preferível modelos mais simples pois eles capturam melhor os conceitos fundamentais do problema. Para isso, a Função de Custo é reescrita por 2.31.

$$E'(\vec{w}) = E(\vec{w}) + \lambda||\vec{w}||^2 \tag{2.31}$$

onde $||\vec{w}||$ é a distância euclidiana dos componentes w_i e λ é uma constante que controla a intensidade da regularização.

Como resultado, os pesos $\vec{w}$ tendem a 0, gerando modelos mais simples, como visto na Figura 18.

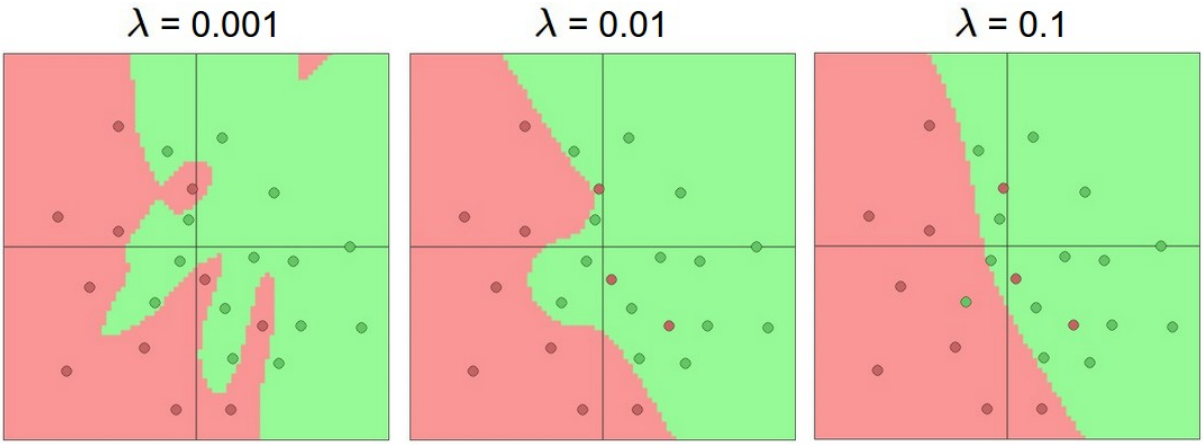


Figura 18 – Regularização L2
Fonte: (KARPATHY, 2016)

2.4.10.3 Dropout

A ideia básica do *Dropout*, apresentada em Hinton, Srivastava, Krizhevsky, Sutskever e Salakhutdinov (2012), é que cada neurônio tem uma probabilidade p (geralmente $p = 50\%$) de estar ativo. Se um neurônio está inativo, então ele não é atualizado naquela iteração. Esta técnica é utilizada apenas durante a etapa de treinamento, reativando todos os neurônios para a validação ou teste do modelo.

Graficamente, o *Dropout* funciona como visto na Figura 19. A cada iteração, sua aplicação resulta na desativação de alguns neurônios (Figura 19b), obrigando a rede a ajustar seus pesos de forma mais diluída (SRIVASTAVA, 2013), sem supervalorizar alguns neurônios. Isso causa um efeito de generalização, reduzindo as chances de *overfitting*. De modo geral, aplicar o *Dropout* é equivalente a tirar a média de um número exponencial de arquiteturas rapidamente.

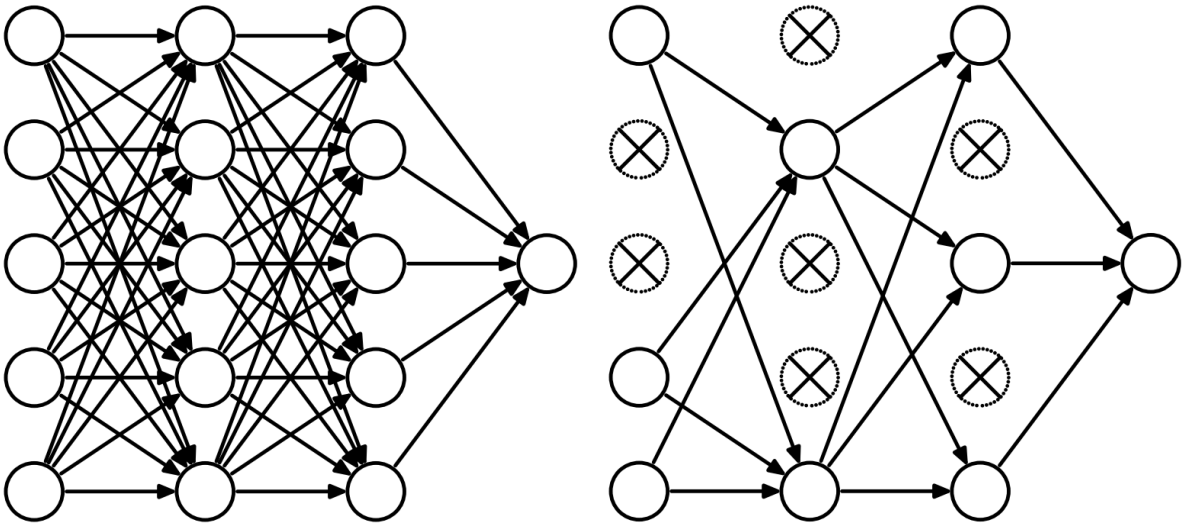
2.4.10.4 Normalização

Dados desnormalizados tendem a dificultar o treinamento das redes, podendo enviesá-las para as variáveis com maior ordem de grandeza. Se duas entradas, x_1 e x_2, possuem domínios em intervalos muito discrepantes, por exemplo, $x_1 \in [-2, 2]$ e $x_2 \in [0, 10000]$, a superfície de otimização tende a exigir atualizações (ou passos) maiores para determinados pesos do que para outros. Isso dificulta o processo de otimização e, para acelerar e permitir a convergência, usualmente aplica-se uma técnica de normalização.

A primeira e mais simples busca mapear os dados a um determinado intervalo utilizando-se os valores máximos e mínimos, definida em 2.32.

$$x_{norm} = \frac{x - x_{max}}{x_{max} - x_{min}} \tag{2.32}$$

Apropriada para distribuições normais, outra técnica de normalização conhecida



(a) Rede neural padrão (b) Após aplicação do Dropout

Figura 19 – Exemplo de Dropout em uma Rede Neural
Fonte: Adaptado de (SRIVASTAVA, 2014)

como padronização, utiliza a fórmula *z-score*, dada por 2.33.

$$z = \frac{x - \mu}{\sigma} \tag{2.33}$$

onde μ é a média de todos os valores de x e σ é o seu desvio padrão. Desta forma, os dados são centralizados em 0 e seu desvio padrão é igualado a 1.

Estas técnicas de transformação já permitem um melhor desempenho da maioria dos métodos de aprendizado de máquina. No contexto das RNAs, os componentes w_i dos vetores de pesos $\vec{w}$ também podem produzir valores discrepantes e causar problemas no processo de otimização. Com o objetivo de realizar normalizações durante o treinamento da rede e reduzir o que os autores chamam de *internal covariate shift*, surgiu o *Batch Normalization*.

Batch Normalization Para calcular o *z-score*, é necessário primeiramente o cálculo da média e o desvio padrão, o que exige a utilização de todos os dados . Entretanto, como visto nas Seções 2.4.3.1 e 2.4.9, na prática utiliza-se otimizadores baseados em *mini-batches*. Desta forma, o *Batch Normalization* estima a média e o desvio padrão a cada *batch* (IOFFE; SZEGEDY, 2015).

Seja uma camada j de d dimensões tal que $x_j = (x_{1j}, ..., x_{dj})$. De acordo com 2.33, sua normalização seria dada por 2.34.

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_{ij}}{\sigma_{ij}} \tag{2.34}$$

Para restaurar o poder de representação da rede, foram adicionados parâmetros γ_j e β_j, resultando em 2.35.

$$y_{ij} = \gamma_j \hat{x}_{ij} + \beta_j \tag{2.35}$$

Estes parâmetros são aprendidos juntamente com os parâmetros do próprio modelo, fazendo com que $\gamma_j = \mu_j$ e $\beta_j = \sigma_j$.

2.4.11 Considerações Adicionais

Foram apresentadas aqui algumas arquiteturas de Redes Neurais Artificiais, mas a versatilidade do método e o aumento do volume de pesquisas sobre o tema tem produzido construções cada vez mais complexas. Desta forma, outros tipos de redes ficaram de fora desta investigação, como as Redes Neurais Recorrentes (RNNs), Autoencoders, Redes Adversárias Generativas (GANs), dentre outras.

Embora os fundamentos das RNAs sejam de meados do Século XX, muitas das técnicas que impulsionaram de forma significativa a qualidade dos resultados surgiram apenas mais recentemente. Uma das explicações históricas são os períodos que ficaram conhecidos como *Inverno da IA*. Testes levavam dias para ser executados e tornavam proibitivos métodos baseados em RNAs em diversos domínios de aplicação.

O aumento do poder computacional permitiu a execução de modelos mais complexos, com camadas mais "profundas", o que fez surgir o termo *Deep Learning*. Muitos pesquisadores defendem que o aprendizado profundo se difere do aprendizado de máquina tradicional[8] à medida que aumenta-se consideravelmente o volume dos dados de treinamento, utiliza-se computação paralela, distribuída e, é claro, redes com camadas cada vez mais profundas.

Em geral, construir um modelo de *Deep Learning* envolverá a definição de uma arquitetura de rede, com suas camadas internas, suas funções de ativação e a camada de saída, de acordo com o problema em questão. É necessário definir então uma função de custo e qual será o otimizador para o processo de treinamento da rede. Por fim, deve-se construir uma rotina eficiente de carregamento dos dados *(Data Loader)* integrado ao ambiente de execução.

O surgimento de *frameworks* de código aberto que realizam o treinamento de forma paralela, especialmente em GPUs, tornou esta tecnologia mais acessível a várias pessoas. Ciresan, Meier, Masci, Gambardella e Schmidhuber (2011) relata que em alguns casos, utilizar GPUs torna o treinamento até 60 vezes mais rápido que se utilizado CPUs, permitindo a realização de treinamentos que gastariam horas em questão de minutos. Assim, embora ainda seja um desafio, a dificuldade de processar os métodos aos poucos vai dando lugar à escassez de dados rotulados.

Isso torna o *Deep Learning* "faminto"por dados. A capacidade destes métodos é tão grande que se não houverem dados suficientes a rede pode se super-ajustar às instâncias de treinamento. Por esta razão, as técnicas de regularização são tão importantes para evitar

[8] Frequentemente referenciado como *Shallow Learning* (aprendizado razo)

o *overfitting*. Além das técnicas apresentadas na Seção 2.4.10, existem outras como a *Data Augmentation*, muito útil em problemas de processamento de imagens.

Assim, ao mesmo tempo que o *Deep Learning* torna viáveis várias aplicações que eram até então impraticáveis, ainda existem vários desafios, limitações e necessidades de entendimento pelos seus usuários, como acontece em qualquer tecnologia.

3 Trabalhos Relacionados

Embora não limitado a problemas envolvendo dados financeiros, este trabalho teve como motivação inicial a criação de um modelo preditivo neste cenário. Assim, o artigo (YEH; LIEN, 2009) logo se mostrou extremamente relevante. Os autores realizaram um estudo comparativo dos principais métodos de aprendizado de máquina, para a predição da probabilidade dos clientes se tornarem futuros inadimplentes no cartão de crédito de um banco de Taiwan. A base de dados utilizada foi disponibilizada na UCI[1] e nos permitiram os primeiros experimentos envolvendo esse tipo de problema. Embora não utilizasse camadas convolucionais (ao menos não foi informado), as Redes Neurais Artificiais foi um dos métodos investigados e já neste trabalho mostrou-se o mais adequado. Além disso, foi proposto o *Sorting Smoothing Method* para avaliar a qualidade dos resultados em relação ao que chamaram de Real Probabilidade de Inadimplência Estimada.

Este tipo de problema é conhecido na literatura como *Credit Scoring*. Diversas pesquisas já abordaram o tema, como (ARAÚJO, 2007) que aplicou este tipo de modelagem em uma instituição financeira. Além do *Credit Scoring*, foi desenvolvido um modelo de *Behavioral Scoring* que não apenas busca mensurar o risco de crédito como também realiza uma análise comportamental dos clientes. Este segundo tipo de abordagem se aproxima mais do problema real investigado neste trabalho, visto que é feito uma análise preditiva da probabilidade de um cliente qualquer contratar novos créditos. Voltando ao *Credit Scoring*, algumas teses foram baseadas neste tipo de problema, como (KARCHER, 2009) que por sua vez utiliza redes bayesianas em seu estudo. A autora investiga alguns tipos de classificadores bayesianos e evidencia que modelos não-ingênuos, ou seja, que não consideram todos os atributos independentes, dada a classe (como o *Naive Bayes*), apresentam os melhores resultados. Outra tese envolvendo análise de risco é (PINHEIRO, 2005), que por sua vez faz um estudo de caso em uma operadora de telefonia, mostrando a vasta aplicabilidade deste tipo de modelo. Neste caso foram utilizadas redes neurais, na arquitetura MLP.

