Redes neurais aplicadas no reconhecimento de símbolos matemáticos manuscritos online

Caíque Quaresma Silva

Trabalho de conclusão de curso apresentado
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Bacharel em Matemática Aplicada

Curso: Matemática aplicada

Habilitação em métodos matemáticos

Orientador: Prof.ª Nina S. T. Hirata

São Paulo, Fevereiro de 2019

Agradecimentos

Agradeço à minha orientadora , Prof.ª Nina S. T. Hirata, pela disponibilidade ao longo do ano.

A Frank Julca-Aguilar por disponibilizar os dados de seu trabalho.

À minha mãe, Maria da Conceição, pelo suporte ao longo de minha graduação.

E agradeço aos meus amigos do IME-USP por tornarem meus dias mais felizes.

Resumo

Devido aos avanços tecnológicos, o reconhecimento de expressões matemáticas manuscritas *online* tem sido uma destacada área de pesquisas. Uma parte importante que afeta o processo de reconhecimento como um todo é a etapa de reconhecimento de símbolos. Inspirado no funcionamento do sistema nervoso, redes neurais são um dos modelos mais utilizados na área de aprendizado de máquina. Neste trabalho, apresentamos um resumo sobre a teoria de redes neurais. Em seguida, treinamos modelos de redes neurais do tipo *MLP (Multilayer Perceptron)* e *CNN (Convolutional Neural Network)*, usados como método de reconhecimento e classificação de símbolos matemáticos manuscritos *online*.

Palavras-chave: Reconhecimento de símbolos matemáticos, aprendizado de máquina, redes neurais, *MLP*, *CNN*.

Abstract

Due to technological advances, the recognition of online handwritten mathematical expressions has been a prominent research area. An important step that affects the whole recognition process is the symbols recognition . Inspired by the the nervous system functioning , neural networks are the most used models in machine learning. In this monography, we present a summary of neural networks theory . Next, we train the neural networks models MLP (Multilayer Perceptron) and CNN (Convolutional Neural Network), used as a method of recognition and classification of online handwritten mathematical symbols.

Keywords: mathematical symbol recognition, machine learning, neural networks, MLP, CNN.

Sumário

Capítulo 1

Introdução

1.1 Reconhecimento de expressões matemáticas

Nos últimos anos, tem-se observado grandes avanços em tecnologias digitais, como *tablets*, lousas interativas e *smartphones*, que através de suas telas *touchscreen* facilitam a comunicação humano-máquina. Com o advento dessas tecnologias, o reconhecimento de expressões matemáticas manuscritas *online* tem sido uma destacada área de pesquisa, motivando até a existência de competições internacionais como o *CROHME* (*Competition on recognition of online handwritten mathematical expressions*)[1].

Segundo Chan e Yeung [2], uma das grandes dificuldades em passar expressões matemáticas para um computador é o fato de que tais expressões são compostas de caracteres diversos: números, letras de alfabetos diferentes, operadores, palavras representando funções. Essa diversidade de símbolos torna necessário o uso de métodos de entrada, como teclados especiais, ou linguagens como o sistema LaTeX, exigindo, a priori, treino e prática em tais métodos para alcançar um bom resultado.

Uma solução que não exige conhecimento prévio, é simplesmente escrever diretamente as expressões usando dispositivos digitais sensíveis ao toque e deixar que um computador realize o processo de reconhecimento. Com isso, no contexto de reconhecimento de expressões matemáticas o termo online significa que, com o uso desses dispositivos, expressões são vistas como uma sequência de traços, e estes, uma sequência de pontos, associando informações temporais ao processo de escrita, ou seja, a ordem com que os traços foram escritos.Um ponto é representado por suas coordenadas bi-dimensionais na tela do dispositivo, enquanto um traço é a sequência dos pontos gerados pelo primeiro instante de contanto com a superfície do dispositivo até o fim do contato. Em contrapartida, o termo *offline* indica que as expressões não contém informações temporais de escrita, mas sim, da sua estrutura de maneira geral, como por exemplo, a imagem de uma expressão escrita.

O processo de reconhecimento normalmente ocorre em 3 etapas fundamentais [3]:

- **Segmentação**: Com o uso de técnicas apropriadas, as expressões, vistas como um conjunto de traços, são particionadas em subconjuntos de traços representando os símbolos que compõem sua estrutura;
- **Reconhecimento**: Aplica-se um modelo de classificação construído com técnicas de *machine learning* em um subconjunto de traços a fim de definir o símbolo matemático que está representado por esses traços, atribuindo um grau de confiança alto para o símbolo correto e baixo para os demais símbolos;

- **Análise estrutural**: Identifica-se relações espaciais, como acima, abaixo, subscrito,sobrescrito, e lógicas entre os símbolos, definindo o significado da expressão.

Sendo a matemática uma linguagem universal, nada mais natural e conveniente do que usar esta mesma linguagem e as tecnologias disponíveis, facilitando a transcrição de expressões matemáticas para um computador.

1.2 Objetivos

Neste trabalho, estamos interessados em construir modelos de classificação para a etapa de reconhecimento de símbolos de expressões matemáticas manuscritas *online* que apresentem bom desempenho,visto que esta etapa tem impacto direto no problema geral de reconhecimento de expressões. A base de dados utilizada será os conjuntos de dados do trabalho de Julca-Aguilar [4] previamente processados divididos em conjuntos de treinamento, validação, e teste. Os dados originais pertencem ao *dataset* público de expressões matemáticas manuscritas *online* disponibilizado pelo *CROHME*. [1]. Como método de classificação, serão utilizados modelos de redes neurais do tipo MLP (*Multilayer Perceptron*) e CNN (*Convolutional Neural Network*) [5]. Para a construção e manipulação das redes, utilizaremos a *API (Application Programming Interface)* para redes neurais *Keras* [6], usando como suporte o sistema para *machine learning Tensorflow* [7], em *python*.

1.3 Organização

No capítulo 2, apresentamos um resumo de conceitos importantes utilizados neste trabalho: *machine learning*, redes neurais, gradiente descendente, *backpropagation*, generalização. No capítulo 3, temos o problema prático de classificação, apresentando os conjuntos de dados, métricas, modelos, resultados e discussão. No capítulo 4, temos a conclusão do trabalho. No apêndice A, apresentamos alguns exemplos de arquiteturas de redes neurais. No apêndice B, temos uma tabela com as classes, símbolos e amostras em cada conjunto de dados.

Capítulo 2

Conceitos

2.1 *Machine learning*

Com o avanço da *internet*, mais e mais informações das mais diversas origens são coletadas e armazenadas todos os dias. Devido ao grande volume de informações, tarefas como organização e análise de dados se tornaram complexas demais para serem executadas de maneira manual por uma pessoa. Essa necessidade, aliada ao avanço das tecnologias computacionais, motivou o uso de métodos automáticos de análise de dados que constituem o chamado *machine learning*, ou aprendizado de máquina.

Nas palavras de Arthur Samuel, famoso cientista da computação precursor da inteligência artificial, creditado como inventor do termo *"machine learning"*, o aprendizado de máquina nada mais é do que um campo de estudo que dá aos computadores a habilidade de aprender sem terem sido programados para tal. Por aprender, Mitchell [8] afirma: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T,as measured by P, improves with experience E"*.

Assim, *machine learning* pode ser visto como um conjunto de técnicas, ou algoritmos, que permitem que um programa de computador, de maneira automatizada, extraia informações relevantes, identifique padrões e encontre relações em grandes quantidades de dados de uma maneira que um ser humano não conseguiria, simplesmente treinando e aprendendo com os dados [9] . Dentre os diversos tipos de dados, podemos citar: imagem, áudio, texto, informações bancárias, informações sobre comportamento de consumo, entre outros.

Goodfellow et al. [10] dividem o aprendizado de máquina em 3 principais paradigmas:

- **Aprendizado supervisionado**: Também conhecido como aprendizado com um professor, é um dos métodos mais utilizados em *machine learning*. Neste paradigma, os dados para treinamento são rotulados, ou seja, são vistos como pares (x_i, y_i), onde x_i é um dado de entrada, e y_i é o resultado desejado, permitindo que um modelo aprenda através de exemplos. Exemplos de algoritmos de aprendizado supervisionado são: árvores de decisão, SVM (*Support Vector Machine*), classificadores Naive Bayes, regressão linear, regressão logística.

- **Aprendizado por reforço**:Semelhante ao processo supervisionado, neste método, ao invés de receber os dados e suas saídas desejadas,o modelo recebe os dados com uma saída qualquer mais uma medida que informa o quão boa é essa saída.Um programa que aprende a jogar xadrez é um exemplo de problema que explora esse paradigma.

- **Aprendizado não supervisionado**:Diferente dos paradigmas anteriores, neste método, o modelo recebe somente os dados de treinamento sem informação alguma dos resultados desejados, ficando encarregado de extrair informações ou aprender padrões por conta própria somente analisando a estrutura dos dados. Exemplos de algoritmos de aprendizado não supervisionado são: algoritmos de agrupamento (*clustering*), SVD (*Singular Value Decomposition*), PCA (*Principal Component Analysis)*), ICA (*Independent Component Analysis*), KNN (*K Nearest Neighbor*).

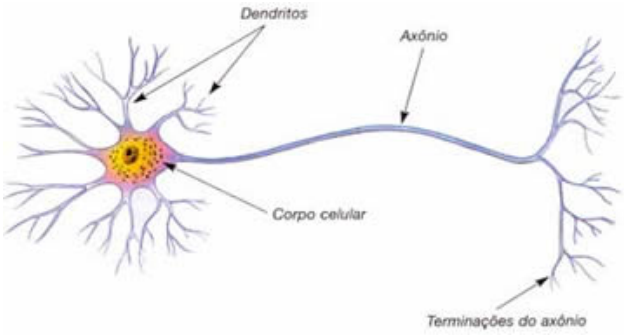
Com a diversidade de algoritmos de aprendizado, as técnicas de *machine learning* são extremamente versáteis, podendo ser empregadas na resolução de uma série de problemas de naturezas diferentes, como: classificação, regressão, transcrição, tradução, detecção de anomalias, predição, redução de ruídos, estimação de função de probabilidade, sistemas de recomendação, visão computacional, etc.

Devido sua grande maleabilidade, podendo ser aplicado na resolução de vários desses problemas, um dos modelos mais utilizados é o modelo de redes neurais.

2.2 Redes neurais

O cérebro é sem dúvida uma das estruturas biológicas mais fascinantes da natureza. No ser humano , representa cerca de 2% de sua massa corporal e tem uma alta demanda energética, fazendo com que a quantidade de sangue bombeada para esta região seja de aproximadamente 25%. Suas principais células estruturais responsáveis por realizar o processamento de informação, os neurônios, são formadas por 3 partes principais:

Figura 2.1: *Estrutura básica de um neurônio.*



Fonte: Só Biologia - Fisiologia Animal.[1]

- **Dendritos**: Filamentos responsáveis por receber e transportar estímulos vindos do ambiente ou de outras células para o corpo celular.
- **Corpo celular**:Também conhecido como soma, é a maior das 3 estruturas. Faz o processamento das informações recebidas pelos dendritos.
- **Axônio**: Transmite os impulsos nervosos gerados pelo corpo celular para outras células através de suas terminações ramificadas.

[1]Disponível em: <https://www.sobiologia.com.br/conteudos/FisiologiaAnimal/nervoso2.php>. Acesso em nov. 2018.

Em resumo, ao receber estímulos na forma de sinais elétricos, o corpo celular processa o conjunto de sinais como um todo e caso o resultado ultrapasse um determinado valor de limiar, o neurônio dispara um impulso nervoso indicando que reagiu aos sinais de entrada, que segue adiante até as terminações do axônio onde, através das sinapses e neurotransmissores, é transmitido para outros neurônios ou outras células [11].

Considerando sua capacidade de processamento, neurônios processam informações na ordem de $10^{-3}s$, enquanto circuitos típicos feitos de silício trabalham na ordem de $10^{-9}s$. No entanto, devido a grande quantidade de neurônios densamente conectados entre si formando grandes redes, cerca de 10 bilhões de neurônios e 60 trilhões de sinapses, o cérebro humano se torna um sistema de processamento altamente complexo e eficiente, atuando de forma não-linear e em paralelo, desempenhando tarefas como reconhecimento de imagens, áudio, controle motor, etc., melhor do que qualquer computador mais moderno. Tudo isso é possível graças a grande capacidade do cérebro humano em aprender e se adaptar com a experiência.

Com o propósito de criar um sistema que operasse de forma parecida ao cérebro humano, foram criados os modelos de redes neurais artificiais, ou simplesmente redes neurais [12], cuja semente ideológica remete ao trabalho de McCulloch e Pitts [13], e posteriormente, o famoso modelo de neurônio artificial de Rosenblatt , *perceptron* [14].

