UNIVERSIDADE FEDERAL DO PARANÁ
MARIA EDUARDA DALL NEGRO MOCHINSKI

ASPECTOS MATEMÁTICOS DO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS

CURITIBA
2023

UNIVERSIDADE FEDERAL DO PARANÁ
MARIA EDUARDA DALL NEGRO MOCHINSKI

ASPECTOS MATEMÁTICOS DO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS

Monografia apresentada como requisito parcial à conclusão do curso de graduação em Matemática Industrial pela Universidade Federal do Paraná.
Orientador: Prof. Dr. Lucas Garcia Pedroso.

CURITIBA
2023

"Whatever it is you seek, you have to put in the time, the practice, the effort. You must give up a lot to get it. It has to be very important to you. And once you have attained it, it is your power. It can't be given away: it resides in you. It is literally the result of your discipline."

Michael Crichton, Jurassic Park

Resumo

Neste trabalho são apresentados os aspectos matemáticos de um algoritmo de treinamento de redes neurais artificiais, que são um modelo de aprendizagem de máquina. Como objetivo, buscamos descrever a estrutura de uma rede neural *feedforward* em detalhes para, então, apresentar os cálculos utilizados no treinamento do modelo. Aplicando conceitos de otimização contínua, são calculadas as derivadas parciais necessárias para a atualização dos parâmetros da rede, utilizando o método de retropropagação (*backpropagation*). O algoritmo estudado foi implementado na linguagem de programação *Python* e testado, como prova de conceito, em um problema de reconhecimento de imagem. Nos experimentos realizados com o algoritmo, foi possível obter uma acurácia de teste de 94,86% com uma estrutura simples de rede neural, com apenas uma camada intermediária.

Palavras-chave: *Redes Neurais Artificiais, Retropropagação, Aprendizagem de Máquina, Aprendizado Supervisionado.*

Abstract

In this work, we present the mathematical aspects of a training algorithm used in artificial neural networks, a machine learning model. The aim is to provide detailed descriptions of the structure of a feedforward neural network in order to present the calculations used to train the network. Applying concepts of continuous optimization, we calculate the partial derivatives needed to update the network parameters using the backpropagation method. The algorithm is implemented in the programming language *Python* and tested in the context of an image recognition problem as a proof of concept. In the experiments conducted with the algorithm, we achieved a test accuracy of 94.86% with a simple neural network structure, featuring only one intermediate layer

Keywords: *Artificial Neural Networks, Backpropagation, Machine Learning, Supervised Learning.*

Sumário

Lista de Figuras

.

Lista de Tabelas

.

Introdução

No contexto de aprendizado de máquina supervisionado, as redes neurais artificiais são um modelo flexível, capaz de modelar problemas diversos, desde problemas de regressão com uma saída em valores contínuos, até problemas de classificação com múltiplas classes. Esse modelo de aprendizagem de máquina ganhou relevância nos últimos anos com os avanços de sistemas autônomos inteligentes, capazes de aprender a desempenhar tarefas complexas de forma automática (CHOLLET, 2017).

Neste trabalho, foi selecionado para estudo um modelo específico de rede neural artificial, aplicado a um problema de reconhecimento de imagem para classificação multiclasse. Essa tarefa está presente no cotidiano quando separamos objetos em diferentes grupos, e, por isso, é um problema bastante explorado na área de estudo da aprendizagem de máquina. As redes neurais artificiais são um modelo de computação bioinspirada, pois seu desenvolvimento tomou como base as interações de neurônios biológicos no sistema nervoso humano. Esse modelo foi introduzido por McCulloch e Pitts (1943), que propuseram um modelo simplificado do funcionamento de neurônios, com um exemplo aplicado às operações da lógica proposicional. Modificações desse primeiro modelo levaram ao desenvolvimento das redes neurais *feedforward*, que são foco deste estudo. O desenvolvimento do método de otimização *backpropagation* para redes neurais (RUMELHART; HINTON; WILLIAMS, 1986), resolveu o desafio do treinamento da rede, fazendo com que esse modelo se tornasse escalável, versátil e, consequentemente, poderoso. Por esse motivo, o modelo é de grande interesse para aplicações diversas.

O principal objetivo deste estudo é identificar as expressões que são calculadas durante o treinamento de uma rede neural *feedforward*. Para isso, a estrutura de uma rede neural desse tipo será detalhada e, tomando como base um caso particular, são desenvolvidos os cálculos necessários usando ideias da otimização contínua. Para fins de validação do método proposto, o algoritmo foi implementado e aplicado em experimentos numéricos na base de dados *MNIST*, muito utilizada para testes de algoritmos de aprendizagem de máquina (LECUN; CORTES, 2010). Para esse problema, são fornecidos como dados de entrada imagens de dígitos manuscritos, esperando que o modelo consiga reconhecer o dígito sendo representado e classificá-lo entre os 10 possíveis algarismos. Esse problema não é trivial e desenvolver um modelo matemático para esse classificador é uma tarefa complexa. Por isso, esse problema é utilizado neste estudo como prova de conceito sobre a capacidade de modelagem das redes neurais.

Neste trabalho, o estudo teórico e de aplicações das redes neurais é apresentado em 4 capítulos. No Capítulo 1 são apresentados os conceitos básicos de otimização contínua, em particular o método do gradiente descendente, que servirão como base para o método, e também, as principais ideias e práticas da aprendizagem de máquina. No Capítulo 2 é introduzida a estrutura de uma rede neural artificial apresentando conceitos como

o neurônio artificial, a estrutura em camadas e as funções de ativação, assim como detalhando o objetivo da rede e, por conseguinte, sua formulação como um problema de otimização irrestrita. No Capítulo 3 são apresentados os cálculos a serem utilizados no algoritmo de treinamento de uma rede neural artificial com uma estrutura específica. O algoritmo proposto é implementado, e a descrição dos experimentos numéricos e seus resultados estão apresentados no Capítulo 4. Por fim, é apresentada a conclusão do trabalho desta monografia e uma proposta para os próximos passos do estudo.

Capítulo 1

Conceitos Preliminares

Neste capítulo, apresentamos os principais conceitos básicos a serem utilizados no desenvolvimento do trabalho. Primeiramente, trazemos conceitos de cálculo e otimização contínua na Seção 1.1. Por fim, na Seção 1.2, as principais ideias e conceitos a respeito da aprendizagem de máquina serão introduzidos.

1.1 Otimização contínua

Nesta seção, expomos conceitos necessários de otimização contínua que servirão como base para o desenvolvimento do método a ser aplicado no treinamento das redes neurais. Primeiramente, relembramos a regra da cadeia no contexto de funções de várias variáveis. Em seguida, detalhamos os conceitos que norteiam os algoritmos de otimização contínua. As principais referências utilizadas foram Ribeiro e Karas (2013), Tibshirani (2013) e Nesterov (1998).

1.1.1 Regra da Cadeia

No contexto de funções de várias variáveis em que cada uma dessas variáveis é também uma função de várias variáveis, definimos as derivadas aplicando a regra da cadeia definida a seguir.

Teorema 1.1 (Regra da cadeia)**.** *Seja $u : \mathbb{R}^n \mapsto \mathbb{R}$ uma função de várias variáveis definida como $u = u(x_1, x_2, \ldots, x_n)$, onde cada x_j é também uma função de várias variáveis, cada uma definida como $x_j = x_j(t_1, t_2, \ldots, t_m) : \mathbb{R}^m \mapsto \mathbb{R}$. Então, a derivada parcial de u em relação a algum dos termos t_i é calculada a partir do somatório*

$$\frac{\partial u}{\partial t_i} = \frac{\partial u}{\partial x_1}\frac{\partial x_1}{\partial t_i} + \frac{\partial u}{\partial x_2}\frac{\partial x_2}{\partial t_i} + \ldots + \frac{\partial u}{\partial x_n}\frac{\partial x_n}{\partial t_i} = \sum_{l=1}^{n} \frac{\partial u}{\partial x_l}\frac{\partial x_l}{\partial t_i}.$$

Demonstração Consultar a seção 14.5 do Volume 2 do livro de Cálculo de Stewart (2013). □

1.1.2 Objetivo da otimização contínua

Na otimização contínua, a grande maioria dos problemas é resolvida usando métodos iterativos, seja pela dificuldade, ou até impossibilidade, de encontrar soluções analíticas

para os problemas de minimização (ou maximização). Como os problemas de maximização podem ser facilmente convertidos em problemas de minimização, trataremos apenas de problemas de minimização daqui em diante.

Sendo assim, dada uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$, definimos o problema de minimização irrestrita

$$\min_{x \in \mathbb{R}^n} f(x).$$

O objetivo dos métodos de otimização contínua é encontrar uma aproximação para um ponto x^* que minimiza a função. A maioria dos algoritmos de otimização estão estruturados da seguinte forma: a cada iteração, uma direção de descida é escolhida e avançamos nessa direção para atualizar a aproximação de x^*. Repetindo esse processo, espera-se que, eventualmente, estejamos próximo o suficiente de um ponto de mínimo, de acordo com uma medida definida.

1.1.3 Direção de descida

Para funções $f : \mathbb{R}^n \mapsto \mathbb{R}$ sabemos que o vetor gradiente da função traz a direção de maior crescimento da função. A partir desse resultado, podemos também mostrar que a direção contrária do vetor gradiente traz a direção de maior decrescimento. Sendo assim, quando o vetor gradiente é fácil de calcular, este se torna uma escolha lógica de direção de busca a fim de computar cada passo a ser tomado.

Definição 1.2 (Método do gradiente)**.** *O método do gradiente é um algoritmo iterativo de otimização contínua caracterizado pela escolha da direção oposta do gradiente a cada iteração. Portanto, a cada iteração, a atualização da aproximação é feita de acordo com o esquema:*

$$x \leftarrow x - \alpha \nabla f(x).$$

onde $\alpha > 0 \in \mathbb{R}$ é o tamanho do passo.