Todos estes trabalhos citados porém, utilizam-se de diversas variáveis categóricas descritivas (como sexo, estado civil, etc), não se baseando mais fortemente em dados numéricos históricos, como feito aqui. Ainda no contexto financeiro, um tipo de problema frequentemente abordado é o de previsão em mercados de bolsas de valores. Para isso é bastante popular o uso de Redes Neurais LSTM (Long Short-Term Memory) como realizado em (MACHADO, 2015), (TERNA, 2016) e (NELSON, 2017), já indicando o potencial dos modelos baseados em *Deep Learning* neste tipo de problema. Outras abordagens mais clássicas como em (BAO, 2008) focava bastante na engenharia de

[1] https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

atributos, buscando maneiras eficientes de representar os dados e extrair informações adicionais para posteriormente aplicar métodos bayesianos ou de árvore de decisão.

A representação e, em geral, as aplicações envolvendo séries temporais compõe uma classe de problemas desafiadores. Uma revisão bastante útil sobre o tema é feita em (FU, 2011) que apresenta uma ampla variedade de questões sob o ponto de vista de mineração de dados em séries temporais. Tarefas de classificação, descoberta de padrões, agrupamento, sumarização, dentre outras, são abordadas. Um ponto particularmente relevante aqui é o de representação de séries multi-dimensionais em que o autor cita (CHEN, 2002), dentre outros trabalhos. Nele é utilizado uma importante e já consolidada tecnologia de banco de dados, os cubos OLAP. Chen, Dong, Han, Wah e Wang (2002) argumenta que apenas uma pequena porção dos dados são necessárias para a criação do seu modelo. No entanto, vale ressaltar que neste caso o autor realizou uma regressão linear de um *streaming* de dados.

Tam (1998) faz uma importante diferenciação entre ROLAP e MOLAP, respectivamente modelos relacionais e multi-dimensionais. Ele defende que existem muitos algoritmos eficientes para tratar ROLAP, enquanto o mesmo não acontece para MOLAP e então propõe uma estratégia para seleção de atributos. Tanto (KORIKACHE; YAHIA, 2014), que apresenta uma boa revisão bibliográfica, quanto (LAMANI, 2019) também utilizam-se da premissa de reduzir os dados, criando sub-cubos para posterior aplicação de árvores de regressão e *clustering*, enquanto (JADAV; PANCHAL, 2012) tenta extrair regras de associação. Já mais voltado às aplicações de negócio, (KOHAVI; SOMMERFIELD, 1998) utiliza tabelas de decisão com o objetivo de facilitar a interpretabilidade do modelo, assim como (SAIR, 2012) também segue essa linha objetivando o surgimento de *insights* sobre os dados.

Cabe mencionar aqui a ferramenta *DBMiner* apresentada em (HAN, 1998), que dentre outras funcionalidades, permite realizar predições através de métodos estatísticos que criam rankings dos atributos mais relevantes e posteriormente aplica métodos baseados em árvores de decisão. Sob esse aspecto, (FU, 2007) considera que as árvores de decisão tradicionais são ineficientes pois criam várias partições recursivamente e então propõe um método semelhante, utilizando dados pré-computados. A partir destas e várias outras tecnologias surgem diversos sistemas de apoio à decisão como visto em (HUA, 2017) e (PALANIAPPAN; LING, 2008), que apresenta uma ferramenta de visualização e descoberta de padrões no contexto da medicina.

Finalmente chegamos aos métodos baseados em *Deep Learning*. Drakopoulos, Spyrou e Mylonas (2019) faz uma importante observação em relação à Álgebra de Tensores. Segundo os autores, Tensores e arrays multi-dimensionais estão presentes em diversos campos da Ciência da Computação sob diferentes nomes incluindo, no caso de banco de dados, os cubos OLAP. Os tensores são parte fundamental dos recentes *frameworks* de *Deep Learning* e, como muitos dados são associados em uma hierarquia OLAP, diversas

aplicações se beneficiaram dessa correspondência.

Thirumuruganathan, Ouzzani e Tang (2019) dizem que a hierarquia dos cubos OLAP permite sumarizar dados e identificar algumas características em pequenos conjuntos de "regiões". Wei, He, Chen, Zhou e Tang (2017) por exemplo, utiliza-se desta tecnologia para a criação de um sistema de recomendação. Entretanto, o que vem mais chamando a atenção na última década é sem dúvidas os resultados alcançados em visão computacional e processamento de imagens. De fato, imagens são matrizes de *pixels* e representam um bom exemplo de arrays multi-dimensionais.

As aplicações em visão computacional vão desde a atividades bem específicas como detecção de pedestres (VARGAS, 2016) por exemplo, até grandes arquiteturas de reconhecimento bem mais genético, como a apresentada em (SZEGEDY, 2015). Esta rede em particular propõe a criação de *Inseptions*, que concatena dados processados através de diferentes fluxos de convolução. Estas redes são frequentemente utilizadas em competições de classificação de imagens e são aperfeiçoadas como em (HE, 2016) e (HUANG, 2017), o que tem as tornado cada vez mais complexas e profundas.

Muito se discute sobre a razão pela qual as Redes Neurais Profundas funcionam na prática, já que são considerados métodos *caixa-preta*, e iniciativas como (THORPE; GENNIP, 2018) buscam provar matematicamente isso. Embora as camadas convolucionais sejam bastante populares, inclusive sendo utilizadas neste trabalho, existem propostas de utilizar redes LSTM (*Long short-term memory*) multi-dimensionais, como em (GRAVES, 2007). Em trabalho posterior, Graves e Schmidhuber (2009) realizaram a classificação textos arábicos manuscritos de maneira muito eficiente.

Ainda assim, as redes *fully-connected* (MLP) são utilizadas e apresentam bons resultados, como visto em (AVATI, 2018) com um modelo para prever a probabilidade de pacientes de um hospital virem a falecer nos próximos 12 meses. Este modelo apresenta 18 camadas ocultas e os autores avaliaram o modelo utilizando a área sob a curva ROC (AUC), como feito nos já citados (YEH; LIEN, 2009) e (KARCHER, 2009).

Por ser uma métrica bastante popular, alguns trabalhos investigam estratégias para otimização da AUC, como visto em (NETO, 2017). Yan, Dodier, Mozer e Wolniewicz (2003) reforça que a AUC não é uma função diferenciável e por isso não deve ser otimizada por métodos baseados em gradiente. O autor então propõe uma função de perda baseada no método *Wilcoxon-Mann-Whitney*, que é diferenciável e equivalente à AUC. Já Eban, Schain, Mackey, Gordon, Saurous e Elidan (2016) argumenta que na prática, por questões de desempenho, os modelos baseados em *ranking* são otimizados para maximizar a acurácia na esperança que a AUC também seja maximizada e propõe uma nova função de perda capaz de manter desempenho aceitável, ao mesmo tempo que melhora os resultados.

4 Trabalho Proposto

Este capítulo tem como objetivo apresentar a abordagem proposta para utilização de Redes Neurais Convolucionais em dados estruturados, a partir da organização dos mesmos como cubos OLAP. Além disso, são definidos alguns critérios para avaliação no estudo de caso realizado com uma base de dados real.

A primeira etapa do processo consiste em modelar os dados de forma a criar as dimensões e métricas a partir de determinadas características que permitam esse tipo de representação dos dados. Aplica-se então algumas operações de cubos OLAP para consolidar tais informações e a hipótese é que, assim como analistas humanos podem ter *insights* nestas novas representações da informação, métodos de aprendizado de máquina também poderão se beneficiar delas na criação do modelo.

Estas novas representações podem tornar mais evidentes alguns aspectos dos dados e diferentes técnicas serão capazes de extrair estas características. Uma das técnicas de maior sucesso no contexto de processamento de imagens são filtros baseados em Kernels Convolucionais. Geralmente utilizadas em Redes Neurais Artificiais (RNAs), estas operações conseguem extrair características ao mesmo tempo que o otimizador da rede aprende os seus padrões. Na nossa modelagem proposta, a natureza matricial dos dados se assemelha às matrizes de *pixels* que formam as imagens, geralmente de duas ou três dimensões.

As caraterísticas extraídas dos dados podem ser bastante úteis para diferentes tarefas, como por exemplo em modelos preditivos. Neste trabalho em específico, investigamos um caso de classificação binária baseada em *Ranking*. Propomos justamente uma arquitetura baseada em Redes Neurais Convolucionais (CNNs) pela capacidade delas extraírem novas características juntamente com o treinamento do classificador, através de um otimizador baseado em decida do gradiente.

Essa arquitetura foi chamada de *OlapNet* pois visa explorar diferentes visões dos dados previamente representados como um cubo OLAP. As operações devem ser feitas na própria RNA, criando diferentes fluxos de processamento dos dados. Como critério de avaliação dos resultados, foi escolhida a AUC, enquanto a validação se deu utilizando o método *Houl-Out*.

4.1 Modelagem de Dados

Uma instância qualquer é representada por um conjunto de características, chamadas de atributos. Estes atributos podem ser de diversos tipos:

- Numéricos
- Categóricos
- Períodos, Datas, etc.
- Textos
- Dados não-estruturados
- Etc.

Em alguns casos, certos atributos constituirão uma hierarquia intrínseca que permitirão a sua representação como um cubo OLAP. O processo de criação desse cubo exigirá conhecimento das regras de negócio e o entendimento de como definir suas dimensões e métricas.

Os atributos numéricos em particular possuem uma propriedade especial: podem ser naturalmente **Agregados**. Desta forma, alguns deles serão as **métricas** do cubo, para então serem realizadas as operações OLAP necessárias. Já os atributos Categóricos (ou mesmo alguns numéricos), incluindo Períodos e Datas, constituirão as **dimensões** do cubo, permitindo que elas sejam consolidadas hierarquicamente a partir das operações realizadas sob as métricas. Outros tipos de atributos como Textos e outros dados não-estruturados exigirão outras abordagens que fogem deste escopo.

Exemplo

Para ilustrar a ideia proposta, apresentamos um exemplo contendo dados hipotéticos. Considere um conjunto de empresas onde deseja-se criar um modelo para estimar o grau de satisfação dos funcionários de cada empresa, por exemplo uma regressão para o intervalo $[0, 10]$.

Para cada instância, ou seja, em cada empresa analisada, existirá um conjunto de atributos que a descreve. Um sub-conjunto destes atributos seria, por exemplo, a listagem dos funcionários, com seus respectivos dados, como apresentados em Tabela 1.

A partir dos dados de entrada apresentados na Tabela 1, podemos remover a identificação do Funcionário e outros atributos menos relevantes, como visto na Tabela 2.

Podemos agora realizar um pivotamento do atributo Sexo, transformando os dados apresentados em linhas, como colunas **F** e **M**. Os valores onde não se aplica, ficarão *nulos* e serão representados por -, como apresentado na Tabela 3.

Desta forma, teríamos duas dimensões: **Admissão** e **Sexo**. Podemos ainda consolidar os dados criando Faixas de Tempo de Admissão, agregando-os através da função MÉDIA. O resultado é apresentado na Tabela 4.

Funcionário	**Admissão**	**Sexo**	**Salário**	**...**
Funcionário 1	01/01/2007	M	$ 5.625,00	...
Funcionário 2	02/02/2012	F	$ 3.750,00	...
Funcionário 3	03/03/2013	M	$ 5.250,00	...
Funcionário 4	04/04/2014	F	$ 6.875,00	...
Funcionário 5	05/05/2015	F	$ 3.750,00	...
Funcionário 6	06/06/2017	F	$ 1.250,00	...
Funcionário 7	07/07/2017	M	$ 10.000,00	...
Funcionário 8	08/08/2018	M	$ 4.187,00	...