Haykin [5] define uma rede neural da seguinte forma: "Uma rede neural é um processador maciçamente paralelamente distribuído construído de unidades de processamento simples que tem propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso. Ela se assemelha ao cérebro em dois aspectos: o conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem; forças de conexão entre neurônios, conhecidas como pesos sinápticos, são usadas para armazenar conhecimento adquirido.". Suas unidades de processamento são conhecidas como neurônios artificiais, e agem de forma a simular de maneira mais simples o funcionamento do neurônio biológico. Segundo Haykin [5], temos as seguintes vantagens no uso de redes neurais:

- **Não-linearidade**:Alguns problemas, como sinal de voz, são melhor resolvidos com o uso de modelos não-lineares.Em uma rede neural, seus neurônios artificias podem ser tanto lineares quanto não-lineares. Em particular, para neurônios não-lineares, essa propriedade é distribuída por toda a rede, gerando um modelo não-linear.

- **Mapeamento de entrada-saída**:Redes neurais possuem bom desempenho quando usadas como modelos de aprendizado supervisionado. De posse de dados de treinamento, os pesos sinápticos da rede são modificados pelo algoritmo de aprendizado com o objetivo de minimizar a diferença entre a saída esperada e a saída real criando um mapeamento entre os dados de entrada e saída, sendo um ótimo modelo para problemas de classificação.

- **Adaptabilidade**:Devido sua estrutura, redes neurais possuem grande capacidade de adaptação, modificando facilmente seus pesos sinápticos a mudanças no ambiente em que atuam. Podem ser projetadas ainda para adaptar seus pesos em tempo real, quando modeladas em ambientes com constantes mudanças.

- **Resposta a evidências**:Em problemas de classificação de padrões, uma rede neural pode ser modelada tanto para indicar a qual classe pertence determinado padrão, quanto para indicar o grau de confiança de sua escolha, o que pode ser usado para eliminar padrões ambíguos.

- **Informação contextual**:Em uma rede neural, seus neurônios são fortemente influenciados entre si, assim, informações contextuais são tratadas naturalmente pela rede.

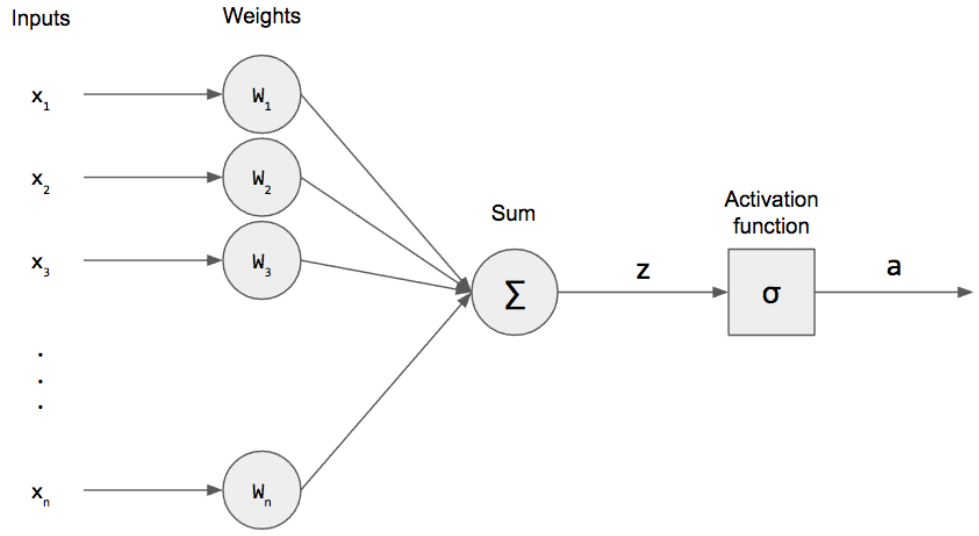
- **Uniformidade de análise e projeto**:Devido a sua estrutura, redes neurais podem ser consideradas processadores de informação universais. Seus neurônios artificiais são estruturas presentes em qualquer modelo de rede neural, tornando possível o compartilhamento de técnicas e teorias entre modelos com propósitos diferentes.

- **Analogia neurobiológica**:Inspirada na estrutura eficiente do cérebro humano, redes neurais artificiais são vistas por áreas como a neurobiologia como um modo de pesquisar e interpretar fenômenos neurobiológicos. O contrário é verdadeiro: engenheiros se voltam para a neurobiologia em busca de formas de resolver problemas que as técnicas usuais não resolvem.

2.2.1 *Perceptron*

Inspirado pelos trabalhos de McCulloch e Pitts na década de 40 [13], o psicólogo e neurobiologista Frank Rosenblatt, como parte de seus estudos sobre a visão, desenvolveu na década de 50 um modelo que simula de forma simplificada o funcionamento de um neurônio: o *perceptron* [14]. Em sua palavras,"um modelo probabilístico para o armazenamento de informação e organização no cérebro"[14].

Figura 2.2: *Perceptron.*



Fonte: Medium.[2]

No modelo de neurônio artificial ou neurônio matemático, tem-se as seguintes estruturas:

- $x \in \mathbb{R}^n$, são os sinais recebidos pelo *perceptron.*

- $w \in \mathbb{R}^n$, são os pesos sinápticos que indicam a importância de cada sinal para o *perceptron* . Valores grandes e positivos indicam maior relevância, valores pequenos e negativos, menor relevância . São o equivalente artificial à memória ou conhecimento do modelo.

[2]Disponível em: <https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1>. Acesso em out. 2018.

- $b \in \mathbb{R}$, é um valor de viés que indica a facilidade do neurônio para "disparar". Valores grandes e positivos indicam maior facilidade, valores pequenos e negativos, menor facilidade. O valor -b pode ser visto como o valor de limiar do neurônio biológico.

- Σ, é um combinador linear responsável por unificar os sinais de entrada em um único valor z, com

$$\begin{aligned} z &= \Sigma(x) \\ &= \sum_{i=1}^{n} x_i w_i + b. \end{aligned} \tag{2.1}$$

- σ, chamada função de ativação, é uma função de $\mathbb{R}$ em $\mathbb{R}$ não constante, geralmente limitada, que retorna um valor de disparo a , com

$$a = \sigma(z). \tag{2.2}$$

De modo geral, dado $w \in \mathbb{R}^n$ e $b \in \mathbb{R}$, o modelo de neurônio artificial pode ser visto como a função $h : \mathbb{R}^n \to \mathbb{R}$,

$$h(x|w, b) = \sigma(\langle x, w \rangle + b). \tag{2.3}$$

A partir do *perceptron* foram desenvolvidas as redes neurais artificiais, que assim como o cérebro, são formadas por um grande número de neurônios artificiais conectados entre sim em uma variedade de configurações e aplicações diversas. Quanto a forma com que os neurônios artificiais se conectam, redes neurais podem ser classificadas em 3 categorias [15]:

- **Redes neurais *feedforward***: Tipo mais comum de rede neural, é formada por camadas de neurônios artificiais de tal forma que a informação entra por uma camada de entrada e é propagada pela rede através de camadas intermediárias sempre "para frente", em direção a camada de saída, não tendo ciclos. Exemplos: *perceptron*, *MLP* (*Multilayer Perceptron*), *CNN* (*Convolutional Neural Network*).

- **Redes neurais recorrentes**: Possui ciclos em sua dinâmica de funcionamento, ou seja, a saída de um neurônio pode se conectar com ele mesmo ou um neurônio anterior.Usado normalmente para modelar dados de natureza sequencial, como séries temporais, som e linguagem natural. Exemplos: *LSTM* (*Long Short-Term Memory*), *GRN* (*Gated Recurrent Unit*).

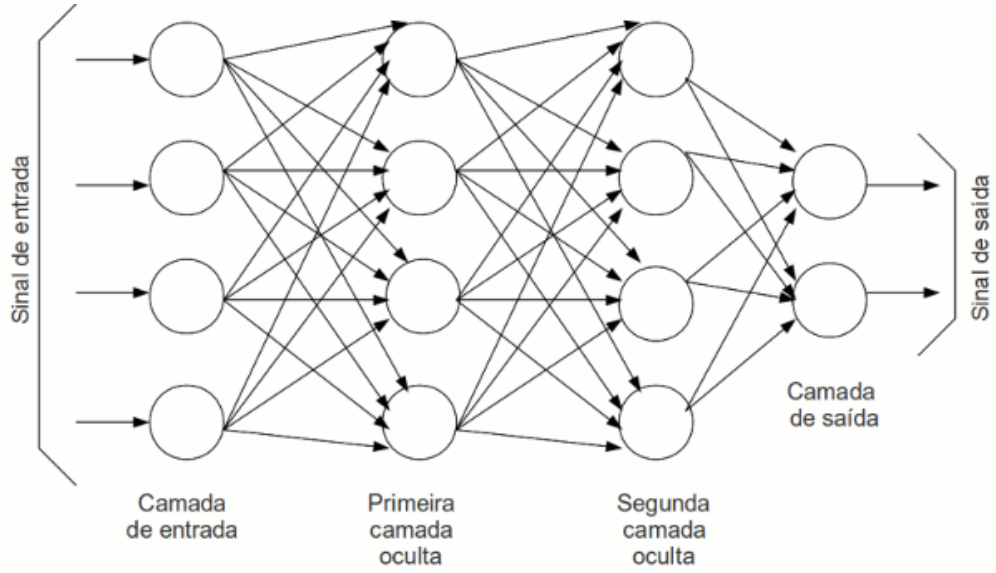
- **Redes neurais simétricas**: Tipos especiais de redes recorrentes, os neurônios artificiais são conectados de maneira simétrica entre si compartilhando os mesmos pesos sinápticos em relação a sua simetria. Exemplos: redes Hopfield, máquinas de Boltzmann.

Para visualizar as arquiteturas dos modelos citados, entre outros, veja apêndice A.
Neste trabalho, estamos interessados nos modelos de redes neurais do tipo *MLP* e *CNN*.

2.2.2 MLP

No modelo *MLP*, a rede neural é construída com camadas de neurônios artificiais dispostas em sequência: uma camada de entrada , onde entram as informações; uma sequência de camadas ocultas, onde ocorrem uma série de processamentos capazes de abstrair em vários níveis as informações recebidas; uma camada de saída, com o resultado final do processamento. Neste modelo, os neurônios de uma camada se conectam a todos os neurônios da camada seguinte agindo como dados de entrada daquela camada, evidenciando o caráter "*feedforward*"do modelo. Assim como o *perceptron*, podemos enxergar um modelo *MLP* como uma função.

Figura 2.3: *Arquitetura de um modelo* MLP *com duas camadas ocultas.*



Fonte: Alexandre Volpi.[3]

Seja $\mathbb{R}^n$, para algum $n > 1$, o domínio dos sinais de entrada. Considere um modelo MLP com número de camadas L e arquitetura

$$c_1 - c_2 - \cdots - c_{L-1} - c_L, \tag{2.4}$$

com c_l o número de neurônios na camada $l = 1, \ldots, L$. Por definição, a primeira camada corresponde aos dados de entrada, assim, $c_1 = n$. Seja σ_i, $i = 1, \ldots, L$, funções de ativação vetorizadas. Seja a^{l-1} o vetor das saídas dos neurônios da camada $l-1$, com a_j^{l-1} saída do neurônio j. Considere w_{ij}^l o peso sináptico da conexão entre o neurônio j, na camada $l-1$, e i, na camada l. Considere b_i^l o viés do neurônio i. Calculando a saída do neurônio i na camada l, temos:

$$\begin{aligned} z_i^l &= \sum_{j=1}^{c_{l-1}} a_j^{l-1} w_{ij}^l + b_i^l \\ a_i^l &= \sigma_l(z_i^l). \end{aligned} \tag{2.5}$$

Dessa forma, para cada camada l pode-se definir uma matriz de pesos sinápticos e um vetor

[3]Disponível em: <https://alexandrevolpi.wordpress.com/2015/10/02/ferramentas-de-python-para-aprendizado-de-maquina/>. Acesso em out. 2018.

de vieses

$$\begin{aligned} w^l &= [w^l_{ij}]_{c_l \times c_{l-1}} \\ b^l &= [b^l_i]_{c_l \times 1}. \end{aligned} \tag{2.6}$$

Logo, a equação (2.5) assume a forma vetorial:

$$\begin{aligned} z^l &= w^l a^{l-1} + b^l \\ a^l &= \sigma(z^l). \end{aligned} \tag{2.7}$$

Ao todo, existem

$$\begin{aligned} d &= nc_2 + c_2c_3 + \cdots + c_{L-1}c_L \\ &= \prod_{l=2}^{L} c_{l-1}c_l \end{aligned} \tag{2.8}$$

pesos sinápticos, e

$$m = \sum_{l=2}^{L} c_l \tag{2.9}$$

vieses. Por fim, dado $w \in \mathbb{R}^d$ e $b \in \mathbb{R}^m$, pode-se definir o modelo MLP como uma função $h : \mathbb{R}^n \to \mathbb{R}^{c_L}$. Em um modelo com 2 camadas ocultas, por exemplo:

$$h(x|w, b) = \sigma(w^4\sigma(w^3\sigma(w^2x + b^2) + b^3) + b^4). \tag{2.10}$$

Um importante resultado de 1989 para modelos *MLP* é o chamado *Teorema da aproximação universal* [16].