1.1.4 Tamanho do passo

Além da direção de busca, a outra escolha a ser feita é quanto caminhar na direção escolhida. Diversos são os métodos para o cálculo deste valor, chamado de tamanho do passo, cada um associado a sua complexidade computacional particular. Alguns exemplos que podemos citar são a busca exata e a busca de Armijo. Contudo, esses métodos adicionam custo computacional ao algoritmo por exigirem uma quantidade adicional de cálculos a cada iteração. Por isso, em situações em que as funções são complexas, por vezes escolhe-se por utilizar um tamanho de passo fixo. A escolha do tamanho de passo é importante para que haja convergência do algoritmo. Por exemplo, sob certas hipóteses, para funções com derivadas Lipschitz contínuas com constante L, podemos mostrar que um tamanho de passo $\alpha \leq \frac{1}{L}$ garante que o método do gradiente gera uma sequência convergente para algum mínimo local (TIBSHIRANI, 2013).

1.2 Aprendizagem de máquina

Segundo Mitchell (1997), a aprendizagem de máquina pode ser definida como a área de conhecimento que tem como objetivo desenvolver sistemas capazes de aprender a exercer tarefas a partir de experiências passadas. Para Géron (2017), a aprendizagem de máquina é a ciência de programar computadores para que possam aprender de forma independente a partir de dados. Nesse sentido, podemos dizer que esta área tem como foco desenvolver algoritmos que sejam capazes de gerar modelos a partir de dados fornecidos, para serem usados posteriormente em outras instâncias de dados, para prever algum valor de interesse. A seguir, apresentamos alguns aspectos da aprendizagem de máquina, assim como métodos e procedimentos comumente utilizados nesta área. As principais referências utilizadas como base para essa seção foram Chollet (2017), Géron (2017), e Swamynathan (2017).

1.2.1 Classificações do tipo de aprendizado

Os principais métodos de aprendizagem de máquina podem ser divididos em dois grandes grupos, de acordo com seu processo de treinamento: aprendizado supervisionado e não supervisionado. A seguir, essas classificações são detalhadas.

O aprendizado supervisionado leva como entrada dados para os quais a saída esperada é conhecida. Com isso, os modelos podem trabalhar em cima dos padrões existentes nas relações entre as entradas e as saídas correspondentes e extrair o conhecimento a partir disso para fazer previsões em dados inéditos. De acordo com o tipo de saída esperada podemos, ainda, classificar esses problemas entre problemas de regressão (para variáveis de resposta contínuas) ou classificação (quando a variável de saída é discreta).

O aprendizado não supervisionado considera conjuntos de dados de entrada não rotulados, ou seja, com saída esperada desconhecida. Neste caso, os algoritmos podem aprender com tendências nos dados em si, encontrando subgrupos nos dados, ou então, encontrando dados discrepantes.

Existem, ainda, outros tipos de aprendizado de máquina, como por exemplo, aprendizado semi-supervisionado e aprendizado por reforço. Para mais detalhes sobre esses modelos, sugere-se consultar o Capítulo 1 de Géron (2017).

O modelo a ser discutido neste trabalho (no caso, a estrutura de uma rede neural *feedforward*), é um modelo de aprendizado supervisionado que será aplicado a um problema de classificação.

1.2.2 Avaliação do modelo

Para avaliar a performance do modelo, precisamos definir métricas que possam ser aplicadas a diferentes tipos de modelos, para que seja possível comparar seus desempenhos. Diversas são as métricas de avaliação de modelos. Neste trabalho empregamos a medida de avaliação chamada acurácia, usada para problemas de classificação. Essa medida pode ser vista como a porcentagem de acertos do modelo em um certo conjunto de dados (seja ele um conjunto de treinamento ou de testes), onde um acerto é definido como o caso em que o modelo classificou corretamente um elemento.

Definição 1.3 (Acurácia)**.** *Seja D o conjunto total de dados para o qual o modelo está sendo avaliado, e $C \subseteq D$ o conjunto dos dados em que o modelo acertou a classificação, definimos a acurácia, a, como:*

$$a = \frac{n(C)}{n(D)}, \tag{1.1}$$

onde $n(\cdot)$ denota o número de elementos em um dado conjunto.

Essas métricas podem ser utilizadas para avaliar a capacidade de generalização de um modelo. Para isso, testamos o modelo, obtido após o treinamento, em um conjunto de dados inéditos. Queremos evitar o fenômeno conhecido como *overfitting*, no qual o modelo adéqua-se muito bem aos dados do conjunto de treino, mas tem uma performance ruim em dados inéditos. Ou seja, em caso de *overfitting*, o modelo, aparentemente, memoriza as respostas esperadas para os dados conhecidos mas não consegue generalizar as observações para dados desconhecidos. Esse fenômeno é caracterizado por uma boa métrica de performance no conjunto de dados de treino e uma métrica muito pior em conjuntos de dados inéditos. Existem métodos para minimizar esse fenômeno, como métodos de regularização, redução de dimensionalidade, entre outros.

1.2.3 Hiperparâmetros

Hiperparâmetros são parâmetros usados pelo algoritmo de aprendizagem em si, não do modelo. Ou seja, são parâmetros utilizados pelo algoritmo para desenvolver o modelo de aprendizagem de máquina. Por exemplo, como veremos a seguir, no contexto de redes neurais, alguns hiperparâmetros são o tamanho das camadas intermediárias e o tamanho do passo. Os hiperparâmetros devem ser definidos pelo usuário antes do treinamento do modelo, e devem manter-se constantes durante todo o processo de treinamento do modelo pelo algoritmo de aprendizagem. Encontrar bons hiperparâmetros é uma parte importante do desenvolvimento de modelos de aprendizagem de máquina eficientes. Esse processo é conhecido na literatura como *hyperparameter tuning*, ou otimização de hiperparâmetros (*hyperparameter optimization*, ou HPO). Um dos métodos mais amplamente utilizados para esse propósito é o *grid search*, em que o usuário escolhe alguns conjuntos de valores para cada um dos hiperparâmetros e a performance do modelo é avaliada para cada combinação de hiperparâmetros possível, a fim de encontrar a combinação com melhor desempenho.

Capítulo 2

Redes Neurais Artificiais

Nesta seção, serão introduzidos os principais conceitos de redes neurais artificiais (ou simplesmente redes neurais) necessários para este trabalho. Serão detalhados os elementos que compõem a estrutura da rede, assim como sua modelagem e sua motivação matemática.

As principais referências bibliográficas usadas como base deste capítulo são: Chollet (2017), Goodfellow, Bengio e Courville (2016), Lecun et al. (1998), Nielsen (2015), Rumelhart, Hinton e Williams (1986), Rumelhart et al. (1995)

2.1 Introdução à estruturação das redes neurais

As redes neurais podem ser vistas como uma malha interconectada de neurônios associados a pesos, vieses e funções de ativação. Vejamos em detalhes cada uma dessas definições a seguir.

2.1.1 Neurônios

Primeiramente, introduzimos a ideia de um neurônio artificial. O primeiro tipo de neurônio artificial, chamado *perceptron*, foi introduzido por Rosenblatt (1958), ao tentar encontrar um modelo para explicar como a informação captada pelos sentidos é armazenada e utilizada pelo cérebro humano. Este modelo de neurônio considera um vetor X de entradas binárias, $x_i \in \{0, 1\}$, cada x_i associado a um peso $w_i \in \mathbb{R}$, para produzir uma saída binária, $p(X) \in \{0, 1\}$, considerando uma tolerância $b \in \mathbb{R}$. Definimos a saída $p(X)$:

$$p(X) = \begin{cases} 0, & \text{se } \sum_{i=1}^{n} x_i w_i + b \leq 0 \\ 1, & \text{se } \sum_{i=1}^{n} x_i w_i + b > 0 \end{cases}, \tag{2.1}$$

onde $n \in \mathbb{N}$ é a dimensão do vetor de entradas, X.

Esse modelo foi, então, modificado para considerar saídas contínuas, para que métodos de otimização contínua pudessem ser utilizados. A ideia é que pequenas perturbações nos valores dos pesos w_i gerem pequenas alterações no valor de $p(X)$. Com o *perceptron* da maneira que foi definido em (2.1) isso não acontece. Uma pequena perturbação pode causar uma alteração no estado qualitativo do *perceptron* mudando seu estado de desligado ($p(X) = 0$) para ligado ($p(X) = 1$), ou vice-versa.

Introduzimos, então, a ideia de neurônios com funções de ativação. De forma semelhante aos *perceptrons*, esses neurônios também têm pesos $w_i \in \mathbb{R}$ associados às entradas,

assim como a variável $b \in \mathbb{R}$, agora denominada viés. A diferença é que as entradas x_i e saídas $p(X)$ podem assumir valores pertencentes ao conjunto dos reais. Além disso, associamos ao neurônio uma função de ativação $f : \mathbb{R} \mapsto \mathbb{R}$. Essa função, a princípio, deve ser diferenciável. A saída desse novo modelo de neurônio, $p(X) \in \mathbb{R}$, está definida como

$$p(X) = f\left(\sum_{i=1}^{n} x_i w_i + b\right). \tag{2.2}$$

Esse neurônio é mais comumente utilizado em aplicações modernas e será utilizado na modelagem deste trabalho.

2.1.2 Estrutura das redes neurais

Considere a estrutura representada na Figura 2.1. Ela ilustra uma estrutura generalizada de uma rede neural, com alguns neurônios de cada camada ocultos, apenas para representação visual. Acima de cada ligação entre uma camada e a próxima, denotamos as matrizes de pesos e vetores de vieses associados, para os quais detalhamos a notação num outro momento.

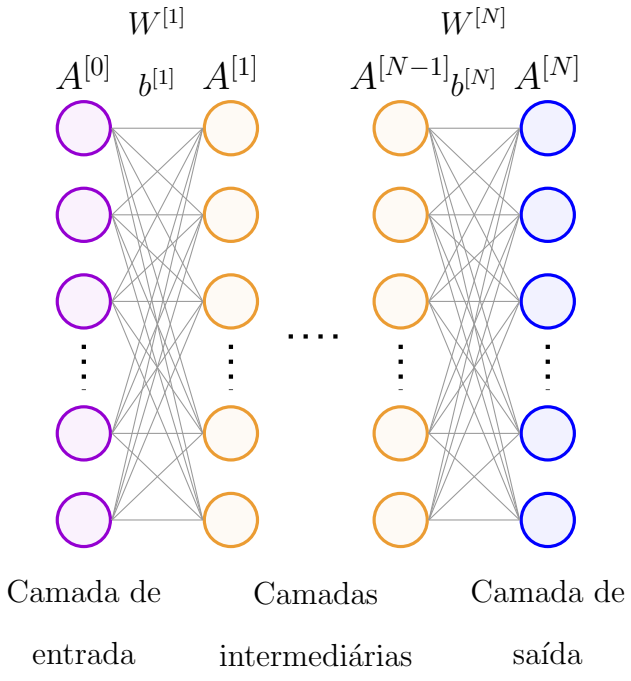

Figura 2.1: Modelo geral de uma rede neural. Fonte: Autoria própria.