Tabela 1 – T1: Dados hipotéticos de funcionários de uma empresa

Admissão	**Sexo**	**Salário**
01/01/2007	M	$ 5.625,00
02/02/2012	F	$ 3.750,00
03/03/2013	M	$ 5.250,00
04/04/2014	F	$ 6.875,00
05/05/2015	F	$ 3.750,00
06/06/2017	F	$ 1.250,00
07/07/2017	M	$ 10.000,00
08/08/2018	M	$ 4.187,00

Tabela 2 – T2: Seleção dos atributos Admissão, Sexo e Salário

Admissão \ **Sexo**	**F**	**M**
01/01/2007	-	$ 5.625,00
02/02/2012	$ 3.750,00	-
03/03/2013	-	$ 5.250,00
04/04/2014	$ 6.875,00	-
05/05/2015	$ 3.750,00	-
06/06/2017	$ 1.250,00	-
07/07/2017	-	$ 10.000,00
08/08/2018	-	$ 4.187,00

Tabela 3 – T3: Pivotamento do atributo Sexo para colunas

Admissão \ **Sexo**	**F**	**M**
Acima de 7 anos	$ 3.750,00	$ 5.625,00
De 4 a 6 anos	$ 5.312,50	$ 5.250,00
Até 3 anos	$ 1.250,00	$ 7.093,75

Tabela 4 – T4: Média salarial dos funcionários de acordo com a Faixa de Tempo de Admissão e o Sexo

Admissão \ Sexo	F	M
Média	$ 3.906,25	$ 6.265,50

Tabela 5 – T5: Média salarial dos funcionários de acordo com o Sexo

Admissão \ Sexo	Média
Acima de 7 anos	$ 4.678,50
De 4 a 6 anos	$ 5.291,67
Até 3 anos	$ 5.145,67

Tabela 6 – T6: Média salarial dos funcionários de acordo com a Faixa de Tempo de Admissão

A Tabela 4 já nos trás algumas informações relevantes, mas partindo da Tabela 3 (T3) poderíamos agregar toda a dimensão de Admissão para obter a Tabela 5. Esta tabela fornece precisamente os salários médios por sexo. Analogamente, a partir de (T3) seria possível agregar a dimensão de Sexo e obter a Tabela 6 (T6) com os salários médios por Faixa de Tempo de Admissão, seguindo o critério adotado em (T4).

Note que pelas Tabelas 5 e 6 é muito mais fácil afirmar respectivamente que, em média, nesta empresa:

1. Homens ganham mais que as mulheres
2. Funcionários com maior tempo de Admissão nem sempre terão os maiores salários

É claro que estas constatações poderiam ser extraídas dos dados originais presentes na Tabela 1, como de fato o foram. Entretanto, após a série de mudanças na representação dos dados, estes *insights* ficaram muito mais claros do que em sua representação original. Porém, não se pode negar que ocorreram perdas de informação neste processo.

As Tabelas 1, 2 e 3 mostram que o maior salário dentre todos os funcionários é de $ 10.000,00, o que não pode ser conhecido baseando-se apenas nas Tabelas 4, 5 ou 6. Já as Afirmações 1 e 2 seriam impossíveis somente a partir das Tabelas 6 e 5, respectivamente. Para tentar minimizar este problema e explorar os ganhos obtidos por estas diferentes visões fornecidas pelas operações de cubos OLAP, a Seção 4.2 a seguir apresenta algumas propostas.

Note que todas estas constatações nada mais são que características geradas a partir dos dados. Elas poderão ser úteis para o processo de tomada de decisão do modelo preditivo. Neste caso foi exemplificado um modelo de regressão, citado no início do exemplo, que de posse destas novas características, poderá conseguir resultados mais aprimorados.

4.2 Modelos Preditivos

Os métodos de aprendizado de máquina tradicionais geralmente recebem os dados de entrada em formato tabular, ou seja, cada instância X é representada por uma linha da tabela e cada coluna representa um atributo da instância. Entretanto a Tabela 4 (T4) é um bom exemplo de que cubos de n-dimensões, neste caso uma tabela-cruzada de duas dimensões, conseguem correlacionar melhor os dados a fim de fornecer um ganho de informação.

Esta correlação está fortemente ligada à maneira como os dados foram arranjados, introduzindo a noção de localidade. Tal noção seria bastante prejudicada em uma representação linear, ou seja, em uma representação tabular tradicional que consiste em apenas uma dimensão.

Um tipo de tarefa onde esse problema se apresenta claramente é o de classificação de imagens. A proximidade entre os *pixels* é um fator crucial para que o classificador possa reconhecer corretamente os objetos. Não por acaso, os métodos que apresentam os melhores resultados nesse tipo de tarefa baseiam-se em operações Convolucionais.

As Redes Neurais Convolucionais (CNNs) permitem não apenas aplicar convoluções baseadas em *kernels* (ou filtros), mas também construir hierarquias de camadas. Essas diferentes camadas são capazes de reprocessar os valores de saída, uma após a outra, de modo que podem reduzir a dimensionalidade dos dados e principalmente, identificar padrões mais complexos.

Entretanto, os primeiros experimentos realizados no estudo de caso relatado no Capítulo 5 mostraram que a hiper-dimensionalidade do cubo é um desafio considerável para a construção da arquitetura da CNN e de qualquer método a ser utilizado. A fim de explorar o impacto da redução de dimensionalidade no comportamento dos classificadores, foram propostas quatro diferentes representações dos dados.

As técnicas de redução da dimensionalidade podem trazer vantagens e desvantagens. O exemplo apresentado na Seção 4.1 ilustra bem que, ao mesmo tempo que novas informações podem ser obtidas, existem perdas significativas de informação. O senso comum sugere que o melhor neste caso seria encontrar o equilíbrio entre os extremos: dados originais *vs* dados resumidos (ou mesmo reduzidos).

De fato, para alguns classificadores testados neste trabalho, uma representação intermediária possibilitou os melhores resultados. Porém, as RNAs possuem a capacidade de, por si só, extrair as informações relevantes de dados. Assim, o grande desafio é encontrar a arquitetura capaz de obter sucesso na tarefa de predição sem necessitar do pré-processamento explícito dos dados.

Essa abordagem permite grandes benefícios. Primeiramente, torna o processo mais

genérico, visto que a própria CNN realizará o ajuste dos pesos dos filtros durante sua etapa de treinamento. Em segundo lugar, é menos dependente do *feeling* do cientista de dados que precisará reconhecer quais as melhores técnicas de redução para cada problema. Por outro lado, a principal desvantagem é a dificuldade de compreensão dos critérios utilizados pela rede, afinal, a interpretabilidade de modelos baseados em Redes Neurais Artificiais é um grande desafio.

A primeira vista, esse processo de pré-processamento implícito da rede neural se assemelha ao apresentado na Seção 2.4.4. De certa forma, a representação simplificada dos dados, em ambos os casos, utilizam-se matematicamente do mesmo princípio. O que muda é a forma como estas diferentes representações são obtidas.

Na redução de dimensionalidade tradicional utilizando *embeddings*, praticamente todo o processo é realizado de maneira automática. Isso o torna genérico para ser aplicado em diferentes domínios, o que acaba sendo uma grande vantagem. Por outro lado, essa abordagem é bastante suscetível a problemas de *over-fitting* caso o conjunto de treinamento não seja significativamente representativo.

Já a proposta de utilizar operações OLAP para consolidar e consequentemente reduzir a dimensionalidade dos dados utiliza-se de conceitos como a agregação automática. Isso sugere ser mais natural à base de dados trabalhada, pois implementa a hierarquia de dados definida no cubo. Após este processo, camadas convolucionais poderão extrair informações de caráter local dos dados. Mais do que isso, esta organização tende a ser menos suscetível a over-fitting devido à organização dos dados e a limitação espacial do kernel convolucional. A principal desvantagem é a falta de aplicabilidade caso os dados não possam ser organizados em um cubo OLAP.

Resta então decidir qual, dentre todas as transformações OLAP possíveis, permitirá o melhor desempenho de classificação? A resposta é muito provavelmente não apenas uma, mas sim um conjunto de transformações, como sugere o Teorema 4.2.1.

Teorema 4.2.1. *Seja uma função $f(x)$ um classificador tal que $f(x)$ possa ser escrita da forma:*

$$f(x) = g(\mathfrak{F}_o(x) \cdot W_o)$$

onde $\mathfrak{F}_o(x)$ seja uma transformação ótima baseada em operações OLAP e $W_o \in \mathbb{R}$ um conjunto de pesos ótimos que maximizem[1] *a qualidade de uma métrica μ qualquer.*

Considerando $\mathfrak{F}_i \in \mathfrak{F}^x$, onde $\mathfrak{F}^x$ é o conjunto das transformações OLAP de x, existe então $f'(x)$ tal que $\mu(f'(x)) \geq \mu(f(x))$, ou seja, $f'(x)$ é melhor ou igual a $f(x)$ se $f'(x)$ é da forma:

$$f'(x) = g'(\mathfrak{F}_1(x) \cdot W_1 \oplus \mathfrak{F}_2(x) \cdot W_2 \oplus ... \oplus \mathfrak{F}_n(x) \cdot W_n)$$

[1] ou minimize, se for uma função do tipo quanto menor melhor

Prova. A hipótese é que exista alguma função $\mathfrak{F}_i(x) \cdot W_i$ tal que $\mathfrak{F}_i(x) \in \mathfrak{F}^x$ e $W_i \neq 0$ e que faça $\mu(f'(x)) > \mu(f(x))$. Neste caso ideal, a condição de que $\mu(f'(x)) \geq \mu(f(x))$ seria atendida.

Por outro lado, considere que a hipótese anterior não seja satisfeita. Fazendo $W_2 = W_3 = ... = W_n = 0$ temos que:

$$f'(x) = g'(\mathfrak{F}_1(x) \cdot W_1 \oplus \mathfrak{F}_2(x) \cdot 0 \oplus ... \oplus \mathfrak{F}_n(x) \cdot 0)$$

$$f'(x) = g'(\mathfrak{F}_1(x) \cdot W_1)$$

Desta forma, basta fazer $\mathfrak{F}_1(x) = \mathfrak{F}_o(x)$ e $W_1 = W_o$ e assim $g'(x) = g(x)$ para que $\mu(f'(x)) = \mu(f(x))$. Consequentemente $\mu(f'(x)) \geq \mu(f(x))$, atendendo ao Teorema 4.2.1. □

Assim, se existe um conjunto $\mathfrak{F}^x$ com várias funções $\mathfrak{F}_i(x)$, uma função, ou no caso uma RNA devidamente otimizada, que utilize várias $\mathfrak{F}_i(x)$ teoricamente será melhor ou igual a uma função que utilize apenas uma delas. Esse é o princípio motivador da arquitetura proposta, intitulada de *OlapNet*.

Graficamente, sua arquitetura genérica pode ser vista na Figura 20. A ideia básica é, a partir dos dados originais, aplicar as funções definidas formalmente como $f(\mathfrak{F}_i(x) \cdot W_i)$. Essas funções foram representadas na ilustração pelas etapas de Transformações. Estas transformações fariam as operações OLAP, além de outros cálculos realizados por camadas convolucionais, por exemplo. Após estes processamentos, suas saídas são concatenadas para seguirem ao classificador, inicialmente também baseado em RNA.

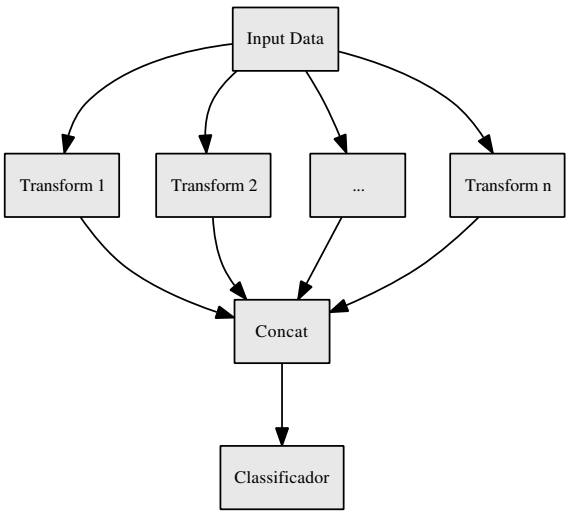


Figura 20 – Arquitetura genérica da rede proposta

4.3 Avaliação

Para avaliar a qualidade de um modelo preditivo, assim como qualquer outro método baseado em aprendizado de máquina, utiliza-se algumas métricas de avaliação. Estas métricas variam bastante de acordo com o tipo de problema investigado. Como será apresentado em detalhes no Capítulo 5, para avaliar o método proposto, foi realizado um estudo de caso de um problema de classificação.

Conforme apresentado por (YEH; LIEN, 2009), mensurar a qualidade de classificação apenas pela taxa de erros (ou acurácia) não é adequado em casos onde haja muito desbalanceamento dos dados. Como visto na Seção 5.1, esta é uma característica da base de dados investigada. Desta forma, será utilizada outra métrica de avaliação, a AUC.

4.3.1 AUC

A AUC é uma técnica que inicialmente classifica as instâncias em ordem descendente da probabilidade estimada para uma classe objetivo (*target*), criando um ranking. Estes dados são plotados ao longo do eixo X em um gráfico de duas dimensões. A cada acerto, ou seja, a cada instância que realmente é da classe objetivo, incrementa-se uma unidade no eixo Y em relação ao último valor plotado. Ao final do processo tem-se curvas como as vistas na Figura 21 e quanto maior a área sob essas curvas, melhor teoricamente o resultado.