Teorema 2.1. *Seja $\sigma : \mathbb{R} \to \mathbb{R}$ função contínua, limitada, não-constante e monotonicamente crescente. Seja $I_n = [0, 1]^n$ o hipercubo n-dimensional e $C(I_n)$ o espaço das funções contínuas em I_n. Então, dado $f \in C(I_n)$ independente de σ e $\epsilon > 0$, existe $N \in \mathbb{N}$, $w_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$ e $\alpha_i \in \mathbb{R}$, onde $i = 1, \ldots, N$, tais que*

$$\begin{aligned} F(x) &= \sum_{i=1}^{N} \alpha_i\sigma(w_i.x + b_i), \\ |F(x) - f(x)| &< \epsilon, \end{aligned} \tag{2.11}$$

$\forall x \in I_n$.

Demonstração. Veja Cybenko [16] □

Fazendo um paralelo com as redes neurais, o teorema estabelece que para qualquer função f contínua definida em I_n existe uma rede neural *MLP* de arquitetura $n-N-1$ com matrizes de pesos sinápticos

$$\begin{aligned} w^2 &= [w_{ij}]_{N \times n} \\ w^3 &= [\alpha_i]_{1 \times N} \end{aligned} \tag{2.12}$$

e vetor de vieses

$$b^2 = [b_i]_{N\times 1} \tag{2.13}$$

que aproxima f com certa precisão. O teorema ainda pode ser generalizado para mais de uma camada e saídas vetoriais. Exemplos de funções de satisfazem o teorema são:

- Sigmoide:

$$\sigma(z) = \frac{1}{1+e^{-z}}. \tag{2.14}$$

- Tangente hiperbólica:

$$\begin{aligned}\sigma(z) &= \frac{2}{1+e^{-2z}} - 1 \\ &= 2Sigmoide(2z) - 1\end{aligned} \tag{2.15}$$

- Softsign:

$$\sigma(z) = \frac{z}{1+|z|}. \tag{2.16}$$

Resultados recentes mostraram que certas funções não limitadas, e portanto não satisfazendo o teorema da aproximação universal, também podem ser usadas como aproximadores universais [17]. São exemplos:

- $ReLU$(*Rectified Linear Unit*):

$$\sigma(z) = \max\{0, z\} \tag{2.17}$$

- *Softlpus*:

$$\sigma(z) = ln(e^z + 1) \tag{2.18}$$

- ELU(*Exponencial Linear Unit*):Para $\alpha \in \mathbb{R}$

$$\sigma(z) = \begin{cases} \alpha(e^z - 1) & \text{se } z < 0 \\ z & \text{se } z \geq 0 \end{cases} \tag{2.19}$$

2.2.3 CNN

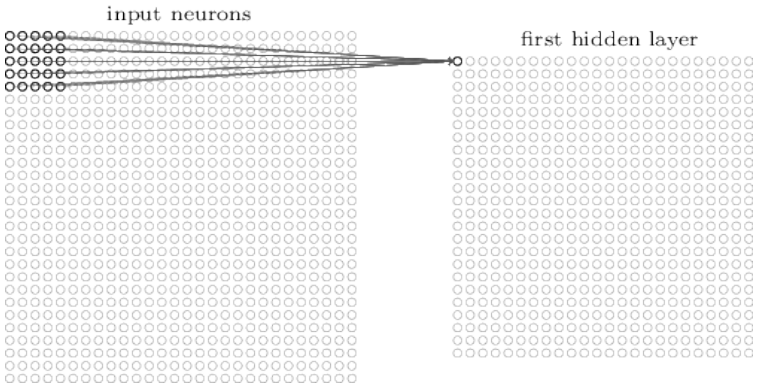
Inspirado no funcionamento do sistema visual dos mamíferos, o modelo CNN (*Convolutional Neural Network*) tem suas origens nos anos 70, mas só ganhou visibilidade com a publicação de LeCun et al.[18] com o modelo convolucional *Lenet5* aplicado no reconhecimento de dígitos manuscritos. Desenvolvida para a resolução de problemas envolvendo dados do tipo imagem, representados por sua matriz de *pixels*, sua arquitetura é uma modificação do modelo *MLP*, formada por um bloco convolucional seguido de uma rede *MLP*. Haykin[5] destaca 3 características principais presentes no bloco convolucional:

- *Extração de características*: Diferente de um modelo *MLP*, cada neurônio recebe sinais de entrada de um campo receptivo local na camada anterior, possibilitando a extração

de características locais. Dessa forma, a posição exata de uma característica se torna irrelevante desde que sua posição em relação a outras características seja aproximadamente preservada.

- *Mapeamento de características*: Cada camada da rede é formada por *feature maps*, onde os neurônios individuais compartilham os mesmos parâmetros entre si. Cada *feature map* é construído através de uma operação de convolução entre uma camada anterior e um *kernel*, ou filtro, de pesos sinápticos. Essa operação faz com que redes convolucionais seja invariantes por translação, além de reduzir o número de parâmetros livres.

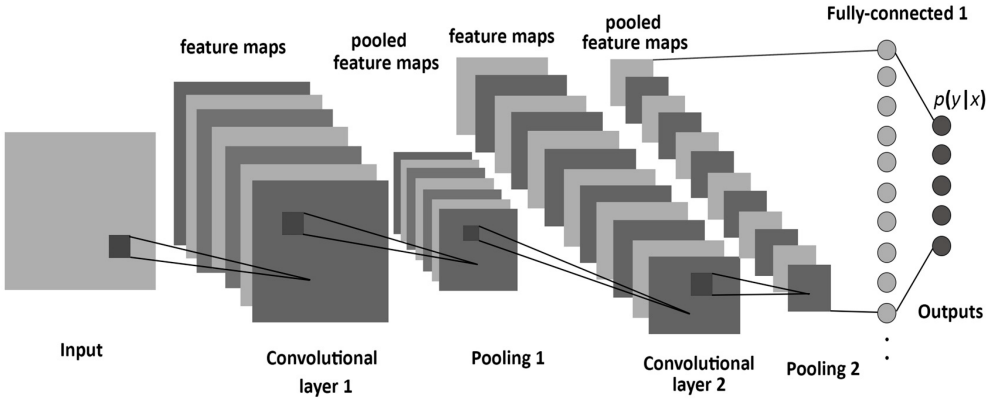
Figura 2.4: *Feature map com filtro* 5×5.



Fonte: Nielsen [12].

- *Pooling*: Cada camada de convolução, em geral, é seguida de uma camada de *pooling* ou subamostragem, responsável por condensar a informação de um *feature map* reduzindo sua dimensão. Também ajuda a reduzir a sensibilidade a deslocamentos.

Figura 2.5: *Arquitetura de um modelo convolucional.*



Fonte: Demi.Ledge - Convolutional Neural Networks.[4]

[4]Disponível em: <http://www.demiledge.com/artificialIntelligence/CNN.php>. Acesso em out. 2018.

Definição 2.1. *Dada uma matriz $I \in \mathbb{R}^{n\times m}$ e um filtro $K \in \mathbb{R}^{k\times k}$, o produto de convolução $I * K$ é uma matriz de dimensões $(n-k+1)\times(m-k+1)$, com*

$$(I * K)_{ij} = \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} I_{(i-a)(j-b)}K_{ab}. \tag{2.20}$$

De um ponto de vista matemático, as redes neurais convolucionais têm a seguinte dinâmica [19]:

- Sua camada de entrada recebe uma imagem como uma matriz de *pixels* $M \in \mathbb{R}^{h\times w\times c}$, onde c é o número de canais, e calcula-se os primeiros *feature maps*.

- Cada camada convolucional l recebe de uma camada anterior um conjunto de *feature maps* representados por matrizes $a^{l-1} = \{a^{(l-1,1)}, \ldots, a^{(l-1,p)}\}$ de dimensões $n \times m$, para algum p .

- Cada camada de convolução possui uma função de ativação σ, um conjunto de filtros $w^l_{i,j} \in \mathbb{R}^{d\times d}$ e um conjunto de vieses b^l_j, onde $i = 1, \ldots, p$ e $j = 1, \ldots, q$, sendo q o número de *feature maps* gerados pela camada atual.

- Recebido os *feature maps* da camada anterior, calcula-se as matrizes de dimensões $(n-d+1)\times(m-d+1)$

$$z^{(l,k)} = \sum_{i=1}^{p} a^{(l-1,i)} * w^l_{i,k} + b^l_k, k = 1, \ldots, q \tag{2.21}$$

  e os *feature maps*

$$a^{(l,k)} = \sigma(z^{(l,k)}), k = 1, \ldots, q \tag{2.22}$$

  que são passados adiante.

- Em uma camada de *pooling*, escolhido um tamanho de área de ação, normalmente 2×2, aplica-se uma função nesta área dos *feature maps* a fim de condensar informação. As funções mais usadas são o máximo entre os valores, método conhecido como *maxpooling*, e a média dos valores, conhecido como *average pooling*.

- Ao final de um bloco de convolução, tem-se um conjunto de matrizes representando as características extraídas da imagem, que são vetorizadas, concatenadas, e enfim, transmitidas para a rede *MLP*.

2.2.4 Treinamento

O teorema da aproximação universal, sob algumas condições, garante a existência de uma rede neural capaz de aproximar uma função, no entanto, não provê meios para encontrar tal rede. Essa garantia de existência fez com que redes neurais dos tipos *MLP* e *CNN* se tornassem boas opções para a resolução de problemas de *machine learning* do tipo supervisionado, em particular, problemas de classificação.

Em um problema de classificação, dado um conjunto de treinamento com dados rotulados com suas respectivas classes $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, onde $y_i \in \{1, 2, \ldots, k\}$ para algum k, supõe-se que exista uma função $f : \mathcal{X} \to \{1, 2, \ldots, k\}$, com $\mathcal{X}$ o domínio dos dados de entrada, tal que $f(x_i) = y_i \ \forall i = 1, \ldots, n$. Usando um algoritmo de aprendizado, quer-se encontrar uma função $h(x|\theta)$, que depende de parâmetros θ, que aproxime f em $\mathcal{X}$. Na prática, o usual é encontrar uma aproximação não para o mapeamento direto, mas sim, para a distribuição de probabilidade dos dados em relação ao total de classes. Por exemplo, se x_1 é de classe 1, $y_1 = [1, 0, \ldots, 0]^t$ com $y_1 \in \mathbb{R}^k$, quer-se aproximação $h : \mathcal{X} \to [0, 1]^k$ tal que $\sum_{i=1}^{k} h(x_1|\theta)_i = 1$ e tem como maior valor $h(x_1|\theta)_1$. Essa abordagem é também conhecida como *one-hot encoding*. No contexto de redes neurais, isso pode ser feito aplicando na camada de saída a função *softmax*. Para $z \in \mathbb{R}^k$, é definida por

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, i = 1, \ldots, k. \tag{2.23}$$

Um dos métodos de aprendizado mais aplicados em modelos *MLP* e *CNN* para problemas de classificação, é o método de correção de erros, onde os modelos "aprendem errando".

Definido um modelo $h(x|w, b)$ como em (2.2.2) ou (2.2.3), os parâmetros iniciais, pesos sinápticos e vieses, são inicializados aleatoriamente como ruído branco de média 0. Para cada dado de treinamento x_i, calcula-se uma medida de custo indicando a discrepância entre o valor estimado $a^L = h(x_i|w, b)$ e o valor exato y_i. Para $u, v \in \mathbb{R}^k$, alguns exemplos de funções de custo são:

- Erro quadrático:

$$C(u, v) = \frac{1}{2k} \sum_{i=1}^{k} (u_i - v_i)^2 \tag{2.24}$$

- Erro absoluto:

$$C(u, v) = \frac{1}{k} \sum_{i=1}^{k} |u_i - v_i| \tag{2.25}$$

- Entropia cruzada:

$$C(u, v) = -\sum_{i=1}^{k} u_i ln(v_i) + (1 - u_i) ln(1 - v_i) \tag{2.26}$$

Em problemas de classificação cuja saída é uma distribuição de probabilidade, o usual é adotar a função de entropia cruzada, usada para medir a diferença entre duas distribuições.

Em seguida, calcula-se uma medida de erro global para o modelo, dada por:

$$L(x, y|w, b) = \frac{1}{n}\sum_{i=1}^{n} C(h(x_i|w, b), y_i). \tag{2.27}$$

Na equação (2.27), os valores x_i e y_i estão determinados, assim, pode-se expressar o erro e o modelo como função dos parâmetros, ou seja:

$$L(w, b|x, y) = \frac{1}{n}\sum_{i=1}^{n} C(h(w, b|x_i), y_i). \tag{2.28}$$

Portanto, treinar uma rede neural pelo método de correção de erro consiste em encontrar parâmetros w e b que minimizem sua função de erro. Sob a suposição de derivabilidade das funções de ativação σ, uma abordagem natural seria encontrar diretamente parâmetros ótimos, no entanto, como visto em (2.10), redes neurais *feedforward*, além da grande quantidade de parâmetros, são estruturadas como composições de funções, tornando a abordagem analítica impraticável. O que é usado na maioria dos casos, é o método numérico de otimização conhecido como *gradiente descendente*.