Definimos a primeira camada $A^{[0]}$ como um conjunto de neurônios, representado como um vetor de números reais. Essa primeira camada é considerada a camada de entrada. No problema de aprendizado supervisionado, essa camada recebe os dados de entrada para os quais queremos calcular a resposta esperada. A última camada $A^{[N]}$ é denominada a camada de saída que, após o treinamento da rede, deve aproximar a saída

esperada para cada entrada do conjunto de dados. Sendo X o vetor de dados de entrada, podemos definir a notação $F(X) = A^{[N]}$ para denotar a saída calculada pela rede para um certo dado de entrada. As outras camadas, denotadas como camadas intermediárias na Figura 2.1, permitem que a rede generalize um leque maior de problemas.

A seguir definimos a notação a ser utilizada neste trabalho para fazer referência às estruturas que compõem a rede:

- Vetores-camada $A^{[i]}$, com $i \in \mathbb{N}, i \in \{0, 1, 2, ..., N\}$, sendo i o número da camada correspondente e $N \in \mathbb{N}$ o número total de camadas no modelo.

- Matrizes de pesos $W^{[i]}$, com $i \in \{1, 2, ..., N\}$, que relacionam os neurônios da camada $i-1$ aos neurônios da camada i. A matriz deve ter a dimensão apropriada para permitir as multiplicações necessárias, ou seja, deve ter o mesmo número de colunas quanto o número de neurônios na camada $i-1$ e o mesmo número de linhas quanto a camada i. Cada elemento $w_{jk}^{[i]}$ relaciona o k-ésimo neurônio da camada $i-1$ ao j-ésimo neurônio da camada i.

- Vetores de vieses $b^{[i]}$, com $i \in \{1, 2, ..., N\}$, relacionados aos neurônios da camada i. O vetor deve ter o mesmo número de elementos quanto o número de neurônios na camada i. Neste vetor, cada elemento $b_j^{[i]}$ representa o viés do j-ésimo neurônio da camada i.

Para facilitar o desenvolvimento futuro das expressões a serem utilizadas no treinamento da rede, introduzimos nessa modelagem camadas auxiliares na estrutura, as camadas $Z^{[i]}$, associadas a cada camada $A^{[i]}, i \in \{1, 2, ..., N\}$, como ilustrado na Figura 2.2. Cada $f_i : \mathbb{R} \mapsto \mathbb{R},\ i = 1, 2, ..., N$ é uma função contínua não-linear, chamada de função de ativação. Por esse motivo em algumas referências a camada auxiliar $Z^{[i]}$ é chamada de camada não ativada da camada i e a camada $A^{[i]}$ é chamada de camada ativada da camada i. Veremos adiante que o papel das funções de ativação é de garantir a não-linearidade do modelo, o que permite que essa estrutura modele problemas diversos.

Dada essa estrutura, o relacionamento entre as camadas é definido da seguinte forma:

$$Z^{[i]} = W^{[i]} A^{[i-1]} + b^{[i]},\ i = 1, 2, ..., N, \tag{2.3}$$

$$A^{[i]} = f_i(Z^{[i]}),\ i = 1, 2, ..., N. \tag{2.4}$$

Na Eq. (2.4) fazemos um abuso de notação. Anteriormente, definimos as funções f_i como funções de uma variável real: $f_i : \mathbb{R} \mapsto \mathbb{R}$. Na Eq. (2.4) ao aplicar a função a um vetor, estamos, na verdade, aplicando a função em cada elemento desse vetor. Sendo assim, definimos:

$$f_i(Z^{[i]}) := \begin{bmatrix} f_i(z_1^{[i]}) \\ f_i(z_2^{[i]}) \\ \vdots \\ f_i(z_m^{[i]}) \end{bmatrix},\ i = 1, 2, ..., N, \tag{2.5}$$

onde $Z^{[i]} \in \mathbb{R}^m$ e cada $z_j^{[i]} \in \mathbb{R}$ é o j-ésimo elemento do vetor $Z^{[i]}$.

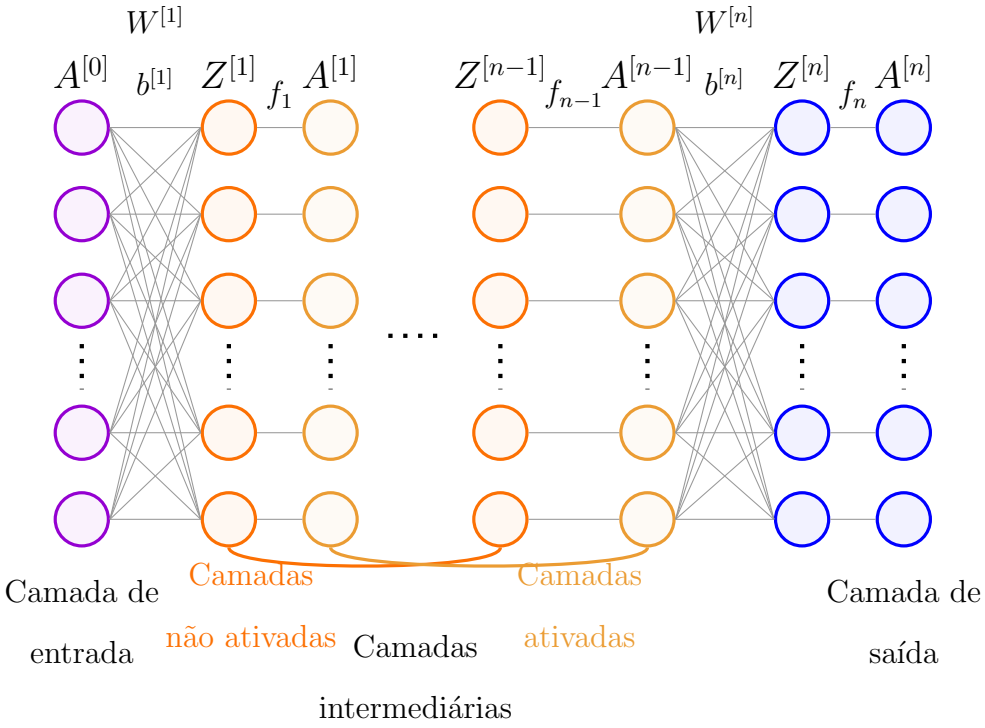

Figura 2.2: Modelo geral de uma rede neural, introduzindo as camadas auxiliares. Fonte: Autoria própria.

Neste modelo, consideramos que a saída de uma camada, o vetor $A^{[i]}$, é a entrada da próxima camada, $Z^{[i+1]}$. Este modelo considera apenas esse movimento "linear", onde a saída de uma camada não pode ser uma entrada dela mesma. Esse modelo de redes neurais é chamado na literatura de redes neurais *feedforward*. Outros modelos de redes neurais como as redes neurais recorrentes ou as redes neurais convolucionais podem ser implementados, mas não são o foco deste trabalho. Para mais detalhes sobre outros modelos de redes neurais consultar o material em Chollet (2017).

Pelas Equações (2.3), (2.4) e (2.5) vemos a importância da inclusão das funções de ativação. Caso não houvessem funções de ativação não lineares entre as camadas, a rede neural completa teria relações puramente lineares entre as camadas. Pelas propriedades das transformações lineares, a passagem da primeira camada à última poderia ser escrita como uma única relação linear do tipo $A^{[N]} = W^{\star}A^{[0]} + b^{\star}$, tornando as camadas intermediárias dispensáveis. Ademais, o modelo só conseguiria um bom ajuste para casos em que a relação entre os dados de entrada e as respectivas saídas esperadas é linear. Isso reduz o leque de problemas para os quais esse modelo poderia ser útil. Como queremos um modelo capaz de modelar problemas diversos, a restrição da linearidade não é vantajosa. Assim, vemos que as funções de ativação têm um papel importante para garantir a não linearidade da relação entre camadas, permitindo, assim, que as redes neurais sejam capazes de generalizar mais tipos diferentes de problemas.

Até agora, temos definidos os dados de entrada, a relação entre uma camada e a próxima e o vetor que aproxima a saída esperada. Porém, ainda não especificamos como

devem ser definidos os valores nas matrizes de pesos e nos vetores de vieses. Esses são os parâmetros que podem ser modificados durante o treinamento da rede neural para que seu desempenho seja adequado, de acordo com o conjunto de dados disposto no problema.

Para avaliar o desempenho da rede neural, precisamos definir uma função para quantificar o erro cometido por esta. Para isso, introduzimos o conceito de uma função custo. A função custo, no contexto de aprendizado supervisionado, leva em consideração a saída esperada conhecida para cada dado de entrada e a saída obtida pelo modelo e quantifica qual foi o erro. Idealmente, essa função assume valores reais não negativos e apenas é igual a 0 quando a saída fornecida pelo modelo é idêntica à saída esperada. Sendo D_i um elemento do conjunto de dados, definido pela tupla $D_i = (X_i, Y_i)$, na qual X_i é um dado de entrada a ser aplicado na rede, entrando como a camada inicial $A^{[0]}$, Y_i o vetor da saída esperada conhecida correspondente e $F(X_i) = A^{[N]}$ a saída obtida pelo modelo para essa entrada específica, definimos a função custo deste dado em específico como

$$C_{D_i} = C(Y_i, F(X_i)). \tag{2.6}$$

Em algumas referências, a função custo aplicada a uma instância dos dados de entrada também é conhecida como função de perda. Para avaliar o desempenho do modelo no conjunto completo de dados de treinamento, definimos o valor da função custo do modelo para o conjunto de dados de treinamento completo.