Nota-se que um classificador aleatório, que classifica todas as instâncias com a mesma probabilidade, terá o comportamento de uma reta e sua área será igual a 0,5. Já o classificador teórico "perfeito" terá um valor próximo a 1, mas nunca igual a 1 já que

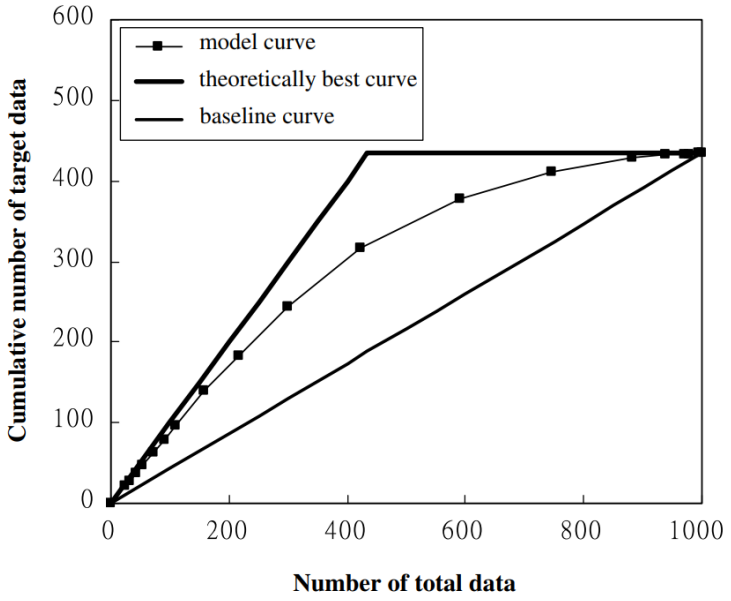


Figura 21 – Exemplo de uma AUC. Fonte: (YEH; LIEN, 2009)

dependerá da quantidade de instâncias verdadeiramente da classe alvo existente na base.

4.3.2 Método *Holdout*

O método *Holdout* cria partições de dados em dois subconjuntos mutualmente exclusivos chamados conjunto de treinamento e conjunto de teste, ou conjunto *Holdout* (KOHAVI, 1995). O classificador utiliza o conjunto de treinamento para criar o modelo de classificação e em seguida utiliza-se o conjunto de teste para validar a qualidade da classificação.

Em geral, o conjunto de treinamento corresponde a 2/3 do total dos dados, restando 1/3 para o conjunto de teste. Já outros autores propõe a divisão 80%-20% para treinamento e teste respectivamente, alegando que poucos dados estariam disponíveis para a criação do modelo, para pequenas bases de dados em especial.

Alguns cuidados devem ser tomados no momento desta divisão. Deve-se definir a existência ou não de sobreposição de instâncias entre os conjuntos. Em geral, é desejado que o classificador não conheça as instâncias de validação, para reduzir o viés do mesmo. Entretanto, mesmo que seja definida a não-reposição das instâncias, outras "armadilhas"podem ocorrer.

A base de dados utilizada no estudo de caso possui um histórico temporal onde cada instância se refere a algum mês em um intervalo de alguns anos. Desta forma, caso a divisão entre conjunto de treino e teste seja feita aleatoriamente, é possível que instâncias muito parecidas estejam em ambos os conjuntos, prejudicando a qualidade dos testes. Por esta razão, a divisão dos dados foi feita considerando este intervalo. Além disso, a base de dados possui mais de 400 mil registros, e métodos de validação como o *10-fold cross validation*, por exemplo, geraria um custo computacional consideravelmente maior.

Outro ponto de atenção é o viés que pode ser causado pelo processo de ajuste de hiper-parâmetros como, por exemplo, a quantidade de camadas de uma RNA. Assim, o conjunto de teste foi isolado e todos os ajustes foram feitos utilizando o próprio conjunto de treinamento. Neste conjunto de treinamento foi criado um novo subconjunto de validação, como visto na Figura 22.

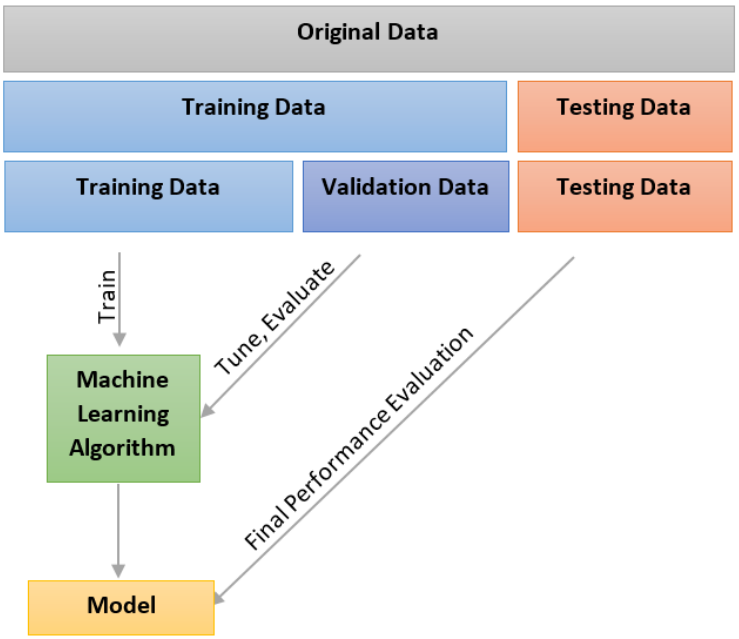


Figura 22 – Metodologia de execução dos experimentos. Fonte: (TANWAR, 2018)

5 Experimentos

Com o objetivo de avaliar a proposta em um cenário real, aplicamos as estratégias apresentadas em um problema do mercado de crédito. Neste setor, as instituições financeiras precisam compreender o comportamento dos seus clientes sob diversos pontos de vista. Assim, surgiram várias tarefas específicas envolvendo modelagens estatísticas como *credit scoring*, *behavioural scoring*, *collections scoring*, *fraud scoring*, dentre outros, conforme apresentado no Capítulo 3.

O problema investigado aqui pode ser classificado como um tipo de escoragem comportamental (*behavioural scoring*), visto que o objetivo é analisar o comportamento dos clientes em relação a possíveis contratações de crédito. Predizer estes casos é particularmente útil pois assim a instituição será capaz de realizar ofertas mais assertivas, captar recursos de acordo com a expectativa de demanda, além de poder acompanhar proativamente o comportamento daqueles clientes que detêm os maiores riscos.

Assim, dado um momento específico, deseja-se classificar os clientes de acordo a propensão destes contratarem créditos ou não em um futuro próximo. Como já mencionado, a metodologia baseou-se naquela utilizada pelo artigo (YEH; LIEN, 2009). Além do emprego de dados financeiros reais, os métodos utilizados e os critérios de avaliação são bastante semelhantes.

Na próxima seção serão apresentadas informações dos dados utilizados e o pré-processamento aplicado. Em seguida, estão relatadas as transformações aplicadas aos dados, de acordo com as diretrizes propostas. Posteriormente estão listados os métodos, *frameworks*, além das configurações de parâmetros e as arquiteturas das redes neurais que apresentaram os melhores resultados. Por fim os resultados são apresentados de forma comparativa.

5.1 Base de Dados

O dados utilizados foram extraídos de uma amostra de dados reais anonimizados, fornecidos por uma instituição financeira[1]. Essa base contém o histórico mensal de endividamento dos clientes entre os meses de janeiro de 2014 a março de 2018. A quantidade de clientes variou entre 25 e 50 mil ao longo deste período de mais de quatro anos.

Nos arquivos originais, os dados foram organizados em uma estrutura hierárquica onde cada cliente era representado como visto na Figura 23.

[1] Os dados utilizados no estudo não são públicos, por isso não estão disponíveis para download

1. Mês 1 (Ex.: Janeiro de 2014):
    a) Modalidade 1 (Ex.: Cartão de Crédito):
        i. Prazo 1 (Ex.: A vencer até 30 dias): **VALOR**
        ii. Prazo 2: **VALOR**
        iii. Prazo 3: **VALOR**
        ...
        xviii. Prazo 18: **VALOR**
    b) Modalidade 2:
        i. Prazo 1: **VALOR**
        ii. Prazo 2: **VALOR**
        ...
    c) Modalidade 3:
    ...
    h) Modalidade 8: ...
2. Mês 2:
...

Figura 23 – Representação hierárquica dos dados originalmente fornecidos

Assim, para cada cliente há um Histórico mês a mês. Em cada mês, existem registros relacionados a oito Modalidades de Crédito. Em cada modalidade existem 18 faixas de Prazos de pagamento dos créditos indicados por **VALOR**. Caso o cliente não tenha endividamento naquele Mês/Modalidade/Prazo, utiliza-se o valor 0 (zero).

5.1.1 Pré-processamento

A partir destes dados foi elaborada uma estratégia para a criação das instâncias de treinamento e teste. Para isso, considerou-se uma janela de 12 meses deslizante ao longo do tempo. Por exemplo, um cliente qualquer possuía uma determinada situação financeira em agosto de 2016. Logo, a instância correspondente teria dados compreendidos entre setembro de 2015 e agosto de 2016.

Para definir os rótulos das instâncias foram analisados os três meses posteriores (no exemplo: setembro, outubro e novembro de 2016). Se o cliente, neste período, tiver contratado novos créditos em modalidades de interesse da instituição financeira, a instância é então rotulada com o valor 1. Caso contrário, ela recebe o rótulo 0 (zero). Assim, um mesmo cliente pode estar representado em várias instâncias diferentes e por consequência com rótulos não necessariamente iguais, desde que em intervalos de tempo distintos.

Desta maneira, tendo 12 meses de Histórico (por instância), 8 Modalidades de crédito e 18 faixas de Prazo de Pagamento, temos 1728 atributos para a classificação. A

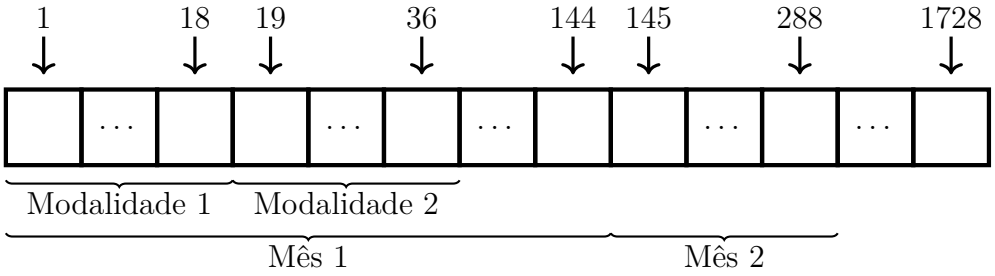


Figura 24 – Representação gráfica simplificada de uma instância da base

Figura 24 mostra uma representação simplificada destes dados organizados em um arranjo. Embora ausente na figura, além das 1728 colunas, existe o rótulo da instância, é claro.

Dentre todos os dados disponíveis foram feitas remoções verticais, ou seja, algumas instâncias foram excluídas. O principal caso em que se fazia remoções é quando clientes não possuem histórico de endividamento no período analisado. Para estes casos, todos os atributos dos 12 meses eram nulos e por isso seriam necessárias informações adicionais para uma classificação realmente efetiva. Embora a instituição disponha de outros dados como Idade, Sexo, Estado Civil, dentre outros, optou-se por não incluí-los a fim de restringir o escopo da pesquisa aos dados financeiros numérico/temporais.

Como os dados de janeiro a novembro de 2014 não possuíam a janela completa de 12 meses anteriores disponíveis, foram criadas instâncias apenas de dezembro de 2015 a dezembro de 2017. Os três últimos meses, referentes a 2018 também não geraram instâncias diretamente, seguindo o mesmo raciocínio para os três meses posteriores. Assim, foram geradas 402.752 instâncias, sendo 350.439 exemplos negativos e apenas 52.313 exemplos positivos. Em outra palavras, apenas 13% da base representava situações em que os clientes efetivamente haviam contratado crédito, o que caracteriza uma base consideravelmente desbalanceada.

5.2 Instanciação da Proposta

Assim como apresentado na Seção 4.1, a proposta primeiramente envolve identificar o conjunto de características que, agrupadas em determinada ordem, fornecem algum significado semântico. Desta forma, espera-se que métodos como as Redes Neurais Convolucionais tirem proveito dessa organização a partir da análise da vizinhança dos atributos.

5.2.1 Transformações dos dados

De fato, nos dados trabalhados as três características gerais presentes, Histórico, Modalidade de Crédito e Prazo de Pagamento, podem ser melhor organizadas. Partindo da

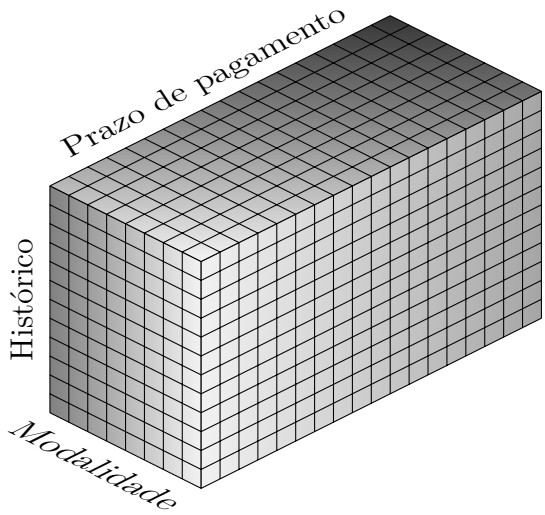


Figura 25 – R1: Dados originais, organizados tridimensionalmente

Figura 24, percebemos que as colunas 1, 19, 37, etc. dizem respeito ao Prazo 1, enquanto as colunas 2, 20, 38, etc. ao Prazo 2, e assim por diante.