2.3 Gradiente descendente

Sabe-se do Cálculo que o gradiente de uma função diferenciável aponta para sua direção de maior crescimento e é perpendicular a suas curvas de nível. Usando a função de erro com parâmetros $\theta = (w, b)$ o método do gradiente descendente permite a busca por pontos mínimos de uma maneira computacional usando a regra de iteração

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L\left(\theta|x, y\right), \tag{2.29}$$

onde η, conhecido como taxa de aprendizado, define o tamanho do passo em direção ao ponto de mínimo.

Na prática, aplicar o método usando todo o conjunto de treinamento pode ser computacionalmente custoso levando a um processo de aprendizado lento. Para contornar este problema, usa-se o *SGD* (*Stochastics Gradient Descent*).

Sendo a função de erro definida por

$$L(\theta|x, y) = \frac{1}{n}\sum_{i=1}^{n} C(h(\theta|x_i), y_i), \tag{2.30}$$

logo,

$$\nabla_\theta L(\theta|x, y) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta C(h(\theta|x_i), y_i). \tag{2.31}$$

Aplicando a *Lei dos grandes números* de estatística em uma amostra aleatória suficientemente grande do conjunto de treinamento de tamanho $m < n$, vale que

$$\frac{1}{m}\sum_{i=1}^{m} \nabla_\theta C(h(\theta|x_i), y_i) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta C(h(\theta|x_i), y_i). \tag{2.32}$$

Assim, o método *SGD* consiste em atualizar os parâmetros em cada época de treinamento t considerando não todos os dados de uma vez, mas em pequenas amostras sem reposição, ou *mini-batches* de tamanho m, usando uma aproximação para o gradiente aplicado em todos os dados:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L\left(\theta_t | x_i^m, y_i^m\right), \forall i = 1, 2, \ldots \tag{2.33}$$

Por sua simplicidade, o método *SGD* é um dos mais usados na resolução de problemas do tipo supervisionado.No entanto, apesar de sua eficiência, possui limitações: uma taxa de aprendizado η muito pequena leva à uma convergência muito lenta, enquanto um valor muito grande pode levar a não convergência; em funções convexas, com um valor adequado de η o método garante convergência para um ponto de mínimo global, porém, devido a estrutura das redes neurais, sua função de erro tende a não ser convexa, fazendo com que o método convirja para um ponto de mínimo local, ou em casos mais extremos, um ponto de sela.

Ao longo dos anos, foram propostos métodos de otimização do *SGD* para contornar suas limitações [20]. São eles:

- **Momento**: Devido a sua complexidade, é comum que a superfície gerada pela função de erro tenha muitas variações de inclinação em direções diferentes. Ao redor de um ponto mínimo, tais variações fazem com que o método *SGD* oscile fortemente levando a uma lenta convergência.

Figura 2.6: SGD *oscilando ao redor de um ponto mínimo.*

Fonte: Genevieve B. Orr.[5]

No método do momento [21], é adicionado uma fração γ do vetor de atualização anterior v_t ao vetor atual, corrigindo a direção de descida, reduzindo as oscilações e diminuindo o tempo de convergência. O usual é considerar $\gamma = 0.9$. Adotando $g_t = \nabla_\theta L(\theta_t | x, y)$, temos a regra de iteração:

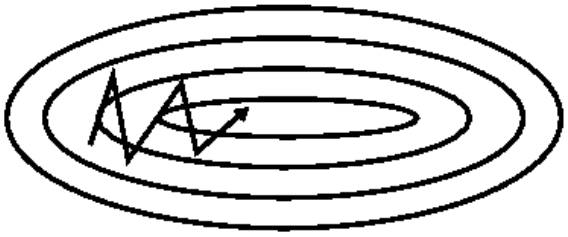
$$\begin{aligned} v_{t+1} &= \gamma v_t + \eta g_t \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{2.34}$$

- **NAG (*Nesterov accelerated gradient*)** : Sendo uma pequena modificação do método do momento, no método *NAG* [22], aplica-se o gradiente em uma aproximação da próxima posição θ_{t+1} dada por $\theta_t - \gamma v_t$ ajudando a corrigir sua trajetória de descida.

$$\begin{aligned} v_{t+1} &= \gamma v_t + \eta g_t(w_t - \gamma v_t) \\ \theta_{t+1} &= \theta_t - v_{t+1} \end{aligned} \tag{2.35}$$

[5]Disponível em: <https://www.willamette.edu/ gorr/classes/cs449/momrate.html>. Acesso em nov. 2018.

Figura 2.7: SGD *com momento.*



Fonte: Genevieve B. Orr.

Como posto por Ruder [20], no método *SGD* e variações, todos os parâmetros são atualizados usando a mesma taxa de aprendizado. No entanto, é interessante atualizar cada parâmetro individualmente, executando pequenas atualizações em parâmetros que respondam a características mais frequentes e grandes atualizações em parâmetros de características menos frequentes. Para isso, foram desenvolvidos os métodos adaptativos. Com intuito de melhorar a forma com que os parâmetros de uma rede neural são atualizados, adaptam taxas de aprendizado individuais para cada parâmetro, sendo boas escolhas para problemas envolvendo dados esparsos. São eles:

- **Adagrad**: No método *Adagrad* [23], dado uma taxa de aprendizado inicial seu valor é modificado em cada época t para cada parâmetro considerando seus gradientes até o tempo presente, atualizando cada parâmetro com a seguinte regra:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} g_{t,i} \tag{2.36}$$

  Onde $G_{t,i} = \sum_{j=1}^{t} g_{j,i}^2$ e ϵ é um valor suficientemente pequeno que evita divisões por zero. Uma das vantagens deste método é que não há necessidade de otimização manual do valor de η, bastando escolher um valor inicial e deixando que o algoritmo o adapte aos parâmetros ao longo do treinamento. O autor sugere como valor inicial $\eta = 0.01$.

- **Adadelta**:Apesar de eficiente, o método *Adagrad* possui um ponto fraco. Como $G_{t,ii} = \sum_{j=1}^{t} g_{j,i}^2$ é crescente ao longo do tempo, o termo $\frac{\eta}{\sqrt{G_{t,ii}+\epsilon}}$ é decrescente e eventualmente será infinitesimal, fazendo com que os parâmetros parem de ser atualizados. Pensando nisso foi criado o método *Adadelta* [24]. Neste método, ao invés de armazenar todos os quadrados dos gradientes passados, usa-se a média móvel exponencial dada por

$$E(g_{t,i}^2) = \gamma E(g_{t-1,i}^2) + (1-\gamma) g_{t,i}^2, \tag{2.37}$$

  onde γ é um parâmetro como no método do momento. O autor ainda salienta que, para manter a escala dos parâmetros, acrescenta-se o termo $\sqrt{E(\Delta\theta_{t-1,i}^2) + \epsilon}$ no lugar de η, e a regra de atualização fica:

$$\begin{aligned} \Delta\theta_{t,i} &= -\frac{\sqrt{E(\Delta\theta_{t-1,i}^2) + \epsilon}}{\sqrt{E(g_{t,i}^2) + \epsilon}} g_{t,i} \\ \theta_{t+1,i} &= \theta_{t,i} + \Delta\theta_{t,i} \end{aligned} \tag{2.38}$$

Note que com a correção de escala o valor de η não precisa mais ser definido.

- **RMSprop**: Paralelamente ao desenvolvimento do método Adadelta, Geoff Hinton [25], em uma de suas aulas na plataforma de cursos *online Coursera*, propôs um método não publicado chamado RMSprop, que nada mais é do que o método Adadelta sem a correção de escala, ou seja:

$$\begin{aligned} \Delta\theta_{t,i} &= -\frac{\eta}{\sqrt{E(g_{t,i}^2)+\epsilon}} g_{t,i} \\ \theta_{t+1,i} &= \theta_{t,i} + \Delta\theta_{t,i} \end{aligned} \tag{2.39}$$

  Geoff Hinton sugere que sejam usados os valores $\gamma = 0.9$ e $\eta = 0.001$.

- **Adam**: No método Adam (*adaptative moment estimation*) [26], junto da média móvel exponencial dos quadrados dos gradientes passados, calcula-se a média móvel exponencial dos próprios gradientes, ou seja:

$$\begin{aligned} E(g_{t,i}^2) &= \beta_2 E(g_{t-1,i}^2) + (1-\beta_2) g_{t,i}^2 \\ E(g_{t,i}) &= \beta_1 E(g_{t-1,i}) + (1-\beta_1) g_{t,i} \end{aligned} \tag{2.40}$$

  Estes valores podem ser vistos como estimadores do segundo e primeiro momento dos gradientes, respectivamente. Na publicação original, os estimadores são iniciados com valor 0. Os autores sugerem uma correção de estimação para evitar viés em torno de 0, definindo a regra de atualização como

$$\begin{aligned} \hat{E}(g_{t,i}^2) &= \frac{E(g_{t,i}^2)}{1-\beta_2^t} \\ \hat{E}(g_{t,i}) &= \frac{E(g_{t,i})}{1-\beta_1^t} \\ \theta_{t+1,i} &= \theta_{t,i} - \frac{\eta}{\sqrt{\hat{E}(g_{t,i}^2)+\epsilon}} \hat{E}(g_{t,i}), \end{aligned} \tag{2.41}$$

  usando os parâmetros $\beta_1 = 0.9$, $\beta_2 = 0.999$ e $\epsilon = 10^{-8}$.

- **AdaMax**: Kingma et al. [26], criadores do método *Adam*, sugerem uma modificação de seu método que se mostra mais estável, além de evitar o viés em torno de 0. O método *AdaMax*, com regra de atualização:

$$\begin{aligned} u_{t,i} &= \max\{\beta_2 E(g_{t-1,i}^2), |g_{t,i}|\} \\ \theta_{t+1,i} &= \theta_{t,i} - \frac{\eta}{u_{t,i}} \hat{E}(g_{t,i}) \end{aligned} \tag{2.42}$$

  Sugerem os parâmetros $\beta_1 = 0.9$, $\beta_2 = 0.999$ e $\eta = 0.002$.

- **Nadam**: O método Nadam (*Nesterov - accelerated adaptative moment estimation*) [27] surgiu como uma unificação das ideias dos métodos Adam e NAG. Neste método, substitui-se $\hat{E}(g_{t,i})$ em (2.38) por uma aproximação de $\hat{E}(g_{t+1,i})$, gerando a regra de

atualização:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{E}(g_{t,i}^2)} + \epsilon}(\beta_1 \hat{E}(g_{t,i}) + \frac{(1-\beta_1)g_{t,i}}{1-\beta_1^t}) \tag{2.43}$$

Onde $\hat{E}(g_{t,i})$ e $\hat{E}(g_{t,i}^2)$ são como em (2.41).

- **AMSgrad**: Reddi et al. [28] chamam atenção para o fato de que em alguns problemas, como reconhecimento de objetos e tradução, os método adaptativos não têm bom desempenho devido a influência das médias móveis exponenciais. Assim, desenvolveram em 2018 o método *AMSgrad*, que visa corrigir essa falha com a seguinte regra de atualização:

$$\begin{aligned} u_t &= \max\{u_{t-1}, E(g_{t,i}^2)\} \\ \theta_{t+1,i} &= \theta_{t,i} - \frac{\eta}{\sqrt{u_t} + \epsilon} E(g_{t,i}) \end{aligned} \tag{2.44}$$

Onde $E(g_{t,i})$ e $E(g_{t,i}^2)$ são como em (2.40).

2.4 *Backpropagation*

O método do gradiente descendente é um dos métodos mais usado para encontrar pontos ótimos de funções de difícil abordagem manual. No entanto, é necessário ter conhecimento prévio de suas derivadas parciais. Lembrando que a função de erro é dada por

$$L(w, b|x, y) = \frac{1}{n}\sum_{i=1}^{n} C(h(w, b|x_i), y_i), \tag{2.45}$$

uma abordagem natural é usar o método de diferenças

$$\begin{aligned} \frac{\partial L}{\partial w_i} &\approx \frac{L(w + he_i, b|x, y) - L(w, b|x, y)}{h}, \\ \frac{\partial L}{\partial b_i} &\approx \frac{L(w, b + he_i|x, y) - L(w, b|x, y)}{h}, \end{aligned} \tag{2.46}$$

onde $h > 0$ é pequeno e e_i denota o vetor canônico na direção i.Apesar de simples, na prática é uma abordagem computacionalmente custosa e lenta. Visto que modelos de redes neurais típicos envolvem de milhares a milhões de parâmetros, é necessário o cálculo dos valores $L(w + he_i, b|x, y)$ e $L(w, b + he_i|x, y)$ milhares ou milhões de vezes, tornando a abordagem inviável. O que se usa de fato é um poderoso e simples método que tem suas origens em artigos dos anos 70, mas só ganhou notoriedade com a publicação de Rumelhart et al.[29]: o algoritmo *backpropagation*, retro-propagação, ou retro-propagação de erros.