Definição 2.1 (Função custo do modelo)**.** *Seja $D = \{D_1, D_2, D_3, ..., D_m\}$ o conjunto de dados a ser utilizado no treinamento de uma rede neural, onde cada D_i é uma tupla $D_i = (X_i, Y_i)$, com $X_i \in \mathbb{R}^o$ o dado de entrada e $Y_i \in \mathbb{R}^n$ a saída esperada correspondente, sendo $o, n \in \mathbb{N}$. Seja $C_{X_i} \in \mathbb{R}$ o valor da função custo para o dado D_i, definida como na Eq.* (2.6)*.*

Definimos a função custo do modelo para o conjunto D completo, $C \in \mathbb{R}$, como sendo a média aritmética dos valores da função custo calculados para cada elemento de D

$$C = \frac{1}{m} \sum_{i=1}^{m} C_{D_i}$$

Uma possível escolha de função custo é o erro quadrático médio, descrito a seguir.

Definição 2.2 (Erro quadrático médio)**.** *Seja $D = \{D_1, D_2, D_3, ..., D_m\}$ o conjunto de dados a ser utilizado no treinamento da rede neural, como na Definição 2.1. Definimos a função perda para o erro quadrático como*

$$C_{D_i} = C(Y_i, F(X_i)) = \frac{1}{2}\|F(X_i) - Y_i\|^2,$$

onde $\|\cdot\|$ é a norma euclidiana do vetor.

Para o conjunto de treino completo de dados, então, o erro quadrado médio é definido como

$$C = \frac{1}{m} \sum_{i=1}^{m} C_{D_i} = \frac{1}{2m} \sum_{i=1}^{m} \|F(X_i) - Y_i\|^2.$$

2.2 Formulação como Problema de Minimização

Como visto anteriormente, o valor da função custo será 0 apenas se o modelo consegue calcular corretamente a saída esperada para cada um dos elementos do conjunto de dados e aumenta conforme a saída calculada se distancia da saída esperada. Sendo assim, podemos formular o problema do treinamento de uma rede neural como um problema de minimização irrestrita. Vimos que as entradas, as saídas esperadas e o relacionamento entre cada camada são definidos pela estruturação da rede e o conjunto de dados de treinamento fornecidos. O que é possível alterar são os parâmetros contidos nas matrizes de pesos e vetores de vieses, $W^{[j]}$ e $b^{[j]}$, $j = 1, 2, ..., N$. Sendo assim, o problema de treinar a rede neural se resume a encontrar os valores para esses parâmetros que permitem minimizar o erro das predições, ou seja, uma minimização irrestrita do valor da função custo em relação às matrizes de peso e vetores de viés:

$$\min_{W,\, b} \quad C = \frac{1}{m} \sum_{i=1}^{m} C_{D_i}. \tag{2.7}$$

Tendo a formulação do problema desta forma, podemos aplicar métodos de otimização para encontrar os parâmetros ótimos. A abordagem a ser utilizada neste trabalho será o algoritmo conhecido na literatura de redes neurais como *backpropagation*, ou retropropagação, que nada mais é que uma aplicação do método do gradiente, como na Definição 1.2. Essa nomenclatura advém do fato de que a derivada parcial pode ser vista como a contribuição de cada parâmetro no erro total. Por conta da estrutura da rede, a relação entre as camadas e a regra da cadeia, ao calcular cada derivada parcial vemos uma propagação do erro a partir da camada mais final, para as camadas iniciais. Assim, ao fazer os cálculos para a implementação computacional, podemos aproveitar os cálculos referentes às últimas camadas para os cálculos referentes às camadas iniciais.

Precisamos, então, encontrar a derivada parcial da função custo em relação a cada um dos elementos dos vetores e matrizes de pesos e vieses. A cada iteração, a atualização dos parâmetros é feita da forma:

$$\begin{aligned} w_{jk}^{[i]} &\leftarrow w_{jk}^{[i]} - \alpha \frac{\partial C}{\partial w_{jk}^{[i]}}, \\ b_{j}^{[i]} &\leftarrow b_{j}^{[i]} - \alpha \frac{\partial C}{\partial b_{j}^{[i]}}, \qquad i \in \{1, 2, ..., N\}, \end{aligned} \tag{2.8}$$

onde o tamanho do passo, $\alpha > 0 \in \mathbb{R}$, é um hiperparâmetro a ser definido pelo usuário e mantém-se fixo durante todas as iterações. No contexto das redes neurais, a maioria das abordagens clássicas utiliza um tamanho de passo fixo durante todo o treinamento da rede. Isso ocorre pois as redes neurais podem se tornar muito complexas conforme adicionamos camadas intermediárias e aumentamos o número de neurônios em cada uma delas. Sendo assim, o custo computacional aumenta e manter um passo fixo permite diminuir a quantidade de operações necessárias para a execução do algoritmo, viabilizando, então, a utilização do método.

Capítulo 3

Modelagem

A presente seção traz o detalhamento algébrico referente aos cálculos das derivadas a serem utilizadas no treinamento da rede neural artificial. É importante salientar que as contas aqui apresentadas são em sua maioria autorais, visto que não conseguimos encontrar na literatura os cálculos com o nível de profundidade que julgamos necessário para este trabalho.

Nas seções anteriores, apresentamos a estrutura da rede considerando um número arbitrário N de camadas. Para este trabalho, propomos uma estrutura com $N = 2$, como ilustrada na Figura 3.1. Essa estrutura será utilizada no detalhamento dos cálculos.

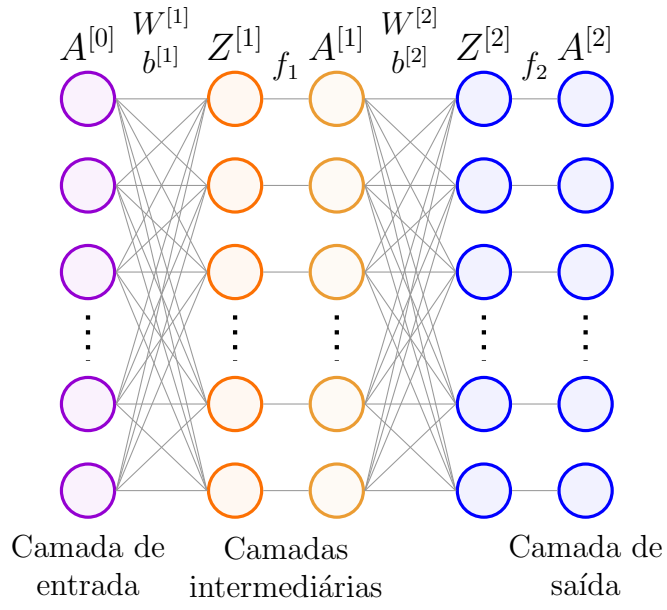

Figura 3.1: Modelo específico da rede neural a ser modelada neste trabalho. Fonte: Autoria própria.

Primeiramente, apresentamos um caso particular para calcular a saída para um único elemento de D, ou seja, a entrada é um vetor $X \in \mathbb{R}^o$ e a saída esperada é um vetor $Y \in \mathbb{R}^n$. Definimos, também, uma camada intermediária com p neurônios. A seguir, detalhamos a notação a ser utilizada para denotar os elementos das camadas, das

matrizes de pesos, e dos vetores de vieses nas Tabelas 3.1 e 3.2.

Dados	Descrição	Dimensão
$X = A^{[0]}$	Vetor de dados de entrada	$\mathbb{R}^o$
Y	Vetor de saída esperada	$\mathbb{R}^n$

Tabela 3.1: Notação usada para os dados fornecidos à rede.

Parâmetro	Dimensão
$W^{[1]}$	$W^{[1]} = \left[w_{ij}^{[1]}\right] \in \mathbb{R}^{p\times o}$
$b^{[1]}$	$b^{[1]} = \left[b_{i}^{[1]}\right] \in \mathbb{R}^{p}$
$W^{[2]}$	$W^{[2]} = \left[w_{ij}^{[2]}\right] \in \mathbb{R}^{n\times p}$
$b^{[2]}$	$b^{[2]} = \left[b_{i}^{[2]}\right] \in \mathbb{R}^{n}$

Tabela 3.2: Notação usada para os parâmetros da rede.

Tendo definida a notação a ser utilizada, podemos definir a relação entre as camadas na Tabela 3.3, seguindo a definição das Equações (2.3) e (2.4).

Camada	Valor	Elementos
$Z^{[1]} \in \mathbb{R}^p$	$Z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$	$z_i^{[1]} = \sum_{k=1}^{o} w_{ik}^{[1]} a_k^{[0]} + b_i^{[1]},\ i = 1, 2, \cdots, p$
$A^{[1]} \in \mathbb{R}^p$	$A^{[1]} = f_1(Z^{[1]})$	$a_i^{[1]} = f_1(z_i^{[1]}),\ i = 1, 2, \cdots, p$
$Z^{[2]} \in \mathbb{R}^n$	$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$	$z_i^{[2]} = \sum_{k=1}^{p} w_{ik}^{[2]} a_k^{[1]} + b_i^{[2]},\ i = 1, 2, \cdots, n$
$A^{[2]} \in \mathbb{R}^n$	$A^{[2]} = f_2(Z^{[2]})$	$a_i^{[2]} = f_2(z_i^{[2]}),\ i = 1, 2, \cdots, n$

Tabela 3.3: Relação entre as camadas de uma rede neural com $N = 2$.

Com relação às funções f_1 e f_2 listadas na estrutura da Figura 3.1, para esse modelo estamos considerando as seguintes escolhas de funções de ativação, com suas respectivas derivadas: a tangente hiperbólica e a função sigmoide, dadas por

$$f_1(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad f_1'(x) = \frac{df_1}{dx}(x) = 1 - \tanh^2(x), \tag{3.1}$$

$$f_2(x) = \sigma(x) = \frac{1}{1+e^{-x}}, \quad f_2'(x) = \frac{df_2}{dx}(x) = \sigma(x)(1-\sigma(x)). \tag{3.2}$$

A motivação por trás dessas escolhas está no fato de que essas funções são limitadas na reta dos reais, e com isso, garantimos que a magnitude dos valores na rede neural fique controlada, evitando *overflow* numérico durante o momento da implementação computacional.