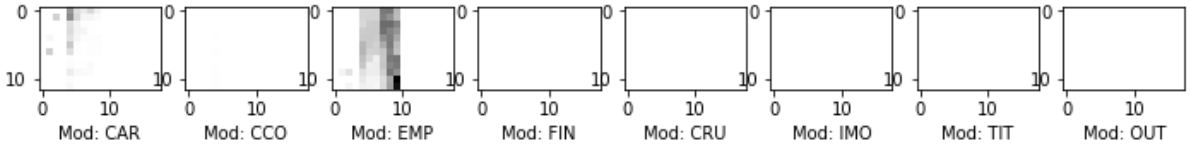
Por outro lado, as colunas de 1 a 18, 145 a 162, 289 a 306, etc. representam a Modalidade 1, enquanto as colunas de 19 a 36, 163 a 180, 307 a 324, etc. representam a Modalidade 2. Já sob o ponto de vista do Histórico, as colunas de 1 a 144 compõe o Mês 1, as colunas 145 a 288 o Mês 2, e assim sucessivamente.

Entretanto, uma representação tabular com um arranjo unidimensional de colunas como mostrado na Figura 24 é incapaz de representar eficientemente bem as relações destas três características gerais. Faz-se necessário portanto o uso das matrizes multi-dimensionais, pois criando novas dimensões é possível agrupar os dados evidenciando as relações de dependência naturalmente existentes.

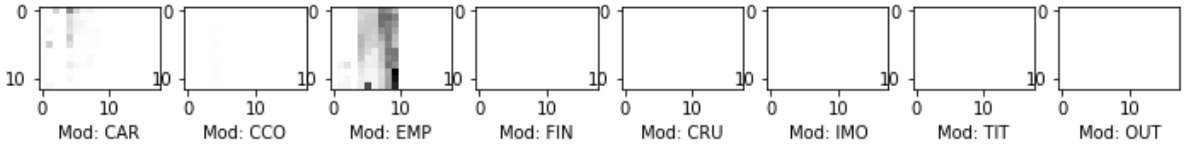
Foram utilizadas três dimensões[2] representando o Histórico, Modalidade e Prazo de Pagamento com comprimentos de 12, 8 e 18 colunas, respectivamente. De forma gráfica, essa representação gera um cubo como mostrado na Figura 25. Já á Figura 26 mostra duas instâncias em momentos distintos, antes e depois da contratação de crédito pelo cliente. Os valores foram plotados em escala de cinza, onde quanto mais escuro, maior o valor do crédito para aquele Histórico (linha) / Prazo (coluna) / Modalidade (página).

Note que apesar da reformulação dos dados (*reshape*) a mudança na organização dos mesmos não alteraram a quantidade de atributos para classificação. Um total de 1728 colunas é um desafio considerável para o classificador que deverá lidar com essa alta

[2] Não confundir as dimensões criadas para representar as características gerais (Histórico, Modalidade de Crédito e Prazo de Pagamento) com o espaço de 1728 dimensões dos dados



(a) Visualização de instância de um cliente prestes a contratar crédito ($target = 1$)



(b) Visualização de instância de um cliente após a contratação

Figura 26 – Apresenta as matrizes (Prazo vs Histórico) das oito Modalidades para um cliente observado em dois momentos. Em (a) ele está prestes a contratar mais créditos, ou seja, $target = 1$. Já em (b), no mês seguinte, ele já o contratou, como observado na última linha da Modalidade EMP (Empréstimo).

dimensionalidade.

5.2.1.1 Redução de Dimensionalidade

Embora existam diversas estratégias para reduzir o número de atributos de classificação em uma base de dados, algumas transformações são intuitivas em alguns cenários. Para o problema em questão, naturalmente pode-se realizar uma série de agregações visto que os dados seguem uma hierarquia bem definida. Desta maneira, é possível que os resultados dos classificadores sejam melhores em dimensionalidade menor.

Como explicado, os valores numéricos estão divididos em Históricos mensais e estes por sua vez estão subdivididos em Modalidades de crédito que estão subdivididos em Prazos de pagamento. Consequentemente, somando os valores de todas as Modalidades de crédito (m), obtém-se o endividamento daquele cliente naquele mês, naquela faixa de Prazo de pagamento. Matematicamente, o endividamento total[3] de um cliente c, em um histórico mensal h, em um prazo p, denotado por $VALOR_{c,h,p}$, será dado por (5.1).

$$VALOR_{c,h,p} = \sum_{m=1}^{8} VALOR_{c,h,p,m} \tag{5.1}$$

Essa agregação ocasiona obviamente perda de informação. Se antes era possível saber se a dívida de um cliente estava composta mais por cartão de crédito (CAR), ou empréstimo (EMP), por exemplo, agora sabe-se apenas o montante dos seus endividamentos a cada mês, em cada faixa de Prazo de pagamento mapeado. O grande benefício por outro lado, é a redução de 1728 para 216 colunas, o que equivale a 12,5% dos dados originais, tornando o processo de aprendizagem do modelo muito menos complexo.

[3] Por utilizar a função de agregação SOMA

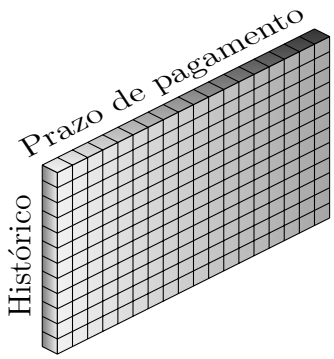


Figura 27 – R2: Representação gráfica dos dados após agregação pela soma das Modalidades de crédito

O resultado desta agregação de dados nos fornece graficamente a visão mostrada na Figura 27. Note que, apenas com a função de agregação soma (Σ) reduzimos a representação original que chamaremos de **R1** (Figura 25), para uma nova representação **R2**.

Além da transformação $R1 \rightarrow R2$, analogamente partindo de R1 é possível realizar outros tipos de agregação. Além de utilizar outras funções como MÉDIA, MÁXIMO, MÍNIMO, etc., o problema permite agregar outra dimensão. Esse tipo de operação é chamada *roll-up* no contexto de cubos OLAP. Agregando o Prazo de pagamento, temos:

Seja o endividamento total de um cliente c, em um histórico mensal h, em uma Modalidade de crédito m. O valor denotado por $VALOR_{c,h,m}$ será dado por (5.2).

$$VALOR_{c,h,m} = \sum_{p=1}^{18} VALOR_{c,h,m,p} \tag{5.2}$$

O resultado desta nova agregação de dados nos fornece a representação **R3**, vista na Figura 28. Ao contrário da transformação $R1 \rightarrow R2$, $R1 \rightarrow R3$ resultou em 96 atributos para classificação.

Por fim, é possível combinar as transformações realizadas em R2 e R3 através das funções de agregação compostas para gerar uma nova representação dos dados **R4**. Neste caso, os somatórios foram combinados a fim de agregar os dados da Modalidade de crédito e do Prazo de pagamento simultaneamente.

Logo, o endividamento do cliente c, em um histórico mensal h, denotado por $VALOR_{c,h}$, é dado por (5.3).

$$VALOR_{c,h} = \sum_{m=1}^{8} \sum_{p=1}^{18} VALOR_{c,h,m,p} \tag{5.3}$$

Assim como nas outras representações, a visualização da matriz resultante é mostrada na Figura 29. Vale observar que neste caso os dados foram significativamente suprimidos e perderam-se informações que poderiam ser úteis para a classificação.

Se por um lado espaços de dimensões menores facilitam a tarefa do classificador, por outro informações que podem ser relevantes eventualmente podem ser perdidas. Encontrar o ponto ideal diante destes dois extremos é um dos desafios do processo de mineração de dados, que é investigado neste trabalho em especial.

5.2.1.2 Normalização

Para cada uma das quatro representações geradas da base, R1, R2, R3 e R4 foi criada uma representação correspondente em que normalizou-se ao intervalo $[0, 1]$ os valores verticalmente, ou seja, a nível de instância, utilizando a Equação 2.32. Assim foram criadas as representações R1', R2', R3' e R4' para R1, R2, R3 e R4, respectivamente. Os resultados evidenciaram que alguns métodos se beneficiam bastante desta transformação, incluindo os baseados em Redes Neurais Artificiais.

5.2.2 Métodos utilizados

As oito representações da base de dados foram submetidas a diversos métodos de aprendizado de máquina, sendo eles: KNN, Regressão Logística, Análise Discriminante, Naive Bayes, Árvore de Decisão e Redes Neurais, apresentados na Seção 2.3.

Com exceção das Redes Neurais, todos os outros métodos foram executados utilizando as implementações da biblioteca *Python*, Scikit-learn (PEDREGOSA, 2011). Os parâmetros foram mantidos nas suas configurações padrão e as representações matriciais

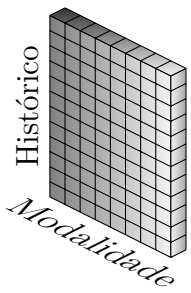


Figura 28 – R3: Representação gráfica dos dados após agregação pela soma dos Prazos de pagamento

Figura 29 – R4: Representação gráfica dos dados após agregação pela soma das Modalidades de crédito e dos Prazos de pagamento simultaneamente

da base foram serializadas para adaptar à entrada dos métodos. Na R2 por exemplo, os dados foram linearizados em um arranjo de 216 colunas.

Já para a execução dos experimentos envolvendo Redes Neurais, foi utilizado o *framework* de *Deep Learning*, *PyTorch* (PASZKE, 2017). O tamanho do *batch* foi de 4000 instâncias, e as ilustrações foram feitas com a biblioteca *HiddenLayer*[4]. A representação R4 não foi investigada para as arquiteturas convolucionais, visto que se resume a um *array* de apenas uma dimensão e 12 colunas. Os resultados utilizando outros métodos indicaram que neste caso a perda de informação durante a redução da dimensionalidade foi muito grande, comprometendo a qualidade da classificação.

Embora diversas arquiteturas tenham sido avaliadas nos experimentos iniciais, apenas três foram escolhidas, uma para cada tipo de representação, sendo elas:

- **R1**: Rede Neural Convolucional proposta a partir das redes utilizadas para as representações R2 e R3, apresentadas logo a seguir. Nela, cada entrada possui 1728 atributos organizados inicialmente em um *array* tridimensional de tamanho $(8 \times 12 \times 18)$ retratando as dimensões Modalidade de crédito, Histórico e Prazo de pagamento, respectivamente.

  A partir dos dados de entrada, dois fluxos de processamento são disparados. No primeiro é realizada uma operação de agregação que mapeia $R1 \rightarrow R2$ e no segundo faz-se $R1 \rightarrow R3$ (ou se for o caso, $R1' \rightarrow R2'$ e $R1' \rightarrow R3'$, respectivamente). Em seguida são realizadas normalizações em *batch* e aplicadas as camadas de processamento com os filtros convolucionais baseados nas redes usadas nas representações R2 e R3.

  As saídas das duas séries de camadas produzem tensores de tamanho $(64 \times 4 \times 8)$ cada,

[4] https://github.com/waleedka/hiddenlayer

que são então serializados (via *Reshape*) para o tamanho (2048), que são concatenados em um único tensor de (4096). A este ponto, as principais características dos dados já foram extraídas e segue-se então para as camadas lineares *fully-connected* realizarem a classificação. Ao final, uma função *Softmax* é responsável por traduzir as saídas para valores percentuais. Toda a arquitetura pode ser vista em detalhes na Figura 30.

- **R2**: Rede Neural Convolucional que aplica normalização em *batch* seguido por duas sequências de convoluções, nova normalização, ativação pela função $ReLU$ e então *Dropout* com probabilidade de 50%. Após as duas sequências de camadas, um *Reshape* serializa os dados que, após normalizados, seguem para as camadas lineares.

  Três camadas *fully-connected* progressivamente reduzem a quantidade de neurônios de saida para 512, 256 e finalmente 2, que após a *Softmax* representam as probabilidades de classes do problema. A arquitetura completa é vista na Figura 31a.

  Foram testadas algumas variações como inversões das ordens de $ReLU$ e *Dropout*, ou remoção de algumas normalizações, por exemplo. Entretanto essa foi a configuração que produziu os melhores resultados no conjunto de validação.