Explorando a estrutura em formato de média da função de erro, o algoritmo permite o cálculo das derivadas parciais de cada custo individual $C(h(w, b|x_i), y_i)$ em função do erro de variação, ou erro delta, de cada neurônio j em uma camada l definido por $\delta_j^l = \frac{\partial}{\partial z_j^l} C(h(w, b|x_i), y_j)$ a partir de 4 equações fundamentais.

Proposição 2.1. *Seja $h : \mathbb{R}^n \to \mathbb{R}^{c_L}$ função de uma rede neural* MLP *de arquitetura n-c_2-...-c_L, com matrizes de pesos sinápticos $W^l = [w^l_{ij}]_{c_l \times c_{l-1}}$, vetores de viés $b^l = [b^l_i]_{c_l \times 1}$ e funções de ativação deriváveis $\sigma_l : \mathbb{R} \to \mathbb{R}$, com $l = 2, \ldots, L$. Dado uma função de custo C, considere $C_x(w, b) = C(h(w, b|x), y)$ função de custo para um único dado de treinamento (x, y) qualquer. Considere as definições:*

a) $z^l = W^l a^{l-1} + b^l$

b) $a^l = \sigma_l(z^l)$

c) $\delta^l = \nabla_{z^l} C_x$

Então, $\forall l = 2, \ldots, L$

$$\delta^L = \nabla_{a^L} C_x \circ \sigma'_L(z^L) \tag{1}$$

$$\delta^l = ((W^{l+1})^t \delta^{l+1}) \circ \sigma'_l(z^l) \tag{2}$$

$$\frac{\partial C_x}{\partial w^l_{ij}} = a^{l-1}_j \delta^l_i \tag{3}$$

$$\frac{\partial C_x}{\partial b^l_i} = \delta^l_i \tag{4}$$

Demonstração. Fixe $l \in \{2, 3, \ldots, L\}$.

Equação (1): Considerando o elemento δ^L_i do vetor δ^L, por *c*), temos que

$$\delta^L_i = \frac{\partial C_x}{\partial z^L_i}, \tag{2.47}$$

e por *b*), vale que

$$a^L = \sigma_L(z^L). \tag{2.48}$$

Aplicando a regra da cadeia em C_x em função do vetor a^L, temos que

$$\begin{aligned} \frac{\partial C_x}{\partial z^L_i} &= \langle \nabla_{a^L} C_x, \frac{\partial a^L}{\partial z^L_i} \rangle \\ &= \sum_{j=1}^{c_L} \frac{\partial C_x}{\partial a^L_j} \frac{\partial a^L_j}{\partial z^L_i}. \end{aligned} \tag{2.49}$$

Mas como a_j^L só depende de z_j^L, para $j \neq i$ $\frac{\partial a_j^L}{\partial z_i^L} = 0$, logo

$$\begin{aligned}\sum_{j=1}^{c_L} \frac{\partial C_x}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_i^L} &= \frac{\partial C_x}{\partial a_i^L}\frac{\partial a_i^L}{\partial z_i^L} \\ &= \frac{\partial C_x}{\partial a_i^L}\sigma'_L(z_i^L).\end{aligned} \tag{2.50}$$

Portanto,

$$\delta^L = \nabla_{a^L} C_x \circ \sigma'_L(z^L). \tag{2.51}$$

Equação (2): Considere agora o elemento δ_i^l do vetor δ^l para algum l. Aplicando a regra da cadeia em C_x em relação a z^{l+1}, temos:

$$\begin{aligned}\delta_i^l = \frac{\partial C_x}{\partial z_i^l} &= \langle \nabla_{z^{l+1}} C_x, \frac{\partial z^{l+1}}{\partial z_i^l} \rangle \\ &= \sum_{j=1}^{c_{l+1}} \frac{\partial C_x}{\partial z_j^{l+1}}\frac{\partial z_j^{l+1}}{\partial z_i^l} \\ &= \sum_{j=1}^{c_{l+1}} \delta_j^{l+1}\frac{\partial z_j^{l+1}}{\partial z_i^l}.\end{aligned} \tag{2.52}$$

Por a):

$$\begin{aligned}z_j^{l+1} &= \sum_{k=1}^{c_l} w_{jk}^{l+1} a_k^l + b_j^{l+1} \\ &= \sum_{k=1}^{c_l} w_{jk}^{l+1}\sigma_l(z_k^l) + b_j^{l+1}.\end{aligned} \tag{2.53}$$

Logo

$$\frac{\partial z_j^{l+1}}{\partial z_i^l} = w_{ji}^{l+1}\sigma'_l(z_i^l). \tag{2.54}$$

Substituindo, temos:

$$\sum_{j=1}^{c_{l+1}} \delta_j^{l+1}\frac{\partial z_j^{l+1}}{\partial z_i^l} = \sum_{j=1}^{c_{l+1}} \delta_j^{l+1} w_{ji}^{l+1}\sigma'_l(z_i^l). \tag{2.55}$$

Portanto

$$\delta^l = ((W^{l+1})^t \delta^{l+1}) \circ \sigma'_l(z^l). \tag{2.56}$$

Equação (3): Considere um peso sináptico w_{ij}^l qualquer. Aplicando a regra da cadeia

em C_x em função de z^l, temos:

$$\begin{aligned}\frac{\partial C_x}{\partial w_{ij}^l} &= \langle \nabla_{z^l} C_x, \frac{\partial z^l}{\partial w_{ij}^l} \rangle \\ &= \sum_{k=1}^{c_l} \frac{\partial C_x}{\partial z_k^l} \frac{\partial z_k^l}{\partial w_{ij}^l} \\ &= \sum_{k=1}^{c_l} \delta_k^l \frac{\partial z_k^l}{\partial w_{ij}^l}.\end{aligned} \tag{2.57}$$

Por a), se $k \neq i$ $\frac{\partial z_k^l}{\partial w_{ij}^l} = 0$. Logo

$$\begin{aligned}\sum_{k=1}^{c_l} \delta_k^l \frac{\partial z_k^l}{\partial w_{ij}^l} &= \delta_i^l \frac{\partial z_i^l}{\partial w_{ij}^l} \\ &= \delta_i^l \sigma_{l-1}(z_j^{l-1}) \\ &= \delta_i^l a_j^{l-1}.\end{aligned} \tag{2.58}$$

Portanto

$$\frac{\partial C_x}{\partial w_{ij}^l} = a_j^{l-1} \delta_i^l. \tag{2.59}$$

Equação (4):Considere um valor de viés b_i^l qualquer. Aplicando a regra da cadeia em C_x em relação a z^l, temos:

$$\begin{aligned}\frac{\partial C_x}{\partial b_i^l} &= \langle \nabla_{z^l} C_x, \frac{\partial z^l}{\partial b_i^l} \rangle \\ &= \sum_{k=1}^{c_l} \delta_k^l \frac{\partial z_k^l}{\partial b_i^l}.\end{aligned} \tag{2.60}$$

Por a), se $k \neq i$ $\frac{\partial z_k^l}{\partial b_i^l} = 0$. Logo

$$\begin{aligned}\sum_{k=1}^{c_l} \delta_k^l \frac{\partial z_k^l}{\partial b_i^l} &= \delta_i^l \frac{\partial z_i^l}{\partial b_i^l} \\ &= \delta_i^l.\end{aligned} \tag{2.61}$$

Portanto

$$\frac{\partial C_x}{\partial b_i^l} = \delta_i^l. \tag{2.62}$$

□

O resultado deixa claro o caráter "*backpropagation*"do método: computa-se os erros na última camada e desloca-se os erros para trás através da transposta da matriz de pesos de cada camada, o que é consequência do fato de que uma rede neural é uma composição de funções. Com algumas modificações, o mesmo pode ser feito em redes neurais convolucionais.

2.5 Generalização

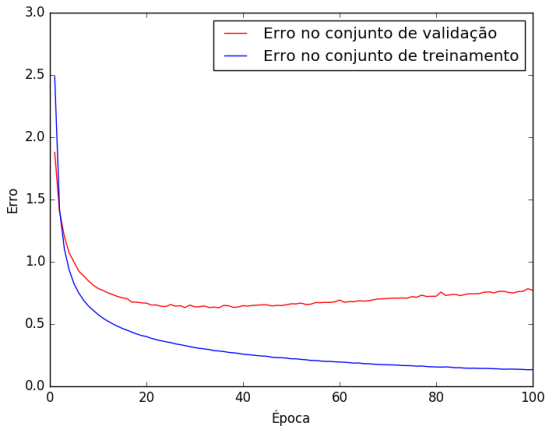
Devido sua capacidade de aprender com os dados, uma das grandes vantagens no uso de modelos de redes neurais é seu poder de generalização. Quando treina-se uma rede neural em um conjunto de dados finito, informações a respeito da estrutura desses dados são armazenadas em seus parâmetros na forma de "conhecimento". Assim, quando aplicada em dados no mundo real, ou seja, dados nunca antes "visto", tem desempenho tão bom quanto no conjunto de treinamento. Esse é o cenário ideal. Na prática, isso nem sempre ocorre.

Em um modelo de rede neural, é costumeiro que seja proposto uma arquitetura inicial, ou seja, o número de neurônios, camadas, e consequentemente o número de parâmetros. Caso o número de parâmetros iniciais seja demasiado grande, lhe dando uma maior capacidade de modelagem, é muito comum que , após o processo de treinamento, o modelo tenha apenas se ajustado aos dados, memorizando-os, e também a eventuais ruídos . Isso, associado ao fato de que o algoritmo de treinamento consiste basicamente em achar parâmetros que minimizem uma função de custo, faz com que o modelo treinado acabe tendo um desempenho ruim em dados nunca antes "vistos", perdendo sua capacidade de generalização. Chama-se esse fenômeno de *overfitting*, ou *sobre ajuste* [9].

Escolher um bom modelo inicial pode exigir habilidade e um pouco de intuição, no entanto, existem algumas técnicas que podem se usadas para identificar e amenizar o efeito do *overfitting*, como:

- **Validação**:Na prática, ao treinar uma rede neural, usa-se dois conjuntos de dados: o conjunto de treinamento e o conjunto de teste. O conjunto de treinamento é o usado para ajustar os parâmetros do modelo. O conjunto de teste simula o "mundo real", onde se calcula uma medida de desempenho para verificar o quanto o modelo é capaz de generalizar para dados fora do conjunto de treinamento, após o processo de treinamento. Uma boa técnica para detectar o *overfitting*, é adotar um terceiro conjunto de dados: o conjunto de validação. Ao final de cada época de treinamento, calcula-se o erro de generalização do modelo nos dados de validação mantendo um monitoramento de sua capacidade de generalização. Na presença de *overfitting*, ao longo das épocas o erro nos dados de treinamento tende a diminuir, enquanto o erro no conjunto de validação tende a aumentar. Nesse instante, o modelo deixa de aprender e passa apenas a "memorizar"o conjunto de treinamento.

Figura 2.8: *Exemplo de ocorrência de overfitting.*



Fonte: Própria

Uma prática comum é o *early stopping*: encerrar o processo de treinamento quando observado que o erro no conjunto de validação não apresentou melhora por algum número de épocas, evitando computações desnecessárias. Além de ajudar a identificar o *overfitting*, o conjunto de validação pode ser usado para o ajuste dos hiper parâmetros do modelo, como a taxa de aprendizagem, valor de *mini-batch*, número de épocas, e a própria arquitetura. Segundo Nielsen [12], uma boa estratégia é treinar o modelo em uma pequena parte dos dados de treinamento, monitorando o erro no conjunto de validação enquanto faz-se pequenas mudanças nos hiper parâmetros. É importante ressaltar que os 3 conjuntos de dados, além de terem a mesma natureza, são disjuntos e usados separadamente. Ajusta-se os parâmetros do modelo no conjunto de treinamento; monitora-se a presença de *overfitting* e ajusta-se os hiper parâmetros no conjunto de validação; calcula-se a capacidade final de generalização do modelo no conjunto de teste.