Em relação à função custo, será utilizado o erro quadrático médio, segundo a Definição 2.2. Como estamos, primeiramente, modelando para apenas um dado de entrada, o custo total da rede é igual à função custo aplicada a apenas aquele elemento, ou seja: $C = C_D$, a função perda. Utilizando a notação definida nesta seção, abrimos a expressão para termos o valor de C em função dos elementos dos vetores da saída calculada $F(X) = A^{[2]}$ e da saída esperada Y:

$$C = C_D = \frac{1}{2}\left\|A^{[2]} - Y\right\|^2 = \frac{1}{2}\sum_{i=1}^{n}(a_i^{[2]} - y_i)^2 \tag{3.3}$$

3.1 Cálculo das expressões para otimização da rede

O processo de treinamento da rede equivale à otimização dos parâmetros, para encontrar parâmetros que minimizam o valor da função custo. Para isso, precisamos calcular a derivada da função custo em relação a cada elemento de $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, e $b^{[2]}$. Ou seja, precisamos encontrar $\dfrac{\partial C}{\partial w_{ij}^{[2]}}, \dfrac{\partial C}{\partial b_i^{[2]}}, \dfrac{\partial C}{\partial w_{ij}^{[1]}}, \dfrac{\partial C}{\partial b_i^{[1]}}$.

Derivação em relação a $A^{[2]}$

Olhando para a expressão na Eq. (3.3) fica evidente que a primeira derivada parcial a ser calculada é em relação aos elementos da camada de saída $A^{[2]}$. Tome $a_i^{[2]}$, para $i = 1, 2, \ldots, n$, um elemento qualquer do vetor. A derivada parcial da função custo em relação a este elemento será

$$\begin{aligned}\frac{\partial C}{\partial a_i^{[2]}} &= \frac{\partial}{\partial a_i^{[2]}}\left(\frac{1}{2}\sum_{g=1}^{n}(a_g^{[2]} - y_g)^2\right), \\ &= \frac{1}{2}\sum_{g=1}^{n}\frac{\partial}{\partial a_i^{[2]}}\left[(a_g^{[2]} - y_g)^2\right].\end{aligned}$$

Vamos avaliar o valor de $\dfrac{\partial(a_g^{[2]} - y_g)^2}{\partial a_i^{[2]}}$ para cada caso:

$$\frac{\partial(a_g^{[2]} - y_g)^2}{\partial a_i^{[2]}} = \begin{cases} 2(a_i^{[2]} - y_i), & \text{se } g = i, \\ 0, & \text{caso contrário.} \end{cases}$$

Então, voltando na definição de $\frac{\partial C}{\partial a_i^{[2]}}$ acima, temos:

$$\begin{aligned}\frac{\partial C}{\partial a_i^{[2]}} &= \frac{1}{2}\sum_{g=1}^{n}\frac{\partial}{\partial a_i^{[2]}}\left((a_g^{[2]}-y_g)^2\right),\\ &= \frac{1}{2}\left(2(a_i^{[2]}-y_i)\right),\\ \frac{\partial C}{\partial a_i^{[2]}} &= a_i^{[2]}-y_i.\end{aligned}$$

O processo utilizado para encontrar as próximas derivadas é semelhante ao utilizado acima. Por isso, alguns trechos parecem se repetir. Contudo, para garantir a completude das informações contidas nesse trabalho, escolhemos por incluir o desenvolvimento detalhado de todas as contas. Os resultados obtidos nesta seção serão reunidos na Tabela 3.4.

Derivação em relação a $Z^{[2]}$

O valor de C depende de $a_i^{[2]}$, com $i = 1, 2, ..., n$. Cada $a_i^{[2]}$ pode depender de $z_k^{[2]}$, $k = 1, 2, ..., n$. Assim,

$$\frac{\partial C}{\partial z_i^{[2]}} = \frac{\partial C}{\partial a_1^{[2]}}\frac{\partial a_1^{[2]}}{\partial z_i^{[2]}} + \frac{\partial C}{\partial a_2^{[2]}}\frac{\partial a_2^{[2]}}{\partial z_i^{[2]}} + \cdots + \frac{\partial C}{\partial a_n^{[2]}}\frac{\partial a_n^{[2]}}{\partial z_i^{[2]}} = \sum_{g=1}^{n}\frac{\partial C}{\partial a_g^{[2]}}\frac{\partial a_g^{[2]}}{\partial z_i^{[2]}}.$$

Se cada $a_g^{[2]}$ depende apenas de elementos da posição g de $Z^{[2]}$, como é o caso da modelagem proposta, onde f_2 é uma função de uma variável aplicada elemento a elemento em $Z^{[2]}$ para gerar $A^{[2]}$, temos:

$$\frac{\partial a_g^{[2]}}{\partial z_i^{[2]}} = \begin{cases}\dfrac{\partial a_g^{[2]}}{\partial z_i^{[2]}}, & \text{se } g = i,\\ 0, & \text{caso contrário.}\end{cases}$$

Logo,

$$\frac{\partial C}{\partial z_i^{[2]}} = \sum_{g=1}^{n}\frac{\partial C}{\partial a_g^{[2]}}\frac{\partial a_g^{[2]}}{\partial z_i^{[2]}} = \frac{\partial C}{\partial a_i^{[2]}}\frac{\partial a_i^{[2]}}{\partial z_i^{[2]}}.$$

Pela definição na Tabela 3.3, temos que $a_i^{[2]} = f_2(z_i^{[2]})$. Logo, a derivada parcial de $a_i^{[2]}$ em relação a $z_i^{[2]}$ fica:

$$\frac{\partial a_i^{[2]}}{\partial z_i^{[2]}} = \frac{\partial f_2(z_i^{[2]})}{\partial z_i^{[2]}} = f_2'(z_i^{[2]}).$$

E, portanto,

$$\frac{\partial C}{\partial z_i^{[2]}} = \frac{\partial C}{\partial a_i^{[2]}} f_2'(z_i^{[2]}).$$

Derivação em relação a $W^{[2]}$

Seja $w_{ij}^{[2]}$ um elemento qualquer de $W^{[2]}$, com $i = 1, 2, ..., n$ e $j = 1, 2, ..., p$. Pela regra da cadeia temos que:

$$\frac{\partial C}{\partial w_{ij}^{[2]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[2]}} \frac{\partial z_g^{[2]}}{\partial w_{ij}^{[2]}}. \tag{3.4}$$

Portanto, para encontrar $\frac{\partial C}{\partial w_{ij}^{[2]}}$, precisamos, primeiramente encontrar as derivadas parciais $\frac{\partial z_g^{[2]}}{\partial w_{ij}^{[2]}}$. Pela definição temos que $z_g^{[2]} = \sum_{k=1}^{p} w_{gk}^{[2]} a_k^{[1]} + b_g^{[2]}$. Sendo assim:

$$\begin{aligned}
\frac{\partial z_g^{[2]}}{\partial w_{ij}^{[2]}} &= \frac{\partial}{\partial w_{ij}^{[2]}} \left(\sum_{k=1}^{p} w_{gk}^{[2]} a_k^{[1]} + b_g^{[2]} \right), \\
&= \frac{\partial}{\partial w_{ij}^{[2]}} \left(\sum_{k=1}^{p} w_{gk}^{[2]} a_k^{[1]} \right) + \frac{\partial b_g^{[2]}}{\partial w_{ij}^{[2]}}, \\
&= \sum_{k=1}^{p} \frac{\partial}{\partial w_{ij}^{[2]}} \left(w_{gk}^{[2]} a_k^{[1]} \right), \\
&= \frac{\partial}{\partial w_{ij}^{[2]}} \left(w_{gj}^{[2]} a_j^{[1]} \right) = \begin{cases} a_j^{[1]}, & \text{se } g = i, \\ 0, & \text{caso contrário.} \end{cases}
\end{aligned}$$

Voltando em Eq. (3.4), temos:

$$\begin{aligned}
\frac{\partial C}{\partial w_{ij}^{[2]}} &= \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[2]}} \frac{\partial z_g^{[2]}}{\partial w_{ij}^{[2]}}, \\
&= \frac{\partial C}{\partial z_i^{[2]}} \frac{\partial z_i^{[2]}}{\partial w_{ij}^{[2]}}, \\
\frac{\partial C}{\partial w_{ij}^{[2]}} &= \frac{\partial C}{\partial z_i^{[2]}} a_j^{[1]}.
\end{aligned}$$

Derivação em relação a $b^{[2]}$

O cálculo das derivadas parciais em relação ao vetor de vieses $b^{[2]}$ é feito de forma similar ao que foi feito para $W^{[2]}$. Pela regra da cadeia temos que, para um elemento arbitrário de $b^{[2]}$, a derivada parcial de C é:

$$\frac{\partial C}{\partial b_i^{[2]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[2]}} \frac{\partial z_g^{[2]}}{\partial b_i^{[2]}}. \tag{3.5}$$

Portanto, para encontrar $\frac{\partial C}{\partial b_i^{[2]}}$, precisamos, primeiramente encontrar as derivadas parciais $\frac{\partial z_g^{[2]}}{\partial b_i^{[2]}}$. Pela definição temos que $z_g^{[2]} = \sum_{k=1}^{p} w_{gk}^{[2]} a_k^{[1]} + b_g^{[2]}$. Sendo assim, temos:

$$\begin{aligned}
\frac{\partial z_g^{[2]}}{\partial b_i^{[2]}} &= \frac{\partial}{\partial b_i^{[2]}}\left(\sum_{k=1}^{p} w_{gk}^{[2]} a_k^{[1]} + b_g^{[2]}\right), \\
&= \frac{\partial}{\partial b_i^{[2]}}\left(\sum_{k=1}^{p} w_{gk}^{[2]} a_k^{[1]}\right) + \frac{\partial b_g^{[2]}}{\partial b_i^{[2]}}, \\
&= \frac{\partial b_g^{[2]}}{\partial b_i^{[2]}} = \begin{cases} 1, & \text{se } g = i, \\ 0, & \text{caso contrário.} \end{cases}
\end{aligned}$$

Voltando em Eq. (3.5), temos:

$$\begin{aligned}
\frac{\partial C}{\partial b_i^{[2]}} &= \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[2]}} \frac{\partial z_g^{[2]}}{\partial b_i^{[2]}}, \\
&= \frac{\partial C}{\partial z_i^{[2]}} \frac{\partial z_i^{[2]}}{\partial b_i^{[2]}}, \\
\frac{\partial C}{\partial b_i^{[2]}} &= \frac{\partial C}{\partial z_i^{[2]}}.
\end{aligned}$$