- **R3**: Rede Neural Convolucional semelhante à utilizada para a representação R2, modificando-se os tamanhos dos *kernels* convolucionais. Como visto em 5.2.1.1, R2 possui tamanho superior a R3. Assim, os *kernels* de tamanho (5×7) e (5×5) foram alterados para (3×3) e (3×3), respectivamente.

  Todas as demais configurações foram mantidas as mesmas. A arquitetura completa é apresentada na Figura 31b.

Todos os métodos citados foram executados uma vez para cada representação, com exceção das Redes Neurais em R4, como já mencionado. Em todos os casos as 253.215 instâncias referentes de dezembro de 2014 a dezembro de 2016 foram utilizadas para treinamento, enquanto os dados de janeiro a dezembro de 2017 foram utilizados para os testes, como sugere o método Hold-Out, Seção 4.3.2.

Por consequência, no processo de ajuste de hiper-parâmetros, realizado experimentalmente, de forma manual, somente instâncias do conjunto de de 2014 a 2016 foram utilizadas. Como apresentado em 4.3.1, a AUC foi utilizada como a métrica de avaliação da qualidade dos modelos. A Figura 32 mostra a Função de Custo e a Figura 33 mostra a AUC ao longo do processo de treinamento da *OlapNet*, rede proposta e aplicada à R1'.

A rede obteve sua melhor relação treino / validação após 1340 épocas. Após este ponto, a qualidade do resultado começou a degradar, definindo assim este hiper-parâmetro. Já para os pares de redes R2, R2' e R3, R3', foram utilizadas respectivamente 1350 e 1265 épocas, seguindo o mesmo procedimento.

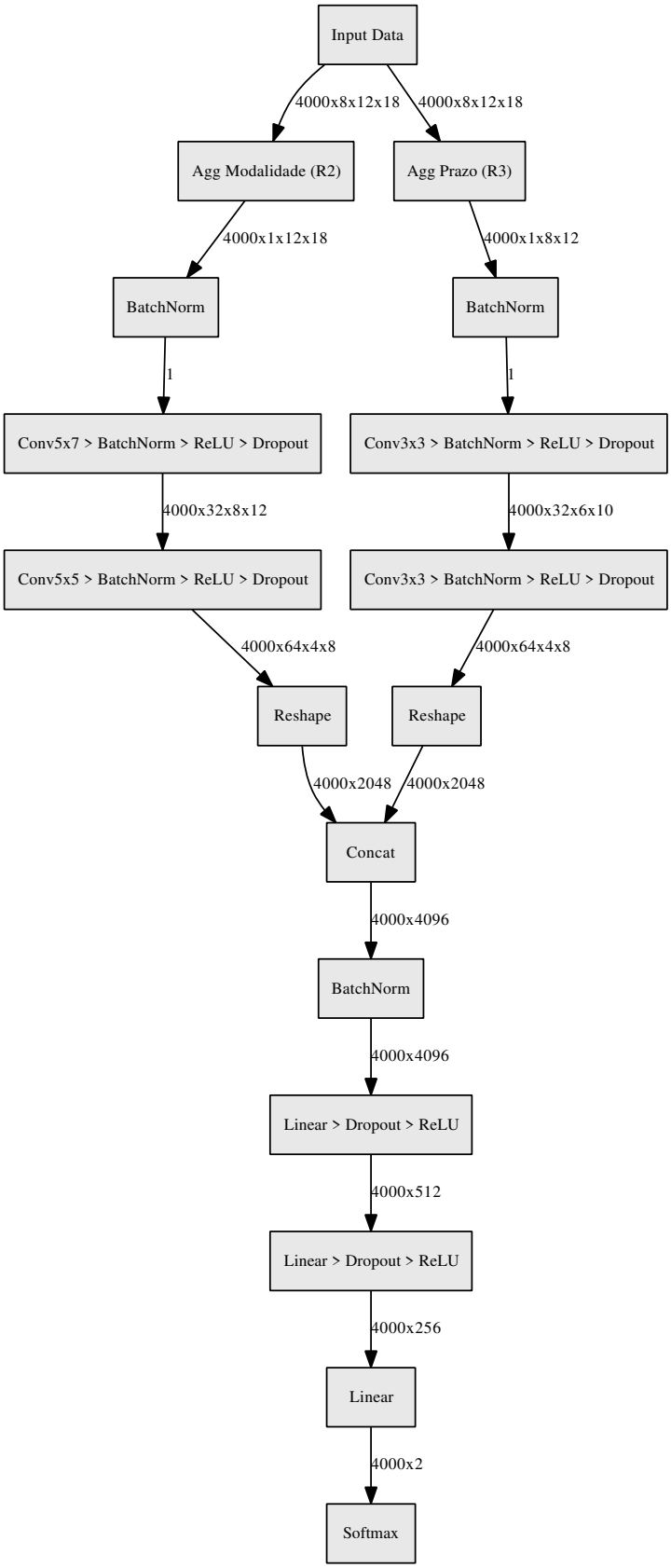
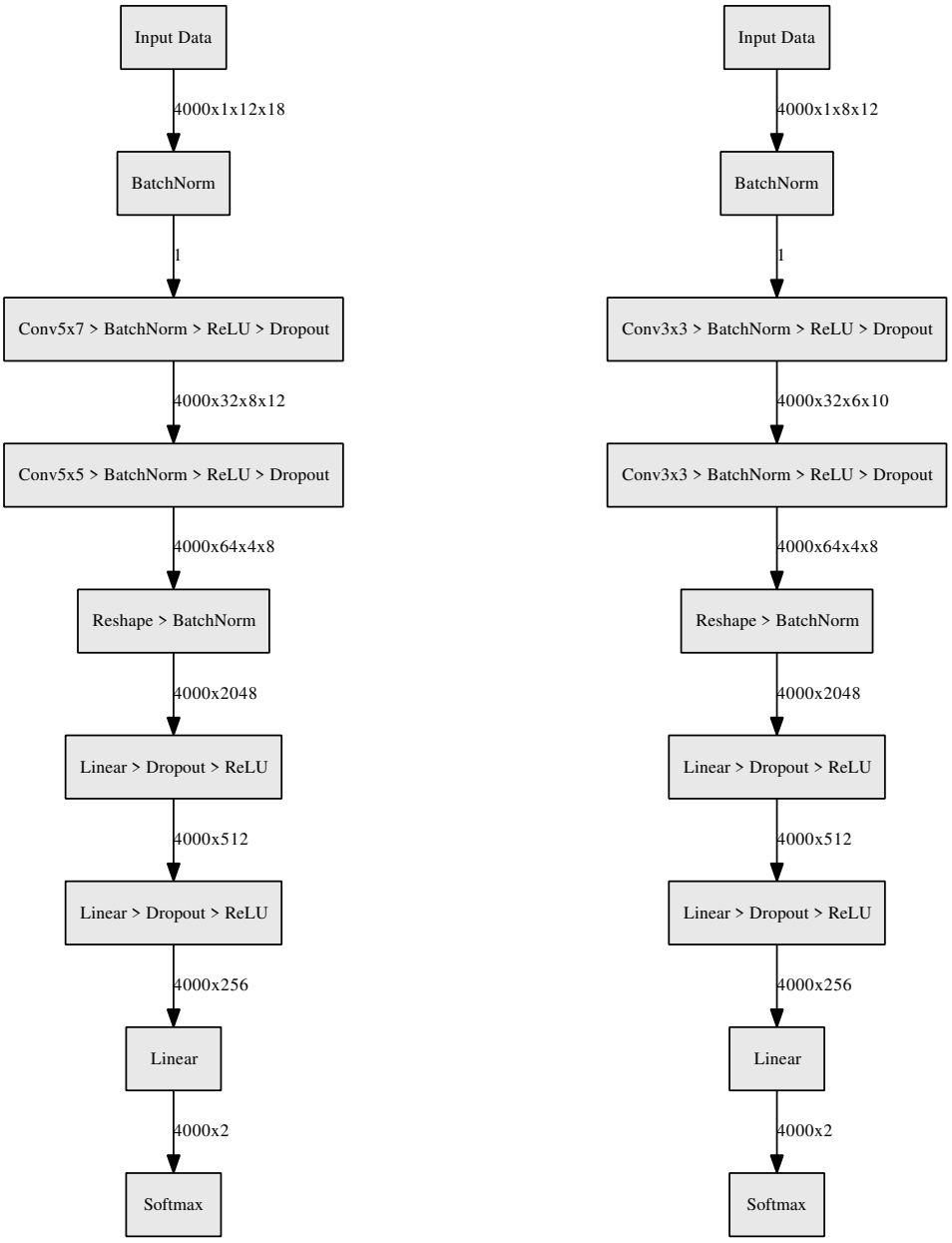


Figura 30 – OlapNet: Arquitetura proposta para dados organizados como um Cubo OLAP

5.3 Resultados

Os experimentos foram realizados em um Linux Mint 18.1 Serena 64 bits, processador Intel®Core™i7-6700 CPU @ 3.40GHz com 16 Gb de memória RAM e GPU NVIDIA GeForce GTX 1070. As versões das bibliotecas utilizadas foram Scikit-Learn 0.20.1 e PyTorch 1.0.1.post2.

A Tabela 7 apresenta os valores das AUC de cada método em cada formato da



(a) CNN para R2 (b) CNN para R3

Figura 31 – Redes propostas para as representações R2 (a) e R3 (b)

base de dados. Os melhores resultados para cada representação foi destacado em negrito, tendo a arquitetura de RNA Convolucional proposta apresentado o melhor dentre todos eles, utilizando-se o formato de representação R1'. A exceção das representações R4 e R4', onde as RNAs não foram testadas, somente na R3 o método Naïve Bayes foi superior.

Vale destacar que o foco deste trabalho não é comparar os diferentes algoritmos entre si, até porque o esforço nas construções das soluções baseadas em RNA foi consideravelmente maior. De qualquer forma, algumas observações ficam claras, ao menos sob o ponto de vista do problema investigado.

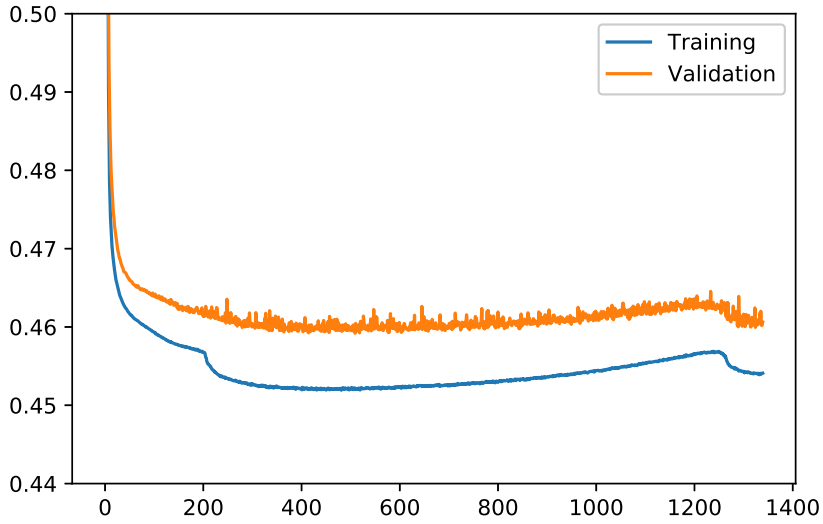


Figura 32 – Treinamento da *OlapNet* - Função de Custo

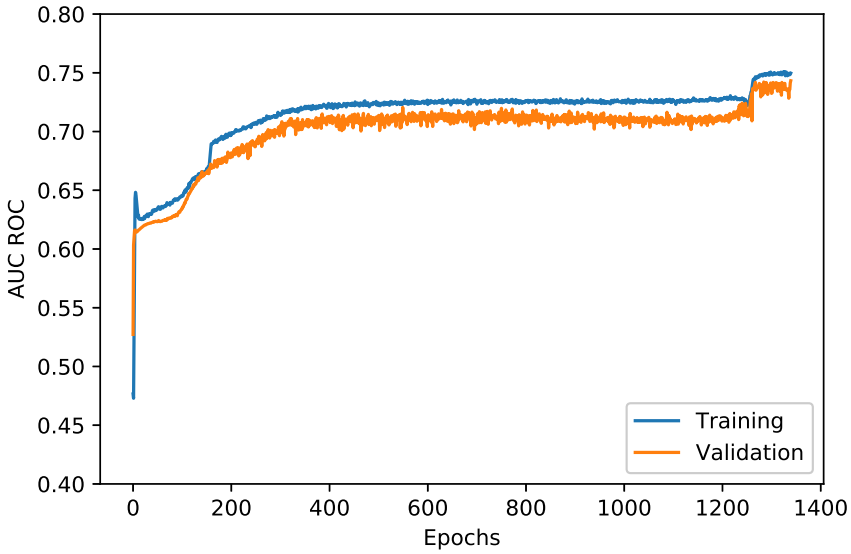


Figura 33 – Treinamento da *OlapNet* - AUC

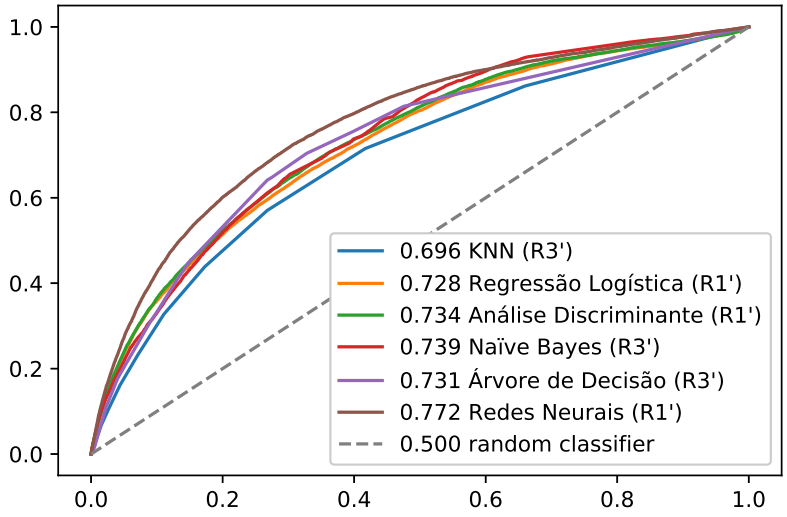


Figura 34 – Melhores curvas ROC obtidas por cada método utilizado

Os métodos clássicos deixam claro que a redução demasiada da dimensionalidade dos dados ocasiona perda de informação relevante para a tarefa de classificação, visto que em praticamente todos os casos R4 e R4' foram as piores configurações para todos eles. Por outro lado, a redução feita em R3 trouxe ganhos notáveis para Naïve Bayes, Árvore de Decisão e KNN em menor grau, o que não aconteceu para a Regressão Logística e Análise Discriminante. Para estes métodos porém, a normalização ao intervalo $[0, 1]$ melhorou significativamente a classificação, assim como nas RNAs.