- **Regularização**:Um dos motivos do *overfitting* é o excesso de parâmetros no modelo. Uma ideia natural é reduzir o número de parâmetros. No entanto, com menos parâmetros, o modelo tem menos capacidade para aprender com os dados. No mundo do *machine learning*, uma das formas de resolução é a aplicação dos métodos de regularização [10]. Nesses métodos, tendo uma função de custo

$$L(\theta|x,y) \tag{2.63}$$

sobre os parâmetros θ, aplica-se uma penalização dada por uma norma dos parâmetros e tem-se uma nova função de custo

$$\hat{L}(\theta|x,y) = L(\theta|x,y) + \lambda r(\theta) \tag{2.64}$$

que será usada no processo de treinamento, onde $\lambda > 0$ é um hiper parâmetro que indica o tamanho da contribuição da componente de penalização.Para $\theta \in \mathbb{R}^n$,alguns exemplos de regularização, são:

  - L1:

$$\begin{aligned} r(\theta) &= ||\theta||_1 \\ &= \sum_{i=1}^{n} |\theta_i| \end{aligned} \tag{2.65}$$

  - Ł2:

$$\begin{aligned} r(\theta) &= \frac{1}{2}||\theta|_2^2 \\ &= \frac{1}{2}\sum_{i=1}^{n} \theta_i^2 \end{aligned} \tag{2.66}$$

Para modelos de redes neurais,é usual adotar uma penalização somente em relação aos pesos. Devido sua estrutura, os valores de *bias* tem pouca influência na ocorrência de *overfitting* [10] . Por exemplo, adotando a regularização $L2$ e $w \in \mathbb{R}^n$, tem-se:

$$\hat{L}(w,b|x,y) = L(w,b|x,y) + \frac{\lambda}{2}||w||^2. \tag{2.67}$$

- **Dropout**:Outro método de regularização aplicado especificamente em redes neurais é o chamado *dropout*, desenvolvido no trabalho de Hinton et al. [25]. Neste método, dado um valor $p \in (0, 1)$, em cada época de treinamento cada neurônio das camadas ocultas tem uma probabilidade p de ser "esquecido", deixando de contribuir com a rede e de ser treinado, voltando ao normal na época seguinte. Nas palavras de Krizhevsky et al. [30]: *"This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons."*

Capítulo 3

Classificação

3.1 Conjuntos de dados

Para o treinamento e avaliação dos modelos de classificação deste trabalho, foram usados os conjuntos de dados separados em treinamento, validação e teste originados do trabalho de Julca-Aguilar [4]. Em parte de seu trabalho, o problema de classificação de símbolos é abordado transformando os dados de natureza *online* em dados *offline*. Nas palavras de Julca-Aguilar: " *While online features introduce important information regarding the writting process, they can be misleading: for instance, the extracted features of a line written from left to right are different from those of a line written from right to left, even if the shapes are the same. Offline features are intended to capture symbol shape patterns disregarding the writting process.Our offline features include 2D histograms calculated over a square box that encloses the symbol. The square box side is defined as the biggest value between the width and height of the symbol. We place the enclosing box so that its center corresponds to the symbol's bounding box center. Thus, the enclosing box is partitioned into* $k \times k$ *cells, and the histogram is calculated by counting the fuzzy weighted contribution of each symbol's point to its four closest cells (Almazan et al., 2011)"..."In addition to the bidimensional histogram of a symbol, we calculate a second histogram that aims to capture contextual information of the symbol. This contextual histogram is calculated using the bounding box of the symbol as enclosing box, and counting points of the expression that do not belong to the symbol."..."These features are able to capture information useful to solve ambiguities.".*

Assim, o conjunto de dados utilizados neste trabalho são os histogramas de símbolos matemáticos em 3 formatos diferentes:

- D1: $k = 20$;

- D2: $k = 30$;

- D3: $k = 20$ com contexto.

Os dados estão divididos em 102 classes: 101 símbolos matemáticos retirados diretamente de expressões matemáticas, e *junk*. Como posto por Julca-Aguilar et al. [31], na etapa de segmentação do processo de reconhecimento de expressões matemáticas manuscritas online, é normal que ocorram erros de segmentação ocasionando em falsas hipóteses de símbolos. Na prática, um classificador treinado apenas com verdadeiros símbolos atribuiria baixo grau de confiança quando aplicado em um falso símbolo, no entanto, resultados anteriores mostraram que adicionar uma classe representando falsos símbolos, ou *junk*, dando ao classificador capacidade de distinguir entre verdadeiras e falsas hipóteses tem potencial de melhorar sua

taxa de classificação e o reconhecimento de expressões como um todo. Os dados *junk* são formados por misturas entre falsos símbolos, não possuindo um padrão específico.

Todos os 3 histogramas possuem as mesmas quantidades de amostras:

- **Treinamento**: 183243 amostras, sendo 91842 símbolos e 91401 *junk*;
- **Validação**: 20001 amostras, sendo 10018 símbolos e 9993 *junk*;
- **Teste**: 24307 amostras, sendo 12156 símbolos e 12151 *junk*.

Para uma relação completa entre as classes, os símbolos que representam e número de amostras veja apêndice B.

Vale observar que, apesar dos dados originais das expressões serem online, a classificação é realizada a partir da imagem estática, representada pela imagem dos histogramas. Os conjuntos D1 e D2 são imagens com, respectivamente, 20×20 e 30×30 *pixels*, enquanto o conjunto D3 pode ser visto como uma imagem com 2 canais, o primeiro sendo o histograma do símbolo, e o segundo, o contexto. Assim, a abordagem deste trabalho também poderia ser aplicada em dados originalmente *offline* de expressões matemáticas.

Figura 3.1: *Símbolo "y" nos conjuntos D1, D2 e D3 em tamanho real.*

(a) *Histograma D1.*

(b) *Histograma D2.*

(c) *Histograma D3.*

(d) *Contexto.*

Por se tratarem de histogramas, todos os seus valores se encontram no intervalo $(0, 1)$ e somam 1, estando dentro das hipóteses do teorema da aproximação universal.

3.2 Métricas

Para avaliar o desempenho dos modelos, inicialmente consideramos a classificação dos dados como símbolo, a classe positiva, ou *junk* (não-símbolo), a classe negativa. Definimos as medidas SE (Sensibilidade) e ES (Especificidade) como:

$$\begin{aligned} SE &= \frac{\text{Acertos positivos}}{\text{Total de positivos}} \\ ES &= \frac{\text{Acertos negativos}}{\text{Total de negativos}}. \end{aligned} \tag{3.1}$$

Estas medidas podem ser interpretadas como a probabilidade de identificar corretamente símbolos e *junk*.

Considerando apenas os símbolos corretamente classificados como símbolos, temos as métricas:

- **Matriz de confusão**: Em um problema de classificação com k classes, definimos a matriz de confusão como um matriz $C_{k\times k}$ onde cada entrada $C_{i,j}$ contém o número de dados de entrada de classe i classificados como sendo de classe j.

- **Acurácia**:A acurácia mede o número de dados corretamente classificados em relação ao total. Em termos da matriz de confusão, temos:

$$Acc = \frac{\sum_{i=1}^{k} C_{i,i}}{\sum_{i=1}^{k}\sum_{j=1}^{k} C_{i,j}}. \tag{3.2}$$

  Pode-se definir ainda a acurácia $Top-d$. Dado um modelo de classificação cuja saída é uma distribuição de probabilidade sob k classes, a acurácia $Top-d$ é a razão entre o número de dados cuja classe está entre as d maiores probabilidades de sua distribuição e o total. Para $d = 1$, tem-se o valor Acc.

- ***Recall***:Dada uma matriz de confusão, definimos o valor de *recall*, sensibilidade, ou taxa de verdadeiros positivos, para cada classe $i = 1, 2, \ldots, k$, como

$$Rec_i = \frac{C_{i,i}}{\sum_{j=1}^{k} C_{i,j}}. \tag{3.3}$$

  O valor Rec_i pode ser interpretado como a probabilidade de classificar corretamente uma entrada de classe i.

- **Precisão**:Dada uma matriz de confusão, definimos o valor de precisão, ou valor preditivo positivo, para cada classe $i = 1, 2, \ldots, k$, como

$$Prec_i = \frac{C_{i,i}}{\sum_{j=1}^{k} C_{j,i}}. \tag{3.4}$$

  O valor $Prec_i$ pode ser interpretado como a probabilidade de uma entrada classificada como de classe i ser de fato de classe i.

- F_1:O valor $F_{1,i}$ junta as informações de Rec_i e $Prec_i$ em uma só medida calculando a média harmônica entras elas, ou seja:

$$F_{1,i} = 2\frac{Rec_i Prec_i}{Prec_i + Rec_i}. \tag{3.5}$$

  De modo mais geral, temos a métrica

$$F_{\beta,i} = (1+\beta^2)\frac{Rec_i Prec_i}{\beta^2 Prec_i + Rec_i}, \tag{3.6}$$

  que pondera a média dando mais importância à Rec_i para $\beta > 1$ e à $Prec_i$ para $0 < \beta < 1$.

- **MCC**: O MCC (*Matthews Correlation Coefficient*) [32], inicialmente criado para analisar problemas de classificação binários, é uma medida de correlação aplicada em uma matriz de confusão que assume valores de -1 até 1. O valor 1 indica alta eficiência na tarefa de classificação; 0, não melhor do que uma classificação aleatória; -1,total

ineficiência na tarefa de classificação. Pode ser generalizado para casos com mais de uma classe. Dada uma matriz de confusão $k \times k$, temos:

$$MCC = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{k} C_{i,i}C_{j,l} - C_{i,j}C_{l,i}}{\sqrt{\sum_{i=1}^{k}(\sum_{j=1}^{k} C_{i,j})(\sum_{\substack{i'=1,\\ i'\neq i}}^{k}\sum_{j'=1}^{k} C_{i',j'})}\sqrt{\sum_{i=1}^{k}(\sum_{j=1}^{k} C_{j,i})(\sum_{\substack{i'=1,\\ i'\neq i}}^{k}\sum_{j'=1}^{k} C_{j',i'})}}. \quad (3.7)$$

Em seu artigo, Chicco [33] explica que o MCC é uma boa medida de desempenho pois considera o desbalanceamento dos dados em seu cálculo, sendo melhor que outras métricas como Acc e F_β.

3.3 Modelos

Devido a falta de uma teoria mais forte na área de redes neurais, não existem métodos exatos para se determinar a arquitetura inicial de modelos *MLP* e *CNN*, levando à tentativa e erro. No entanto, alguns autores desenvolveram regras empíricas para se determinar uma boa arquitetura inicial.

Heaton [34] afirma que, na prática, modelos *MLP* não precisam de mais do que duas camadas ocultas para ter um bom desempenho, além de propor algumas regras para determinar o número de neurônios em uma camada oculta:

- M1: O número de neurônios em uma camada oculta deve estar entre o tamanho da entrada e o tamanho da saída. A média entre a entrada e a saída é uma boa medida.
- M2: O número de neurônios deve ser 2/3 da entrada mais a saída.
- M3: O número de neurônios deve ser menor que o dobro da entrada.

Já Masters[35], propõe que o número de neurônios seja a média geométrica entre a entrada e a saída (M4). Portanto, os modelos *MLP* preliminares deste trabalho seguem estas 4 abordagens com uma e duas camadas ocultas em cada tipo de histograma.

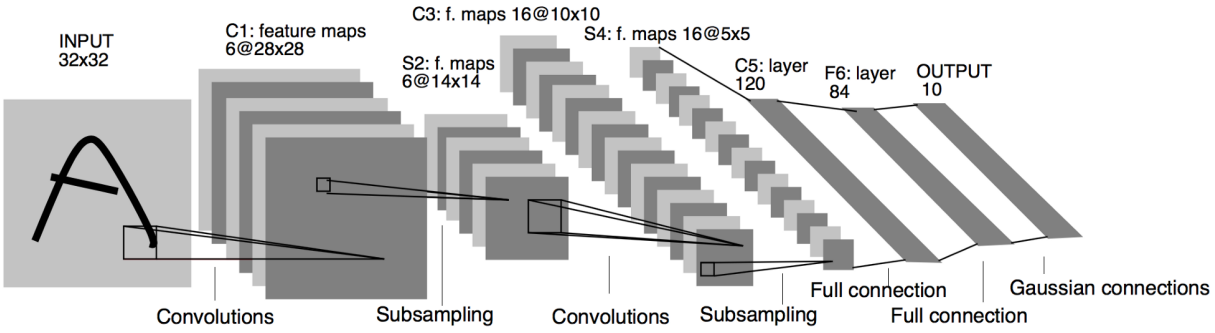
Quanto aos modelos convolucionais, foi usado como ponto de partida o modelo *Lenet5* (L5) desenvolvido no artigo de LeCunn et al.[18]. Com o propósito semelhante ao deste trabalho, este modelo foi criado para resolver o problema de classificação de dígitos manuscritos usando o famoso banco de dados *mnist*.

Sua arquitetura possui uma camada de convolução com 6 *feature maps* e filtro 5×5, seguida de uma camada de *maxpooling* 2×2, seguida de uma camada de convolução com 16 *feature maps* e filtro 5×5, novamente é aplicado uma camada de *maxpooling* 2×2 e enfim uma rede *MLP* de arquitetura $400 - 120 - 84 - 10$.

A partir do modelo *Lenet5*, propomos as seguintes variações no seu bloco convolucional:

- C1: Primeira camada de convolução com 10 *feature maps* e filtro 5×5, segunda camada de convolução com 20 *feature maps* e filtro 3×3.
- C2: Primeira camada de convolução com 20 *feature maps* e filtro 5×5, segunda camada de convolução com 10 *feature maps* e filtro 3×3.
- C3: Primeira camada de convolução com 20 *feature maps* e filtro 5×5, segunda camada de convolução com 30 *feature maps* e filtro 3×3.

Figura 3.2: *Arquitetura do modelo Lenet5.*



Fonte: LeCunn et al.[18].

- C4: Primeira camada de convolução com 30 *feature maps* e filtro 5×5, segunda camada de convolução com 20 *feature maps* e filtro 3×3.