Derivação em relação a $A^{[1]}$

Temos que a derivada parcial de C em relação aos elementos de $A^{[1]}$ pode ser definida como:

$$\frac{\partial C}{\partial a_i^{[1]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[2]}} \frac{\partial z_g^{[2]}}{\partial a_i^{[1]}}. \tag{3.6}$$

Calculamos, então, as derivadas parciais dos elementos de $Z^{[2]}$ em relação aos elementos individuais de $A^{[1]}$:

$$\begin{aligned}
\frac{\partial z_g^{[2]}}{\partial a_i^{[1]}} &= \frac{\partial}{\partial a_i^{[1]}}\left(\sum_{k=1}^{p} w_{gk}^{[2]} a_k^{[1]} + b_g^{[2]}\right), \\
&= \frac{\partial}{\partial a_i^{[1]}}\left(\sum_{k=1}^{p} w_{gk}^{[2]} a_k^{[1]}\right) + \frac{\partial b_g^{[2]}}{\partial a_i^{[1]}}, \\
&= \sum_{k=1}^{p} \frac{\partial}{\partial a_i^{[1]}}\left(w_{gk}^{[2]} a_k^{[1]}\right), \\
&= \frac{\partial}{\partial a_i^{[1]}} w_{gi}^{[2]} a_i^{[1]} = w_{gi}^{[2]}.
\end{aligned}$$

Voltando em (3.6) temos:

$$\frac{\partial C}{\partial a_i^{[1]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[2]}} \frac{\partial z_g^{[2]}}{\partial a_i^{[1]}},$$
$$\frac{\partial C}{\partial a_i^{[1]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[2]}} w_{gi}^{[2]}.$$

Derivação em relação a $Z^{[1]}$

Pela definição temos que:

$$\frac{\partial C}{\partial z_i^{[1]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial a_g^{[1]}} \frac{\partial a_g^{[1]}}{\partial z_i^{[1]}}. \tag{3.7}$$

Seguindo o processo de cálculo utilizado nas seções anteriores, calculamos, agora, as derivadas parciais dos elementos de $A^{[1]}$ em relação aos elementos de $Z^{[1]}$. De acordo com a Tabela 3.3, os elementos de $A^{[1]}$ são definidos como $a_i^{[1]} = f_1(z_i^{[1]})$. Logo, a derivada parcial $\frac{\partial a_g^{[1]}}{\partial z_i^{[1]}}$ será:

$$\frac{\partial a_g^{[1]}}{\partial z_i^{[1]}} = \begin{cases} f_1'(z_i^{[1]}), & \text{se } i = g, \\ 0, & \text{caso contrário.} \end{cases}$$

Sendo assim, a derivada da Eq. (3.7) fica:

$$\begin{aligned} \frac{\partial C}{\partial z_i^{[1]}} &= \sum_{g=1}^{n} \frac{\partial C}{\partial a_g^{[1]}} \frac{\partial a_g^{[1]}}{\partial z_i^{[1]}}, \\ &= \frac{\partial C}{\partial a_i^{[1]}} \frac{\partial a_i^{[1]}}{\partial z_i^{[1]}}, \\ \frac{\partial C}{\partial z_i^{[1]}} &= \frac{\partial C}{\partial a_i^{[1]}} f_1'(z_i^{[1]}). \end{aligned}$$

Derivação em relação a $W^{[1]}$

Pela regra da cadeia temos que:

$$\frac{\partial C}{\partial w_{ij}^{[1]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[1]}} \frac{\partial z_g^{[1]}}{\partial w_{ij}^{[1]}}. \tag{3.8}$$

Portanto, para encontrar $\frac{\partial C}{\partial w_{ij}^{[1]}}$, precisamos, primeiramente encontrar as derivadas parciais $\frac{\partial z_g^{[1]}}{\partial w_{ij}^{[1]}}$. Pela definição temos que $z_g^{[1]} = \sum_{k=1}^{p} w_{gk}^{[1]} a_k^{[0]} + b_g^{[1]}$. Sendo assim, temos:

$$
\begin{aligned}
\frac{\partial z_g^{[1]}}{\partial w_{ij}^{[1]}} &= \frac{\partial}{\partial w_{ij}^{[1]}}\left(\sum_{k=1}^{p} w_{gk}^{[1]} a_k^{[0]} + b_g^{[1]}\right), \\
&= \frac{\partial}{\partial w_{ij}^{[1]}}\left(\sum_{k=1}^{p} w_{gk}^{[1]} a_k^{[0]}\right) + \frac{\partial b_g^{[1]}}{\partial w_{ij}^{[1]}}, \\
&= \sum_{k=1}^{p} \frac{\partial}{\partial w_{ij}^{[1]}}\left(w_{gk}^{[1]} a_k^{[0]}\right), \\
&= \frac{\partial}{\partial w_{ij}^{[1]}}\left(w_{gj}^{[1]} a_j^{[0]}\right) = \begin{cases} a_j^{[0]}, & \text{se } i = g, \\ 0, & \text{caso contrário.} \end{cases}
\end{aligned}
$$

Voltando em Eq. (3.8), temos:

$$
\begin{aligned}
\frac{\partial C}{\partial w_{ij}^{[1]}} &= \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[1]}} \frac{\partial z_g^{[1]}}{\partial w_{ij}^{[1]}}, \\
&= \frac{\partial C}{\partial z_i^{[1]}} \frac{\partial z_i^{[1]}}{\partial w_{ij}^{[1]}}, \\
\frac{\partial C}{\partial w_{ij}^{[1]}} &= \frac{\partial C}{\partial z_i^{[1]}} a_j^{[0]}.
\end{aligned}
$$

Derivação em relação a $b^{[1]}$

O cálculo das derivadas parciais em relação ao vetor de vieses $b^{[1]}$ é feito de forma similar ao que foi feito para $W^{[1]}$ e para $b^{[2]}$. Pela regra da cadeia temos que:

$$
\frac{\partial C}{\partial b_i^{[1]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[1]}} \frac{\partial z_g^{[1]}}{\partial b_i^{[1]}}. \tag{3.9}
$$

Portanto, para encontrar $\frac{\partial C}{\partial b_i^{[1]}}$, precisamos, primeiramente encontrar as derivadas parciais $\frac{\partial z_g^{[1]}}{\partial b_i^{[1]}}$. Pela definição temos que $z_g^{[1]} = \sum_{k=1}^{p} w_{gk}^{[1]} a_k^{[0]} + b_g^{[1]}$. Sendo assim, temos:

$$
\begin{aligned}
\frac{\partial z_g^{[1]}}{\partial b_i^{[1]}} &= \frac{\partial}{\partial b_i^{[1]}}\left(\sum_{k=1}^{p} w_{gk}^{[1]} a_k^{[0]} + b_g^{[1]}\right), \\
&= \frac{\partial}{\partial b_i^{[1]}}\left(\sum_{k=1}^{p} w_{gk}^{[1]} a_k^{[0]}\right) + \frac{\partial b_g^{[1]}}{\partial b_i^{[1]}}, \\
&= \frac{\partial b_g^{[1]}}{\partial b_i^{[1]}} = \begin{cases} 1, & \text{se } i = g, \\ 0, & \text{caso contrário.} \end{cases}
\end{aligned}
$$

Voltando em Eq. (3.9), temos:

$$\frac{\partial C}{\partial b_i^{[1]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[1]}} \frac{\partial z_g^{[1]}}{\partial b_i^{[1]}},$$
$$= \frac{\partial C}{\partial z_i^{[1]}} \frac{\partial z_i^{[1]}}{\partial b_i^{[1]}},$$
$$\frac{\partial C}{\partial b_i^{[1]}} = \frac{\partial C}{\partial z_i^{[1]}}.$$

Com isso, finalizamos os cálculos das derivadas parciais no caso da rede neural aplicada a um elemento do conjunto D. De forma resumida, apresentamos as derivadas na Tabela 3.4.

Termo	Derivada
$A^{[2]}$	$\frac{\partial C}{\partial a_i^{[2]}} = a_i^{[2]} - y_i,\ 1 \leq i \leq n$
$Z^{[2]}$	$\frac{\partial C}{\partial z_i^{[2]}} = \frac{\partial C}{\partial a_i^{[2]}} f_2'(z_i^{[2]}),\ 1 \leq i \leq n$
$W^{[2]}$	$\frac{\partial C}{\partial w_{ij}^{[2]}} = \frac{\partial C}{\partial z_i^{[2]}} a_j^{[1]},\ 1 \leq i \leq n,\ 1 \leq j \leq p$
$b^{[2]}$	$\frac{\partial C}{\partial b_i^{[2]}} = \frac{\partial C}{\partial z_i^{[2]}},\ 1 \leq i \leq n$
$A^{[1]}$	$\frac{\partial C}{\partial a_i^{[1]}} = \sum_{g=1}^{n} \frac{\partial C}{\partial z_g^{[2]}} w_{gi}^{[2]},\ 1 \leq i \leq p$
$Z^{[1]}$	$\frac{\partial C}{\partial z_i^{[1]}} = \frac{\partial C}{\partial a_i^{[1]}} f_1'(z_i^{[1]}),\ 1 \leq i \leq p$
$W^{[1]}$	$\frac{\partial C}{\partial w_{ij}^{[1]}} = \frac{\partial C}{\partial z_i^{[1]}} a_j^{[0]},\ 1 \leq i \leq p,\ 1 \leq j \leq o$
$b^{[1]}$	$\frac{\partial C}{\partial b_i^{[1]}} = \frac{\partial C}{\partial z_i^{[1]}},\ 1 \leq i \leq p$

Tabela 3.4: Sumarização das derivadas de C em relação aos termos da estrutura da rede neural, considerando um dado de entrada.

3.2 Treinamento com o conjunto de dados completo

Durante o treinamento da rede neural estamos interessados no custo total da rede aplicada a todos os elementos do conjunto de treino D. Como na Definição 2.1, o custo total será a média aritmética dos custos individuais. Para isso, precisamos calcular a saída da rede

para cada elemento do conjunto da rede, para podermos encontrar o valor da função de perda em cada caso.