Considerando como linha base um classificador aleatório de AUC = .500, este resultado (.772) se comparado ao melhor dentre os outros métodos (.739 para o Naïve Bayes em R3 e R3') apresenta ganho de $13,8\%$. O comparativo com as melhores AUCs de cada método é visto na Figura 34.

Ficou bem evidente que a agregação que resultou em R2 não foi muito útil visto que mesmo a RNA apresentou uma grande discrepância se comparado aos resultados obtidos em R1, R3, e suas respectivas representações normalizadas. A RNA aplicada à R3' se mostrou competitiva, enquanto a melhor configuração ficou mesmo para a arquitetura

Resultados	R1	R2	R3	R4	R1'	R2'	R3'	R4'
KNN	.668	.643	.676	.602	.682	.648	.696	.611
Regressão Logística	.466	.411	.460	.393	.728	.643	.695	.611
Análise Discriminante	.663	.636	.663	.649	.734	.645	.696	.613
Naïve Bayes	.717	.648	**.739**	.589	.717	.648	.739	.589
Árvore de Decisão	.725	.669	.724	**.657**	.722	.658	.731	**.622**
Redes Neurais Artificiais	**.735**	**.693**	.731	-	**.772**	**.707**	**.762**	-

Tabela 7 – Resultados dos experimentos

proposta neste trabalho como *OlapNet*, utilizando a representação R1'. O gráfico com as curvas ROC obtidas pelas diferentes RNAs é visto na Figura 35.

Entretanto, cabe aqui uma observação. O resultado obtido com a rede em R1' só foi possível a partir da construção híbrida entre as melhores redes utilizadas em R2' e R3'. Sem a utilização de fluxos de processamento concorrentes, como visto na Figura 30, todas as redes experimentadas, inclusive utilizando Convoluções 3D, não apresentaram resultados satisfatórios. Isso além de evidenciar o problema da hiper-dimensionalidade, reforça a ideia de que a arquitetura proposta na *OlapNet* é promissora.

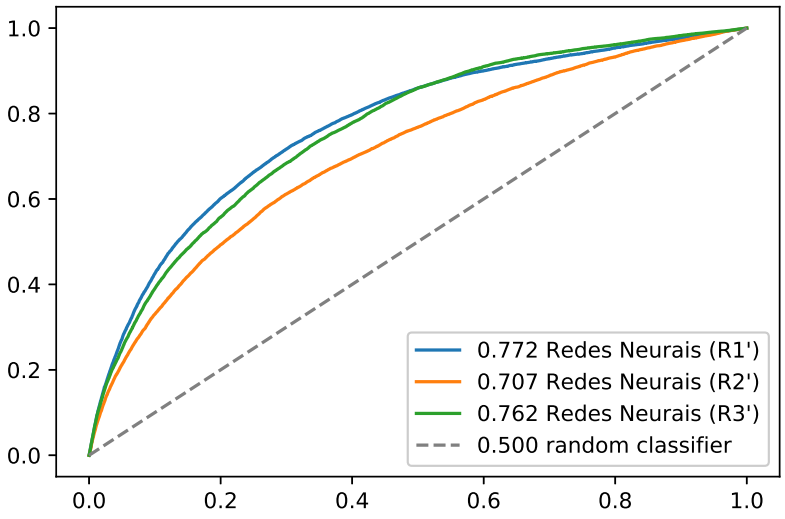


Figura 35 – Curvas ROC obtidas pelas RNAs

6 Conclusões

Este trabalho permitiu a aplicação e compreensão do funcionamento das Redes Neurais Artificiais e as principais técnicas necessárias para a construção de um projeto baseado em *Deep Learning*. Ficou evidente que os problemas relatados em livros, artigos e outros materiais de referência realmente são muito comuns e aconteceram em diversas situações durante os experimentos.

Um dos maiores problemas enfrentados para encontrar a configuração de hiper-parâmetros ideal foi o *overfitting*. Esta dificuldade sugere, dentre outras coisas, que apesar da base de treinamento utilizada ter centenas de milhares de registros, como o modelo de *Deep Learning* criado possui também muitos parâmetros, ele é robusto o suficiente para se super-ajustar aos dados.

Além das técnicas de regularização utilizadas, uma maneira de contornar este problema seria obtendo mais dados. Neste ponto vemos diferença entre o Aprendizado Profundo e o tradicional, pois tradicionalmente é comum reduzir o tamanho da base de treinamento através de amostragem. O KNN por exemplo, levou 24 horas para executar a classificação em um dos testes realizados, e poderia ser bem mais rápido com uma base menor. Já em modelos baseados em redes neurais profundas, é desejável que as bases de dados sejam maiores.

Por outro lado, o treinamento das CNNs também não foi rápido e poderia ser muito mais demorado. Pelo tamanho da base, foi possível carregar todas as instâncias para a memória, o que acelerou bastante as rotinas. Além disso, é claro, o processamento realizado em GPU fez muita diferença. Em média, o treino de cada rede demorou entre 1 e 3 horas. Logo, se a proporção relatada por (CIRESAN, 2011) for aplicável, o treinamento em CPU poderia levar dias.

Vale ressaltar que alguns dados não foram incluídos. Instâncias de clientes sem histórico de endividamento, que resultavam em um cubo zerado foram removidos da base e dados como Sexo, Idade, Estado Civíl e outras características descritivas dos clientes não foram incluídas para nenhuma instância na base. Isso sugere que, adicionando estas novas informações, é possível aprimorar os resultados do modelo.

Ainda sobre os problemas enfrentados durante o treinamento, felizmente as técnicas de regularização permitiram contorná-los eficientemente. As regularizações L2 e Dropout, mantiveram ótimos níveis de generalização. Como visto na Figura 33, a AUC do conjunto de validação (desconhecido), que naturalmente tende a ser ligeiramente inferior à AUC do conjunto de teste (conhecido), acompanhou a evolução do resultado do conjunto de treinamento durante toda a rotina.

Outra questão que chama a atenção é a estagnação da AUC (Figura 33 na maior parte do treino, até a época 1250, quando a rede consegue melhorar consideravelmente. A Figura 32 explica o que aconteceu. O otimizador muito provavelmente encontrou um mínimo local e o *Momentum* lidou de escapar dele.

Curiosamente, após essa melhoria a rede começou a degradar-se rapidamente, até divergir completamente do objetivo (não mostrado nos gráficos). Uma das causas possíveis é que haviam dois parâmetros de *Momentum*, um no otimizador Adam e outro nas camadas de *Batch Normalization*. É provável que o *Momentum "acelerou"* muito o passo, o que tornou possível escapar do mínimo local, mas passou por outro mínimo rápido demais. Neste caso, bastou utilizar a estratégia *(Early Stopping)* e parar ao atingir a época desejada.

Desta forma, o trabalho mostrou formalmente e com experimentos empíricos que, ao utilizar CNNs em dados estruturados como cubos de dados, os resultados seguindo as diretrizes da *OlapNet* tendem a ser melhores do que utilizando transformações baseadas em operações *Roll-up*. Embora estas operações reduzam a dimensionalidade dos dados e facilite a tarefa de classificação para alguns métodos, elas ocasionam perdas de informação que poderiam, com as devidas precauções, melhorar os resultados da classificação.

Já a arquitetura proposta permite avaliar em seu espaço de busca várias combinações destas operações OLAP. Isso pode ser visto como uma investigação automática de diferentes tipos de transformações de dados. Assim, o processo fica menos dependente da intervenção humana, o que em geral, é uma característica desejada no processo de KDD. Além disso, a rede fará a combinação de diferentes tipos de transformações simultaneamente, aplicando múltiplas delas se necessário, como foi o caso.

Nos experimentos realizados, com a representação completa ou seja, utilizando a *OlapNet*, o melhor ajuste foi atingido com 1340 épocas, enquanto que na rede com a representação R3' foram necessárias 1265. Desta forma, com menos de 6% de épocas a mais no treinamento, a rede proposta conseguiu atingir um resultado 13,8% melhor. Vale ressaltar que a rede utilizada em R1' é mais complexa que a utilizada em R3'. Logo, a depender da sua complexidade computacional, a arquitetura apresentada pode ser uma escolha interessante em muitos cenários.

Além do custo computacional, outro fator relevante para a aplicação deste métodos em diversas organizações como empresas, por exemplo, é a relativa complexidade para configurar corretamente todos os hiper-parâmetros existentes. Como visto, na Seção 2.4, existem parâmetros dos mais diversos, muitas das vezes específicos para tratar determinado problema dentre os vários desafios possíveis.

Por esta razão, os métodos tradicionais ainda são relevantes para muitas tarefas e podem ser uma boa opção caso não se tenha recursos disponíveis para utilização de modelos baseados em *Deep Learning*. Por outro lado, a disponibilização de *frameworks* abertos,

a democratização permitida pela utilização de GPUs, a possibilidade de contratação de serviços em nuvem e a rápida difusão de trabalhos relacionados ao tema pela internet, tornam esta nova tecnologia cada vez mais acessível.

A propósito, este trabalho buscou organizar as principais informações necessárias para a introdução do leitor ao assunto e propor uma abordagem prática para lidar com problemas envolvendo dados estruturados. Os resultados obtidos são muito animadores e faz-se necessários outros estudos, preferencialmente com outras aplicações práticas, no sentido de que sejam levantadas novas evidências sobre a superioridade (ou não) de desempenho deste tipo de abordagem.

Assim, concluímos que atualmente o *Deep Learning* é o estado-da-arte para muitas tarefas com dados não estruturados, mas para problemas mais tradicionais de aprendizado de máquina, como os envolvendo dados estruturados, a abordagem ainda divide opiniões. O principal contra-ponto é a dificuldade de interpretação destes modelos, o que pode ser desejável e até necessário em algumas aplicações. Para os demais problemas em que a maior precisão, por exemplo, é preferível em relação à interpretabilidade, o trabalho apresentado pode ser um caminho promissor.

Referências

WATSON, H. J.; WIXOM, B. H. The current state of business intelligence. *Computer*, IEEE, v. 40, n. 9, p. 96–99, 2007. Citado na página 16.

CODD, E. F. Providing olap (on-line analytical processing) to user-analysts: An it mandate. *http://www. arborsoft. com/papers/coddTOC. html*, 1993. Citado na página 16.

CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and olap technology. *ACM Sigmod record*, ACM, v. 26, n. 1, p. 65–74, 1997. Citado na página 17.

ELMASRI, R.; NAVATHE, S. *Sistemas de Bancos de Dados-Fundamentos e Aplicações, 5 edição*. [S.l.]: LTC, 2010. Citado na página 18.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery in databases. *AI magazine*, v. 17, n. 3, p. 37–37, 1996. Citado 2 vezes nas páginas 19 e 20.

MITCHELL, T. M. *Machine Learning*. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072. Citado 5 vezes nas páginas 20, 23, 24, 25 e 35.