- C5: Primeira camada de convolução com 30 *feature maps* e filtro 5×5, segunda camada de convolução com 30 *feature maps* e filtro 3×3.

Todos os modelos usam função de ativação *ReLU* e *softmax* na camada de saída para classificação.

3.4 Resultados

Como visto na seção (2.3), a natureza esparsa dos dados utilizados neste trabalho indica o uso de otimizadores adaptativos. Testando os otimizadores vistos na seção (2.3) em um pequeno conjunto de dados, o algoritmo *Nadam* se mostrou mais eficiente para modelos *MLP*, enquanto o algoritmo *Adadelta* se mostrou mais eficiente para modelos convolucionais. Todos os otimizadores foram usados com os parâmetros propostos por seus autores.

Para reduzir o *overfitting*, nos modelos *MLP* de uma camada foi usado probabilidade de *dropout* 0.5 antes da camada de saída, e nos modelos com duas camadas, *dropout* de 0.5 entre a primeira e a segunda camada. Nos modelos convolucionais, foi usado *dropout* de 0.2 na transição do bloco convolucional para a rede *MLP*.

Todos os modelos foram treinados durante 100 épocas usando *mini-batch* de tamanho 500. Ao final do treinamento de cada modelo, é considerado apenas os parâmetros com menor erro no conjunto de validação, refletindo sua capacidade de generalização. Como função de erro, foi usada a função de entropia cruzada.

Para os modelos *MLP*, temos os resultados preliminares:

Tabela 3.1: *Erro de generalização no conjunto de validação D1.*

	Camadas ocultas	
Modelo	1	2
M1	0.57	0.48
M2	0.55	0.46
M3	0.53	0.46
M4	0.58	0.49

Tabela 3.2: *Erro de generalização no conjunto de validação D2.*

	Camadas ocultas	
Modelo	1	2
M1	0.57	0.51
M2	0.56	0.50
M3	0.56	0.50
M4	0.60	0.54

Tabela 3.3: *Erro de generalização no conjunto de validação D3.*

	Camadas ocultas	
Modelo	1	2
M1	0.53	0.47
M2	0.52	0.45
M3	0.52	0.45
M4	0.55	0.49

Analisando os resultados das Tabelas (3.1), (3.2) e (3.3), fica claro que, de modo geral, os modelos *MLP* com duas camadas ocultas possuem menor erro em seus respectivos conjuntos de validação e portanto maior poder de generalização. Individualmente, os modelos M2 e M3 exibem melhor e igual performance em cada conjunto de dados, portanto, podemos considerar como modelo definitivo em cada histograma o de menor arquitetura e menor quantidade de parâmetros, ou seja, o modelo M2. Na Tabela (3.4) temos os valores de erro, *SE* e *ES* de cada modelo enquanto na Tabela (3.5) temos os valores *MCC* e *Top-d*, todos calculados em seus conjuntos de teste, ou seja, representam a capacidade final de generalização de cada modelo, visto que em nenhum momento do processo de treinamento os modelos entraram em contato com os dados de teste.

Tabela 3.4: *SE e ES dos modelos MLP.*

Histograma	Erro	SE (%)	ES (%)
D1	0.35	95.39	92.58
D2	0.38	95.59	91.85
D3	0.33	97.25	94.04

Tabela 3.5: *MCC e acurácia Top-d dos modelos MLP.*

Histograma	MCC	Top-1 (%)	Top-2 (%)	Top-3 (%)	Top-4 (%)	Top-5 (%)
D1	0.89	90.09	95.79	97.59	98.37	98.89
D2	0.89	90.09	95.27	97.28	98.10	98.64
D3	0.88	89.00	94.96	96.85	97.66	98.22

Procedendo da mesma forma para os modelos convolucionais, temos os resultados:

Tabela 3.6: *Erros de generalização nos conjuntos de validação.*

	Modelo					
Histograma	C1	C2	C3	C4	C5	L5
D1	0.39	0.40	0.37	0.37	0.38	0.45
D2	0.43	0.43	0.42	0.40	0.44	0.52
D3	0.33	0.34	0.32	0.30	0.31	0.42

De acordo com a Tabela (3.6), o modelo C4 é o que melhor generaliza nos 3 conjuntos de dados. Tal como nos modelos *MLP*, temos suas medidas de desempenho:

Tabela 3.7: *SE e ES dos modelos convolucionais.*

Histograma	Erro	SE (%)	ES (%)
D1	0.29	96.78	93.48
D2	0.32	96.55	93.33
D3	0.24	97.83	96.48

Tabela 3.8: *MCC e acurácia Top-d dos modelos convolucionais.*

Histograma	MCC	Top-1 (%)	Top-2 (%)	Top-3 (%)	Top-4 (%)	Top-5 (%)
D1	0.91	91.39	96.43	98.22	99.04	99.39
D2	0.90	90.76	96.19	98.19	98.83	99.12
D3	0.91	91.76	96.86	98.26	98.86	99.15

3.5 Discussão

Em relação as arquiteturas, comparando as Tabelas (3.4) e (3.7) é perceptível que os modelos convolucionais, além de possuírem os menores erros de generalização, se mostram melhores em distinguir símbolos e *junk* do que modelos *MLP*, alcançando valores SE e ES de no máximo 97.83% e 96.48% enquanto os modelos *MLP* alcançam no máximo 97.25% e 94.04% respectivamente.

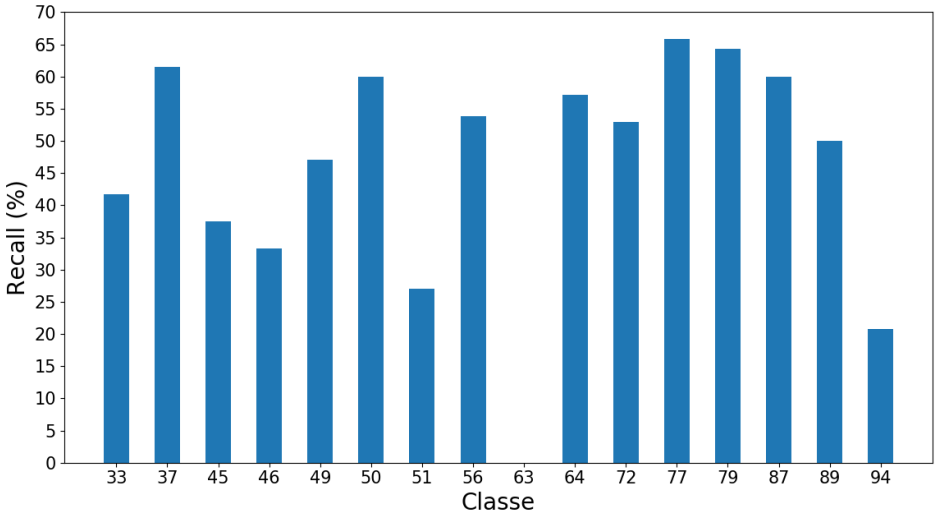
De acordo com as Tabelas (3.5) e (3.8), modelos convolucionais também mostram um desempenho levemente superior na tarefa de classificação de símbolos. Enquanto os modelos *MLP* possuem valores MCC, Top-1 e Top-5 de no máximo 0.89, 90.09% e 98.89%, modelos convolucionais alcançam no máximo os valores 0.91, 91.76% e 99.39% respectivamente. Vale observar que os modelos convolucionais mostram melhor desempenho usando um menor número de parâmetros. De fato, o menor modelo *MLP* possui 311.508 parâmetros, enquanto o maior modelo convolucional possui 85.154 parâmetros.

Analisando a matriz de confusão e o *recall* de cada símbolo em cada modelo estabelecendo um valor de corte de 70%, podemos notar que 16 classes possuem valor inferior tanto nos modelos *MLP* quanto nos convolucionais. Na Figura (3.3), temos os máximos desses valores. Para identificar os símbolos, veja apêndice B.

Grande parte desses erros de classificação são causados por ambiguidades entre os símbolos. Como apontado por Blostein e Grbavec [3], devido a estrutura das expressões matemáticas, existem muitas ambiguidades entre símbolos, tanto do ponto de vista semântico quanto

estrutural. Por exemplo, um ponto pode significar um ponto decimal, uma multiplicação, ou ser simplesmente um ruído. A variedade de caracteres, tipografias, tamanhos, escrita pessoal, somada as ambiguidades, torna difícil o processo de reconhecimento de símbolos isolados.

Figura 3.3: *Máximo dos recalls menores que 70%.*



Analisando os tipos de padrões presentes nos conjuntos de dados, percebe-se a presença de pelo menos 12 padrões ambíguos envolvendo 31 símbolos diferentes. Exemplos de ambiguidades são:

- Os símbolos "3", "∃";
- Os símbolos "9","g", "q";
- Os símbolos ",", "1 ", "l","|", "′ ".

Um outro fator que contribui para baixas taxas de *recall* é a quantidade de amostras disponíveis para treinamento. De fato, nos conjuntos de treinamento deste trabalho, existem símbolos com amostras inferiores a 100, porém, é interessante observar que, desses símbolos, apenas 1 está presente na relação exposta na Figura (3.3), o símbolo "γ"(79). De maneira geral, todos os melhores modelos deste trabalho mostraram uma boa taxa de *recall* em alguns dos símbolos com amostras inferiores a 100, alguns chegando a quase 100% nos conjuntos de teste.

Em relação aos conjuntos de dados, de acordo com as Tabelas (3.4) e (3.7), D2 exibe os maiores erros de generalização alcançando no mínimo 0.32, enquanto D1 e D3 alcançam, respectivamente, no mínimo 0.29 e 0.24. Além de possuir os menores erros de generalização, o conjunto D3 aparenta garantir maior poder de distinção entre símbolos e *junk* para os modelos *MLP* e *CNN*, gerando os maiores valores SE e ES dentre todos os conjuntos.

Considerando que D3, o histograma com contexto, tem o propósito de ajudar na classificação de símbolos ambíguos, podemos comparar seu desempenho com D1 na classificação de tais símbolos. Analisando o *recall* dos 31 símbolos identificados com padrões ambíguos, temos o comparativo entre os modelos *MLP* na Figura (3.4), e entre os modelos convolucionais na Figura (3.5). Para identificar os símbolos, veja apêndice B.

Figura 3.4: *Recall de símbolos ambíguos para modelos MLP.*

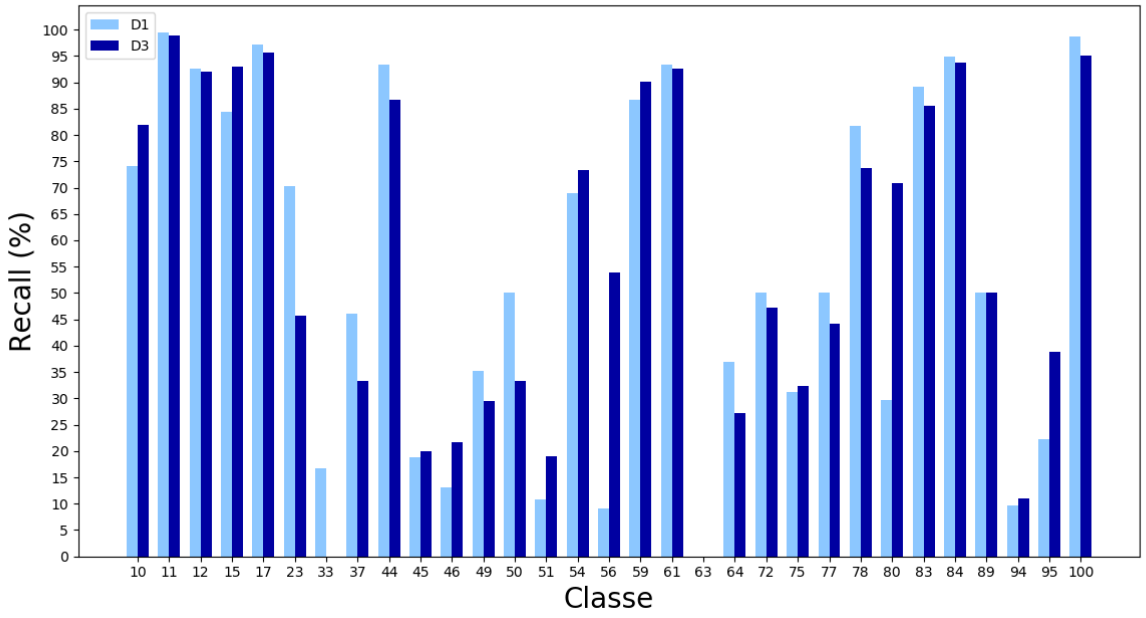
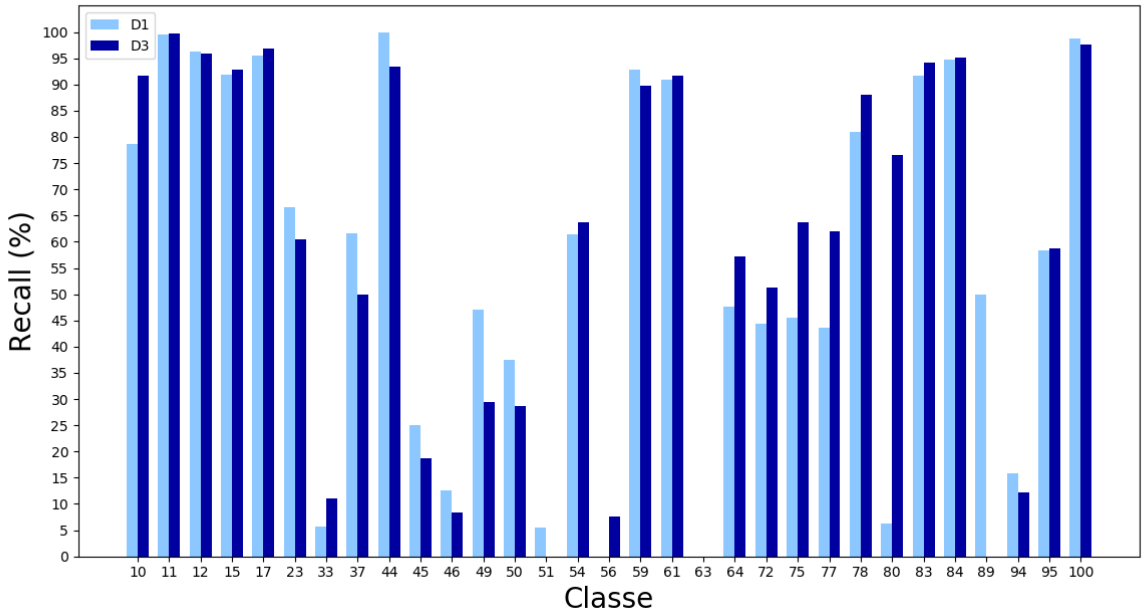