Podemos aproveitar as propriedades da multiplicação matricial para calcular a saída da rede para todos os elementos do conjunto D. Até agora tratamos a entrada como um vetor representando um elemento do conjunto de dados de entrada. Se, em vez disso, considerarmos como entrada uma matriz $X \in \mathbb{R}^{o \times m}$, onde cada coluna X_i é um elemento do conjunto, ao avançar pela rede, usando as relações definidas em (2.3) e (2.4)[1], teremos matrizes $Z^{[1]} \in \mathbb{R}^{p \times m}$, $A^{[1]} \in \mathbb{R}^{p \times m}$, $Z^{[2]} \in \mathbb{R}^{n \times m}$, e $A^{[2]} \in \mathbb{R}^{n \times m}$ em que cada coluna será a versão daquela camada para cada elemento dos dados de entrada. Consideramos também $Y \in \mathbb{R}^{n \times m}$ a matriz em que cada coluna é a saída esperada de um dos dados de entrada. Isso permite uma implementação mais eficiente, pois podemos aproveitar das otimizações implementadas nas linguagens de programação para operações matriciais, ao invés de ter que calcular a saída esperada para cada elemento de entrada separadamente em um laço de repetição.

Neste caso, para o cálculo dos custos individuais, consideramos a coluna j da matriz $A^{[2]}$ a saída correspondente ao dado de entrada $X_j = A_j^{[0]}$, a ser comparada com a saída esperada Y_j. Logo, cada custo individual será:

$$C_{D_j} = C(Y_j, F(X_j)) = C(Y_j, A_j^{[2]}) = \frac{1}{2} \left\| A_j^{[2]} - Y_j \right\|^2 = \frac{1}{2} \sum_{i=1}^{n} (a_{ij}^{[2]} - y_{ij})^2. \tag{3.10}$$

Para a atualização dos parâmetros, também podemos fazer uso desta estrutura matricial. Neste caso, as Equações (2.8) podem ser reescritas como

$$\begin{aligned} W^{[i]} &\leftarrow W^{[i]} - \alpha \frac{\partial C}{\partial W^{[i]}}, \\ b^{[i]} &\leftarrow b^{[i]} - \alpha \frac{\partial C}{\partial b^{[i]}}. \end{aligned} \tag{3.11}$$

Definimos a notação para esse caso na Tabela 3.5.

[1] Aqui estendemos a definição da soma dos vetores $b^{[i]}$ na Eq. (2.3), para significar a soma do vetor $b^{[i]}$ correspondente em cada coluna da matriz em questão.

Camada	Notação
Camada de entrada	$A^{[0]} = X = \left[a_{ij}^{[0]}\right] \in \mathbb{R}^{o\times m}$, onde $a_{ij}^{[0]}$ representa o i-ésimo elemento do j-ésimo vetor de entrada.
Camada intermediária não ativada	$Z^{[1]} = \left[z_{ij}^{[1]}\right] \in \mathbb{R}^{p\times m}$
Camada intermediária ativada	$A^{[1]} = \left[a_{ij}^{[1]}\right] \in \mathbb{R}^{p\times m}$
Camada de saída não ativada	$Z^{[2]} = \left[z_{ij}^{[2]}\right] \in \mathbb{R}^{n\times m}$
Camada de saída (ativada)	$A^{[2]} = F(X) = \left[a_{ij}^{[2]}\right] \in \mathbb{R}^{n\times m}$

Tabela 3.5: Notação matricial a ser utilizada no caso do treinamento da rede com todos os elementos do conjunto de dados.

Vejamos, então, como encontrar as derivadas parciais da função custo total. Definimos o custo total como:

$$C = \frac{1}{m}\sum_{h=1}^{m} C_{D_h}.$$

Pela linearidade da derivada, a derivada parcial do custo total C em relação a um elemento genérico θ será:

$$\frac{\partial C}{\partial \theta} = \frac{1}{m}\sum_{h=1}^{m} \frac{\partial C_{D_h}}{\partial \theta}.$$

Utilizando as derivadas parciais calculadas na seção anterior, e sumarizadas na Tabela 3.4, calculamos as derivadas para o custo total. Novamente, o processo utilizado para encontrar a derivada parcial em relação a cada elemento da estrutura é semelhante, e repetitivo. Escolhemos incluir todos os detalhes neste trabalho para garantir a completude das informações, e particularizar pontos eventualmente não triviais. Os resultados desta seção estão reunidos na Tabela 3.6.

Derivada em relação a $A^{[2]}$

Podemos definir a derivada parcial de C em relação a um elemento qualquer de $A^{[2]}$, $a_{ij}^{[2]}$, como

$$\frac{\partial C}{\partial a_{ij}^{[2]}} = \frac{1}{m}\sum_{h=1}^{m} \frac{\partial C_{D_h}}{\partial a_{ij}^{[2]}}.$$

Como definimos acima, cada C_{X_j} somente leva em consideração valores da coluna j da matriz $A^{[2]}$. Logo, $\frac{\partial C_{D_h}}{\partial a_{ij}^{[2]}} = 0$ se $h \neq j$. Sendo assim, temos:

$$\frac{\partial C}{\partial a_{ij}^{[2]}} = \frac{1}{m}\frac{\partial C_{D_j}}{\partial a_{ij}^{[2]}} = \frac{1}{m}\left(a_{ij}^{[2]} - y_{ij}\right).$$

Estendendo a ideia do vetor gradiente, definimos a derivada parcial em uma notação matricial como:

$$\frac{\partial C}{\partial A^{[2]}} = \left[\frac{\partial C}{\partial a_{ij}^{[2]}}\right] = \begin{bmatrix} \frac{\partial C}{\partial a_{11}^{[2]}} & \frac{\partial C}{\partial a_{12}^{[2]}} & \cdots & \frac{\partial C}{\partial a_{1m}^{[2]}} \\ \frac{\partial C}{\partial a_{21}^{[2]}} & \frac{\partial C}{\partial a_{22}^{[2]}} & \cdots & \frac{\partial C}{\partial a_{2m}^{[2]}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial a_{o1}^{[2]}} & \frac{\partial C}{\partial a_{o2}^{[2]}} & \cdots & \frac{\partial C}{\partial a_{om}^{[2]}} \end{bmatrix}.$$

A derivada parcial da função custo em relação à matriz $A^{[2]}$ fica:

$$\frac{\partial C}{\partial A^{[2]}} = \frac{1}{m}\left(A^{[2]} - Y\right).$$

Derivada em relação a $Z^{[2]}$

Pela linearidade da derivada, a derivada parcial de C em relação a um elemento qualquer de $z^{[2]}$, $z_{ij}^{[2]}$, vale

$$\frac{\partial C}{\partial z_{ij}^{[2]}} = \frac{1}{m}\sum_{h=1}^{m}\frac{\partial C_{D_h}}{\partial z_{ij}^{[2]}}.$$

Pela definição temos que $\frac{\partial C_{D_h}}{\partial z_{ij}^{[2]}} = 0$ quando $h \neq j$ logo:

$$\frac{\partial C}{\partial z_{ij}^{[2]}} = \frac{1}{m}\frac{\partial C_{D_j}}{\partial z_{ij}^{[2]}} = \frac{1}{m}\frac{\partial C_{D_j}}{\partial a_{ij}^{[2]}} f_2'(z_{ij}^{[2]}).$$

Para escrever em notação matricial, definimos a operação de multiplicação de matrizes elemento a elemento, $\odot$.

Definição 3.1 (Multiplicação de matrizes elemento a elemento)**.** *Sejam $A, B \in \mathbb{R}^{m\times n}$ matrizes arbitrárias. Definimos a operação matricial de multiplicação elemento a elemento, $\odot$, da seguinte forma:*

$$C = A \odot B,$$

onde cada elemento c_{ij} da matriz resultante C é definido

$$c_{ij} = a_{ij}b_{ij}, \quad i = 1, 2, ..., m,\ j = 1, 2, ..., n.$$

Logo, podemos definir a derivada parcial do custo total em relação à matriz $Z^{[2]}$, em notação matricial como

$$\frac{\partial C}{\partial Z^{[2]}} = \frac{\partial C}{\partial A^{[2]}} \odot f_2'(Z^{[2]}),$$

sendo $f_2'(Z^{[2]})$ a função $f_2' : \mathbb{R} \mapsto \mathbb{R}$, definida na Eq. (3.2), aplicada a cada elemento da matriz $Z^{[2]}$, conforme a Eq. (2.5).

Derivada em relação a $W^{[2]}$

A expressão geral para a derivada parcial de C em relação a cada elemento de $W^{[2]}$ é

$$\frac{\partial C}{\partial w_{ij}^{[2]}} = \frac{1}{m} \sum_{h=1}^{m} \frac{\partial C_{D_h}}{\partial w_{ij}^{[2]}}.$$

Neste caso, como todas as colunas da matriz de saída são influenciadas pelos elementos de $W^{[2]}$, não podemos eliminar nenhum elemento do somatório.

$$\begin{aligned}
\frac{\partial C}{\partial w_{ij}^{[2]}} &= \frac{1}{m} \sum_{h=1}^{m} \frac{\partial C_{D_h}}{\partial w_{ij}^{[2]}}, \\
&= \frac{1}{m} \sum_{h=1}^{m} \frac{\partial C_{D_h}}{\partial z_{ih}^{[2]}} a_{jh}^{[1]}, \\
&= \sum_{h=1}^{m} \frac{1}{m} \frac{\partial C_{D_h}}{\partial a_{ih}^{[2]}} f_2'(z_{ih}^{[2]}) a_{jh}^{[1]}.
\end{aligned} \tag{3.12}$$

Sabemos pelo que foi definido acima, que cada termo da matriz $\frac{\partial C}{\partial Z^{[2]}}$ tem valor:

$$\left[\frac{\partial C}{\partial Z^{[2]}}\right]_{ij} = \frac{1}{m} \frac{\partial C_{D_j}}{\partial a_{ij}^{[2]}} f_2'(z_{ij}^{[2]}).$$

Portanto, é fácil ver que a definição de $\frac{\partial C}{\partial w_{ij}^{[2]}}$ na Eq. (3.12) pode ser vista como um produto interno entre a i-ésima linha da matriz $\frac{\partial C}{\partial Z^{[2]}}$ e a j-ésima linha da matriz $A^{[1]}$. Logo, matricialmente, a derivada pode ser definida como:

$$\frac{\partial C}{\partial W^{[2]}} = \frac{\partial C}{\partial Z^{[2]}} \left(A^{[1]}\right)^T.$$