LEE, H. D.; MONARD, M. C.; WU, F. C. Seleçao de atributos relevantes enao redundantes usando a dimensao fractal do conjunto de dados. In: *Anais do V Encontro Nacional de Inteligência, XXV Congresso da Sociedade Brasileira de Computaçao, Porto Alegre, RS*. [S.l.: s.n.], 2005. p. 444–453. Citado na página 21.

YEH, I.-C.; LIEN, C.-h. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Syst. Appl.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 36, n. 2, p. 2473–2480, mar. 2009. ISSN 0957-4174. Disponível em: <http://dx.doi.org/10.1016/j.eswa.2007.12.02>. Citado 5 vezes nas páginas 21, 50, 52, 60 e 63.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943. Citado na página 24.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 24.

MINSKY, M.; PAPERT, S. Perceptron: an introduction to computational geometry. *The MIT Press, Cambridge, expanded edition*, v. 19, n. 88, p. 2, 1969. Citado na página 25.

PENG, Y. *Tikz example – Kernel trick*. 2013. http://blog.pengyifan.com/tikz-example-kernel-trick/. Acessado em 27/11/2019. Citado na página 26.

ROBBINS, H.; MONRO, S. A stochastic approximation method. *The annals of mathematical statistics*, JSTOR, p. 400–407, 1951. Citado na página 29.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989. Citado na página 30.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org. Citado 2 vezes nas páginas 30 e 44.

OLAH, C. *Neural Networks, Manifolds, and Topology*. 2014. http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/#fnref2. Acessado em 27/11/2019. Citado na página 31.

CS231N. *Convolutional Neural Networks for Visual Recognition*. 2017. http://cs231n.github.io/neural-networks-1/. Acessado em 27/11/2019. Citado na página 33.

DSA, D. S. A. *Deep Learning Book*. [S.l.: s.n.], 2019. http://www.deeplearningbook.com.br/. Acessado em 27/11/2019. Citado na página 34.

SHARMA, A. *Understanding Activation Functions in Neural Networks*. 2017. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0. Acessado em 27/11/2019. Citado na página 34.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 1097–1105. Citado 2 vezes nas páginas 34 e 41.

PARKER, D. B. Learning logic technical report tr-47. *Center of Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA*, 1985. Citado na página 34.

LECUN, Y. Learning process in an asymmetric threshold network. In: *Disordered systems and biological organization*. [S.l.]: Springer, 1986. p. 233–240. Citado na página 34.

WERBOS, P. Beyond regression:"new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974. Citado na página 34.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986. Citado na página 34.

PASZKE, A.; GROSS, S.; CHINTALA, S.; CHANAN, G.; YANG, E.; DEVITO, Z.; LIN, Z.; DESMAISON, A.; ANTIGA, L.; LERER, A. Automatic differentiation in pytorch. 2017. Citado 3 vezes nas páginas 35, 44 e 70.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, Springer, v. 36, n. 4, p. 193–202, 1980. Citado na página 37.

DENKER, J. S.; GARDNER, W.; GRAF, H. P.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D.; BAIRD, H. S.; GUYON, I. Neural network recognizer for hand-written zip code digits. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1989. p. 323–331. Citado na página 37.

WATANABE, S. *Pattern recognition: human and mechanical*. [S.l.]: John Wiley & Sons, Inc., 1985. Citado na página 37.

HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, Wiley Online Library, v. 160, n. 1, p. 106–154, 1962. Citado na página 37.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989. Citado 2 vezes nas páginas 39 e 40.

LECUN, Y. Generalization and network design strategies. In: *Connectionism in perspective*. [S.l.]: Citeseer, 1989. v. 19. Citado na página 39.

DUMOULIN, V.; VISIN, F. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016. Citado na página 39.

VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: *Proceedings of the XXIX Conference on Graphics, Patterns and Images*. [S.l.: s.n.], 2016. p. 1–4. Citado 2 vezes nas páginas 40 e 52.

BAUM, E. B.; HAUSSLER, D. What size net gives valid generalization? In: *Advances in neural information processing systems*. [S.l.: s.n.], 1989. p. 81–90. Citado na página 40.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M. Imagenet large scale visual recognition challenge. *International journal of computer vision*, Springer, v. 115, n. 3, p. 211–252, 2015. Citado na página 41.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: SPRINGER. *European conference on computer vision*. [S.l.], 2014. p. 818–833. Citado na página 41.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 1–9. Citado 2 vezes nas páginas 41 e 52.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Citado na página 41.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778. Citado 2 vezes nas páginas 41 e 52.

HUANG, G.; LIU, Z.; MAATEN, L. V. D.; WEINBERGER, K. Q. Densely connected convolutional networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 4700–4708. Citado 2 vezes nas páginas 41 e 52.

RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. Citado 2 vezes nas páginas 42 e 44.

QIAN, N. On the momentum term in gradient descent learning algorithms. *Neural networks*, Elsevier, v. 12, n. 1, p. 145–151, 1999. Citado na página 42.

GOH, G. *Why momentum really works*. 2017. e6 p. https://distill.pub/2017/momentum/. Citado na página 42.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, v. 12, n. Jul, p. 2121–2159, 2011. Citado na página 43.

ZEILER, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. Citado na página 43.

HINTON, G.; SRIVASTAVA, N.; SWERSKY, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, v. 14, p. 8, 2012. Citado na página 43.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Citado na página 43.

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org>. Citado na página 44.

LIN, J.; CAMORIANO, R.; ROSASCO, L. Generalization properties and implicit regularization for multiple passes sgm. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2016. p. 2340–2348. Citado na página 45.

PRECHELT, L. Early stopping-but when? In: *Neural Networks: Tricks of the trade*. [S.l.]: Springer, 1998. p. 55–69. Citado na página 45.

YAO, Y.; ROSASCO, L.; CAPONNETTO, A. On early stopping in gradient descent learning. *Constructive Approximation*, Springer, v. 26, n. 2, p. 289–315, 2007. Citado na página 45.

ROUGHGARDEN, T.; VALIANT, G. Cs168: The modern algorithmic toolbox lecture# 6: Stochastic gradient descent and regularization. *Lecture Notes, Tim Roughgarden's Homepage, http://theory. stanford. edu/˜ tim/s16/l/l6. pdf*, v. 13, 2016. Citado na página 45.

KARPATHY, A. Cs231n convolutional neural networks for visual recognition. *Neural networks*, v. 1, 2016. Citado na página 46.

HINTON, G. E.; SRIVASTAVA, N.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. Citado na página 46.

SRIVASTAVA, N. Improving neural networks with dropout. *University of Toronto*, v. 182, p. 566, 2013. Citado na página 46.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014. Citado na página 47.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. Citado na página 47.

CIRESAN, D. C.; MEIER, U.; MASCI, J.; GAMBARDELLA, L. M.; SCHMIDHUBER, J. Flexible, high performance convolutional neural networks for image classification. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2011. Citado 2 vezes nas páginas 48 e 77.

ARAÚJO, E. A.; CARMONA, C. U. de M.; NETO, A. A. Aplicação de modelos credit scoring na análise da inadimplência de uma instituição de microcrédito. *Revista Ciências Administrativas*, Universidade de Fortaleza, v. 13, n. 1, p. 110–121, 2007. Citado na página 50.

KARCHER, C. *Redes Bayesianas aplicadas à análise do risco de crédito*. Tese (Doutorado) — Universidade de São Paulo, 2009. Citado 2 vezes nas páginas 50 e 52.

PINHEIRO, C. A. R. Redes neurais para prevenção de inadimplência em operadoras de telefonia. *Universidade Federal do Rio de Janeiro. Redes Neurais*, 2005. Citado na página 50.

MACHADO, E. J.; OLIVEIRA, R.; PEREIRA, A. C. M.; GERAIS, M.; HORIZONTE-BRAZIL, B. H.-B. B. Proposal and implementation of machine learning and deep learning models for stock markets. 2015. Citado na página 50.

TERNA, P.; D'ACUNTO, G.; CASELLE, M. A deep learning model to forecast financial time-series. 2016. Citado na página 50.

NELSON, D. M. Q. Uso de redes neurais recorrentes para previsão de séries temporais financeiras. UFMG, 2017. Citado na página 50.

BAO, D. A generalized model for financial time series representation and prediction. *Applied Intelligence*, Springer, v. 29, n. 1, p. 1–11, 2008. Citado na página 50.

FU, T.-c. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 24, n. 1, p. 164–181, 2011. Citado na página 51.

CHEN, Y.; DONG, G.; HAN, J.; WAH, B. W.; WANG, J. Multi-dimensional regression analysis of time-series data streams. In: ELSEVIER. *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. [S.l.], 2002. p. 323–334. Citado na página 51.

TAM, Y. J. *Datacube: Its implementation and application in olap mining*. [S.l.]: Simon Fraser University, 1998. Citado na página 51.

KORIKACHE, N.; YAHIA, A. *Coupling OLAP and Data Mining for Prediction*. [S.l.]: Academic Press, 2014. Citado na página 51.

LAMANI, A.; ERRAHA, B.; ELKYAL, M.; SAIR, A. Data mining techniques application for prediction in olap cube. *International Journal of Electrical and Computer Engineering*, IAES Institute of Advanced Engineering and Science, v. 9, n. 3, p. 2094, 2019. Citado na página 51.

JADAV, J. J.; PANCHAL, M. Association rule mining method on olap cube. *International Journal of Engineering Research and Applications (IJERA)*, v. 2, n. 2, p. 1147–1151, 2012. Citado na página 51.

KOHAVI, R.; SOMMERFIELD, D. Targeting business users with decision table classifiers. In: *KDD*. [S.l.: s.n.], 1998. p. 249–253. Citado na página 51.

SAIR, A.; ERRAHA, B.; ELKYAL, M.; LOUDCHER, S. Prediction in olap cube. *International Journal of Computer Science Issues (IJCSI)*, Citeseer, v. 9, n. 3, p. 449, 2012. Citado na página 51.

HAN, J. Olap mining: an integration of olap with data mining. In: *Data Mining and Reverse Engineering*. [S.l.]: Springer, 1998. p. 3–20. Citado na página 51.

FU, L. Construction of decision trees using data cube. In: *Enterprise Information Systems VII*. [S.l.]: Springer, 2007. p. 87–94. Citado na página 51.

HUA, Y. Decision tree algorithm based on olap multidimensional data set system. 2017. Citado na página 51.

PALANIAPPAN, S.; LING, C. Clinical decision support using olap with data mining. *International Journal of Computer Science and Network Security*, Citeseer, v. 8, n. 9, p. 290–296, 2008. Citado na página 51.

DRAKOPOULOS, G.; SPYROU, E.; MYLONAS, P. Tensor clustering: A review. *computing*, v. 22, p. 23, 2019. Citado na página 51.

THIRUMURUGANATHAN, S.; OUZZANI, M.; TANG, N. Explaining entity resolution predictions: Where are we and what needs to be done? 2019. Citado na página 52.

WEI, J.; HE, J.; CHEN, K.; ZHOU, Y.; TANG, Z. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, Elsevier, v. 69, p. 29–39, 2017. Citado na página 52.

THORPE, M.; GENNIP, Y. van. Deep limits of residual neural networks. *arXiv preprint arXiv:1810.11741*, 2018. Citado na página 52.

GRAVES, A.; FERNÁNDEZ, S.; SCHMIDHUBER, J. Multi-dimensional recurrent neural networks. In: SPRINGER. *International conference on artificial neural networks*. [S.l.], 2007. p. 549–558. Citado na página 52.

GRAVES, A.; SCHMIDHUBER, J. Offline handwriting recognition with multidimensional recurrent neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2009. p. 545–552. Citado na página 52.

AVATI, A.; JUNG, K.; HARMAN, S.; DOWNING, L.; NG, A.; SHAH, N. H. Improving palliative care with deep learning. *BMC medical informatics and decision making*, BioMed Central, v. 18, n. 4, p. 122, 2018. Citado na página 52.

NETO, J. M. M.; CASTRO, C. L.; BRAGA, A. P. Otimização da auc via redes neurais evolutivas para classificação de dados desbalanceados. XIII Brazilian Congress on Computational Intelligence, 2017. Citado na página 52.

YAN, L.; DODIER, R. H.; MOZER, M.; WOLNIEWICZ, R. H. Optimizing classifier performance via an approximation to the wilcoxon-mann-whitney statistic. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. [S.l.: s.n.], 2003. p. 848–855. Citado na página 52.

EBAN, E. E.; SCHAIN, M.; MACKEY, A.; GORDON, A.; SAUROUS, R. A.; ELIDAN, G. Scalable learning of non-decomposable objectives. *arXiv preprint arXiv:1608.04802*, 2016. Citado na página 52.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: MONTREAL, CANADA. *Ijcai*. [S.l.], 1995. v. 14, n. 2, p. 1137–1145. Citado na página 61.

TANWAR, S. *How to get a grip on Cross Validations*. 2018. https://www.freecodecamp.org/news/how-to-get-a-grip-on-cross-validations-bb0ba779e21c/. Acessado em 27/11/2019. Citado na página 62.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado na página 69.