Figura 3.5: *Recall de símbolos ambíguos para modelos CNN.*



Em relação aos modelos *MLP*, nota-se que, dos 31 símbolos e 12 padrões, apenas 12 símbolos envolvendo 6 padrões tiveram algum tipo de melhoramento, um se manteve igual (89), e o restante teve desempenho reduzido. Destacam-se os símbolos de classe 56 ("/") e 80 ("|"), que foram, respectivamente, de 9.09% e 29.68% para 53.84% e 70.76%. Já para os modelos convolucionais, há alguma melhora em 17 símbolos de 9 padrões, com o restante tendo desempenho inferior. Destaca-se o símbolo de classe 80, que foi de 6.25% para 76, 56%.

Considerando todos os resultados, usar modelos convolucionais treinados no conjunto de dados com contexto se mostra o modelo de classificação mais balanceado, obtendo bons resultados tanto na identificação de símbolos e *junk* (SE = 97.83%, ES = 96.48%) quanto na classificação de símbolos (MCC =0.91, Top-1 = 91.76%, Top-5 = 99.15%), além de ter o menor erro de generalização (0.24).

É válido observar que o desempenho final dos modelos deste trabalho é reflexo das escolhas de arquitetura e dos conjuntos de dados disponíveis usados para treinamento. O uso de um conjunto de dados com um número de amostras por classe mais balanceado e a adoção de outras arquiteturas podem levar a modelos mais eficientes.

Capítulo 4

Conclusão

Neste trabalho, construímos modelos de classificação de símbolos matemáticos visando a etapa de reconhecimento de símbolos do processo de reconhecimento de expressões matemáticas manuscritas *online*. Como modelos de classificação, foram utilizadas redes neurais do tipo *MLP (Multilayer Perceptron)* e *CNN (Convolutional Neural Network)* treinadas em 3 conjuntos de dados do trabalho de Julca-Aguilar [31].

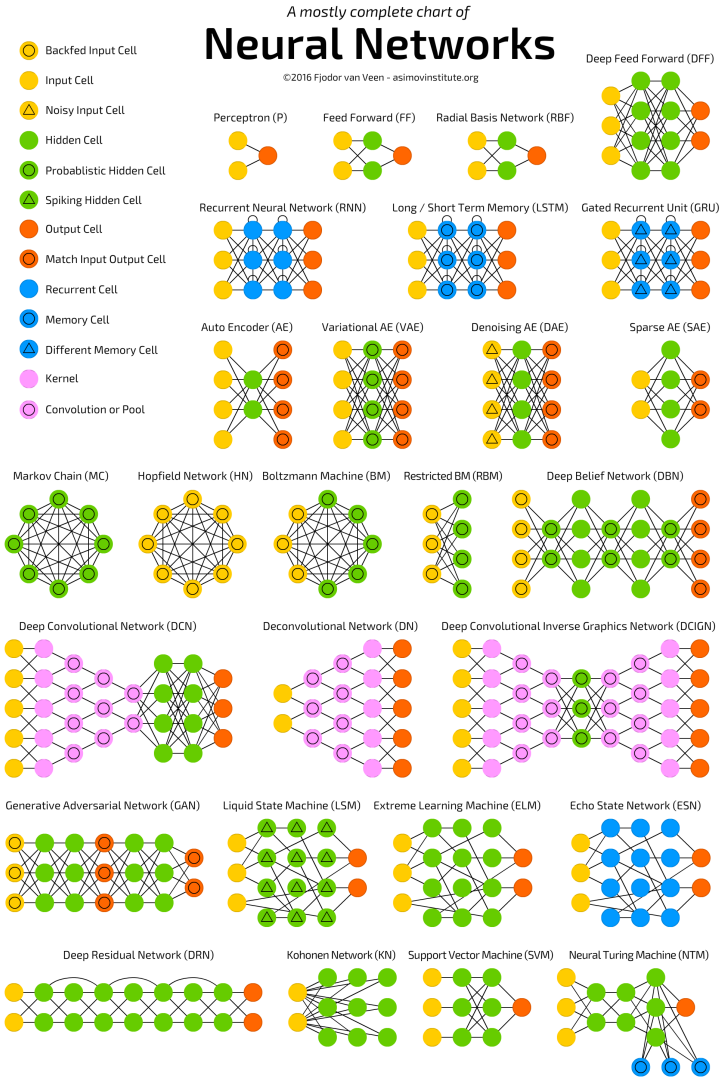
Ao final dos processos de treinamento, os melhores modelos mostraram boa capacidade de distinção entre símbolos e *junk*, e classificação de símbolos, com destaque para o modelo convolucional treinado no conjunto com contexto. Mesmo com a incapacidade de reconhecer a grande quantidade de símbolos ambíguos, os modelos podem ser usados na etapa de análise estrutural do reconhecimento de expressões fornecendo bons candidatos que serão analisados conforme o contexto da expressão matemática em questão.

Considerando o desempenho geral do histograma com contexto, uma proposta de trabalho futuro, além de explorar outras arquiteturas de redes neurais, é explorar essa estrutura de outras formas, como por exemplo, calcular o contexto de cada símbolo não se limitando ao seu *bounding box* mas abrangendo uma maior área ao redor do símbolo, ou ainda calcular contextos individuais para regiões do símbolo como acima, abaixo, direita e esquerda, gerando imagens com mais de 2 canais.

Apêndice A

Arquiteturas

Figura A.1: *Arquiteturas de redes neurais.*



Fonte: The Asimov Institute.[1]

[1]Disponível em: <http://www.asimovinstitute.org/neural-network-zoo/>. Acesso em out. 2018.

Apêndice B

Dados

Tabela B.1: *Classes, símbolos e número de amostras.*

		Número de amostras		
Classe	Símbolo	Treinamento	Validação	Teste
1	$\leq$	233	31	33
2	}	75	7	11
3	α	412	43	55
4	!	103	21	14
5	+	5673	622	758
6	(	4235	458	567
7	)	4230	458	566
8	.	741	21	55
9	/	224	19	28
10	,	715	82	97
11	-	8377	909	1118
12	3	2573	289	349
13	2	6530	715	889
14	{	75	7	11
15	1	6532	721	890
16	$\in$	23	3	4
17	0	1922	214	257
18	7	781	87	108
19	6	831	90	116
20	5	1052	122	146
21	4	1712	183	234
22	$\geq$	154	17	21
23	9	753	95	104
24	8	755	83	103
25	$\forall$	11	2	3
26	=	3928	434	520
27	ϕ	91	12	13
28	E	196	18	21
29	F	189	22	24
30	G	114	9	13
31	A	320	33	40
32	B	265	28	35

		Número de amostras		
Classe	Símbolo	Treinamento	Validação	Teste
33	C	288	31	36
34	∞	401	50	52
35	Δ	56	5	6
36	L	185	19	23
37	M	140	10	15
38	N	173	17	20
39	H	127	15	15
40	I	96	7	10
41	log	311	36	36
42	$<$	118	14	17
43	T	150	14	15
44	V	158	15	15
45	P	165	17	16
46	S	166	18	24
47	R	236	23	28
48	]	220	31	49
49	Y	125	13	17
50	σ	66	13	8
51	X	331	30	37
52	[	220	31	49
53	f	731	78	94
54	g	345	30	46
55	d	1148	119	154
56	$\prime$	125	11	13
57	e	502	53	66
58	b	1679	185	223
59	c	992	88	132
60	a	2611	279	341
61	n	2379	267	307
62	$\neq$	119	14	16
63	o	130	11	12
64	l	137	13	22
65	m	521	58	67
66	j	325	37	43
67	k	664	75	90
68	h	291	3	40
69	i	1055	114	132
70	tan	293	39	40
71	w	166	15	20
72	v	348	40	41
73	u	383	37	47
74	t	765	81	93
75	s	291	21	35
76	r	524	40	72
77	q	347	41	44
78	p	628	65	84
79	γ	99	11	14
80	|	536	62	65

		Número de amostras		
Classe	Símbolo	Treinamento	Validação	Teste
81	z	1123	120	152
82	...	159	40	34
83	y	1847	225	246
84	x	5303	587	690
85	$\sqrt{}$	1877	213	242
86	lim	305	35	31
87	λ	31	7	5
88	sin	822	95	98
89	$\exists$	6	4	2
90	μ	63	7	7
91	cos	608	70	77
92	θ	568	64	79
93	$\sum$	649	74	79
94	$\times$	632	72	82
95	$\div$	154	19	18
96	$\rightarrow$	319	37	43
97	$\pm$	161	17	22
98	$>$	65	7	8
99	π	523	68	74
100	$\int$	630	67	81
101	β	306	39	42
102	*junk*	91401	9993	12151
	Total	183243	20011	24307

Referências Bibliográficas

[1] Harold Mouchere, Christian Viard-Gaudin, Richard Zanibbi, and Utpal Garain. Icfhr 2014 competition on recognition of on-line handwritten mathematical expressions (crohme 2014). In *Frontiers in handwriting recognition (icfhr), 2014 14th international conference on*, pages 791–796. IEEE, 2014. 1, 2

[2] Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15, 2000. 1

[3] Horst Bunke and Patrick Shen-pei Wang. *Handbook of character recognition and document image analysis*. World scientific, 1997. 1, 31

[4] Frank Dennis Julca Aguilar. *Recognition of online handwritten mathematical expressions using contextual information*. PhD thesis, Universidade de São Paulo. 2, 25

[5] Simon Haykin. *Redes neurais: princípios e prática*. Bookman Editora, 2007. 2, 5, 10

[6] François Chollet et al. Keras (2015), 2017. 2

[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 2

[8] Tom M Mitchell. Machine learning (mcgraw-hill international editions computer science series). 1997. 3

[9] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook New York, NY, USA:, 2012. 3, 22

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 3, 23

[11] Khan Academy. *Visão geral da estrutura do neurônio e a sua função*. Disponível em: <https://pt.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function>, 2018. 5

[12] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press USA, 2015. 5, 11, 23

[13] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. 5, 6

[14] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 5, 6

[15] Data Science Academy. *Deep Learning Book*. Disponível em: <http://www.deeplearningbook.com.br/>, 2018. 7

[16] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 9

[17] Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *arXiv preprint arXiv:1505.03654*, 2015. 10

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 10, 28, 29

[19] Zhifei Zhang. Derivation of backpropagation in convolutional neural network (cnn). 2016. 12

[20] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 15, 16

[21] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999. 15

[22] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pages 543–547, 1983. 15

[23] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 16

[24] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 16

[25] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 17, 24

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 17

[27] Timothy Dozat. Incorporating nesterov momentum into adam. 2016. 17

[28] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. 2018. 18

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986. 18

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 24

[31] Frank JulcaAguilar, Nina ST Hirata, Christian ViardGaudin, Harold Mouchère, and Sofiane Medjkoune. Mathematical symbol hypothesis recognition with rejection option. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 500–505. IEEE, 2014. 25, 35

[32] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975. 27

[33] Davide Chicco. Ten quick tips for machine learning in computational biology. *BioData mining*, 10(1):35, 2017. 28

[34] Jeff Heaton. *Introduction to neural networks with Java*. Heaton Research, Inc., 2008. 28

[35] Timothy Masters. *Practical neural network recipes in C++*. Morgan Kaufmann, 1993. 28