Derivada em relação a $b^{[2]}$

A derivada em relação ao termo $b^{[2]}$ é desenvolvida de forma similar ao que foi feito para a matriz $W^{[2]}$,

$$\frac{\partial C}{\partial b_i^{[2]}} = \frac{1}{m} \sum_{h=1}^{m} \frac{\partial C_{D_h}}{\partial b_i^{[2]}}.$$

Neste caso, como todas as colunas da matriz de saída são influenciadas pelos elementos de $b^{[2]}$, não podemos eliminar nenhum elemento do somatório.

$$\begin{aligned}\frac{\partial C}{\partial b_i^{[2]}} &= \frac{1}{m}\sum_{h=1}^{m}\frac{\partial C_{D_h}}{\partial z_{ih}^{[2]}},\\ &= \sum_{h=1}^{m}\frac{1}{m}\frac{\partial C}{\partial a_{ih}^{[2]}}f_2'(z_{ih}^{[2]}).\end{aligned}$$

Vemos, então que cada termo $\frac{\partial C}{\partial b_i^{[2]}}$ é a soma da i-ésima linha de $\frac{\partial C}{\partial Z^{[2]}}$. Se definirmos o vetor de uns, $e_{m\times 1}$ como

$$e_{m\times 1} = \begin{pmatrix}1\\1\\\vdots\\1\end{pmatrix} \in \mathbb{R}^{m\times 1}.$$

Podemos definir matricialmente a derivada $\frac{\partial C}{\partial b^{[2]}}$ como

$$\frac{\partial C}{\partial b^{[2]}} = \frac{\partial C}{\partial Z^{[2]}}e_{m\times 1}.$$

Derivada em relação a $A^{[1]}$

Como visto anteriormente, quando tratamos de apenas um dado de entrada, a derivada de C em relação a um elemento do vetor $A^{[1]}$ é definida como

$$\begin{aligned}\frac{\partial C}{\partial a_i^{[1]}} &= \sum_{g=1}^{n}\frac{\partial C}{\partial z_g^{[2]}}w_{gi}^{[2]},\\ &= \sum_{g=1}^{n}\frac{\partial C}{\partial a_g^{[2]}}f_2'(z_g^{[2]})w_{gi}^{[2]}.\end{aligned}$$

Cada C_h depende apenas das entradas da coluna h da matriz $A^{[1]}$. Sendo assim,

$$\begin{aligned}\frac{\partial C}{\partial a_{ij}^{[1]}} &= \frac{1}{m}\sum_{h=1}^{m}\frac{\partial C_{D_h}}{\partial a_{ij}^{[1]}},\\ &= \frac{1}{m}\frac{\partial C_{D_j}}{\partial a_{ij}^{[1]}},\\ &= \sum_{g=1}^{n}\frac{1}{m}\frac{\partial C_{D_j}}{\partial a_{gj}^{[2]}}f_2'(z_{gj}^{[2]})w_{gi}^{[2]}.\end{aligned}$$

Mais uma vez, podemos ver que cada termo $\frac{\partial C}{\partial a_{ij}^{[1]}}$ pode ser visto como um produto interno entre a j-ésima coluna de $\frac{\partial C}{\partial Z^{[2]}}$ com a i-ésima coluna de $W^{[2]}$. Então, matricialmente podemos escrever:

$$\frac{\partial C}{\partial A^{[1]}} = \left(W^{[2]}\right)^T \frac{\partial C}{\partial Z^{[2]}}.$$

Derivada em relação a $Z^{[1]}$

O desenvolvimento dessa e das próximas derivadas se assemelha ao processo feito para $Z^{[2]}$, $W^{[2]}$ e $b^{[2]}$. Isso sugere que o processo pode ser facilmente generalizado para contemplar um número diferente de camadas, desde que a estrutura básica se mantenha e que a escolha de funções de ativação sejam semelhantes (ou seja, funções de uma variável).

Para cada dado de entrada vale:

$$\begin{aligned}\frac{\partial C}{\partial z_i^{[1]}} &= \frac{\partial C}{\partial a_i^{[1]}} f_1'(z_i^{[1]}), \\ &= \sum_{g=1}^{n} \frac{\partial C}{\partial a_g^{[2]}} f_2'(z_g^{[2]}) w_{gi}^{[2]} f_1'(z_i^{[1]}),\end{aligned}$$

onde cada C_{X_h} depende apenas dos elementos da coluna h de $Z^{[1]}$. Para o custo total, então, temos:

$$\begin{aligned}\frac{\partial C}{\partial z_{ij}^{[1]}} &= \frac{1}{m} \sum_{h=1}^{m} \frac{\partial C_{D_h}}{\partial z_{ij}^{[1]}}, \\ &= \frac{1}{m} \frac{\partial C_{D_j}}{\partial z_{ij}^{[1]}}, \\ &= \frac{1}{m} \sum_{g=1}^{n} \frac{\partial C_{D_j}}{\partial a_{gj}^{[2]}} f_2'(z_{gj}^{[2]}) w_{gi}^{[2]} f_1'(z_{ij}^{[1]}).\end{aligned}$$

Ou seja, cada termo $\frac{\partial C}{\partial z_{ij}^{[1]}}$ é resultante do produto entre a entrada $\frac{\partial C}{\partial a_{ij}^{[1]}}$ com o termo $f_1'(z_{ij}^{[1]})$, o que equivale matricialmente a

$$\frac{\partial C}{\partial Z^{[1]}} = \frac{\partial C}{\partial A^{[1]}} \odot f_2'(Z^{[1]}).$$

Derivada em relação a $W^{[1]}$

Todas as colunas de saída dependem dos valores em $W^{[1]}$, logo, não é possível eliminar nenhum termo do somatório.

$$\begin{aligned}\frac{\partial C}{\partial w_{ij}^{[1]}} &= \frac{1}{m}\sum_{h=1}^{m}\frac{\partial C_{D_h}}{\partial w_{ij}^{[1]}},\\ &= \frac{1}{m}\sum_{h=1}^{m}\left(\sum_{g=1}^{n}\frac{\partial C}{\partial a_{gh}^{[2]}} f_2'(z_{gh}^{[2]}) w_{gi}^{[2]} f_1'(z_{ih}^{[1]}) a_{jh}^{[0]}\right),\\ &= \sum_{h=1}^{m}\left(\frac{1}{m}\sum_{g=1}^{n}\frac{\partial C}{\partial a_{gh}^{[2]}} f_2'(z_{gh}^{[2]}) w_{gi}^{[2]} f_1'(z_{ih}^{[1]})\right) a_{jh}^{[0]}.\end{aligned}$$

Temos que o termo $\left[\frac{\partial C}{\partial Z^{[1]}}\right]_{ij}$ vale

$$\left[\frac{\partial C}{\partial Z^{[1]}}\right]_{ij} = \frac{1}{m}\sum_{g=1}^{n}\frac{\partial C}{\partial a_{gj}^{[2]}} f_2'(z_{gj}^{[2]}) w_{gi}^{[2]} f_1'(z_{ij}^{[1]}).$$

Logo, podemos ver os termos $\frac{\partial C}{\partial w_{ij}^{[1]}}$ como um produto interno entre a i-ésima linha de $\frac{\partial C}{\partial Z^{[1]}}$ com a j-ésima linha de $A^{[0]}$. Matricialmente, escrevemos

$$\frac{\partial C}{\partial W^{[1]}} = \frac{\partial C}{\partial Z^{[1]}}\left(A^{[0]}\right)^T.$$

Derivada em relação a $b^{[1]}$

Finalmente, o último termo a ser calculado para a estrutura proposta, é $\frac{\partial C}{\partial b^{[1]}}$. Sabemos que todas as saídas são influenciadas pelos valores em $b^{[1]}$, logo não eliminamos nenhum termo no somatório.

$$\begin{aligned}\frac{\partial C}{\partial b_i^{[1]}} &= \frac{1}{m}\sum_{h=1}^{m}\frac{\partial C_{D_h}}{\partial b_i^{[1]}},\\ &= \frac{1}{m}\sum_{h=1}^{m}\sum_{g=1}^{n}\frac{\partial C}{\partial a_{gh}^{[2]}} f_2'(z_{gh}^{[2]}) w_{gi}^{[2]} f_1'(z_{ih}^{[1]}),\\ &= \sum_{h=1}^{m}\left(\frac{1}{m}\sum_{g=1}^{n}\frac{\partial C}{\partial a_{gh}^{[2]}} f_2'(z_{gh}^{[2]}) w_{gi}^{[2]} f_1'(z_{ih}^{[1]})\right),\\ &= \sum_{h=1}^{m}\left[\frac{\partial C}{\partial Z^{[1]}}\right]_{ih}.\end{aligned}$$

Vemos que cada termo $\frac{\partial C}{\partial b_i^{[1]}}$ é a soma dos valores da i-ésima linha de $\frac{\partial C}{\partial Z^{[1]}}$. Utilizando a notação introduzida anteriormente, podemos escrever:

$$\frac{\partial C}{\partial b^{[1]}} = \frac{\partial C}{\partial Z^{[1]}} e_{m\times 1}.$$

Juntando as expressões encontradas nesta seção, podemos sumarizar as expressões a serem calculadas na Tabela 3.6.

Termo	Derivada
$A^{[2]}$	$\dfrac{\partial C}{\partial A^{[2]}} = \dfrac{1}{m}\left(A^{[2]} - Y\right)$
$Z^{[2]}$	$\dfrac{\partial C}{\partial Z^{[2]}} = \dfrac{\partial C}{\partial A^{[2]}} \odot f_2'(Z^{[2]})$
$W^{[2]}$	$\dfrac{\partial C}{\partial W^{[2]}} = \dfrac{\partial C}{\partial Z^{[2]}}\left(A^{[1]}\right)^T$
$b^{[2]}$	$\dfrac{\partial C}{\partial b^{[2]}} = \dfrac{\partial C}{\partial Z^{[2]}} e_{m\times 1}$
$A^{[1]}$	$\dfrac{\partial C}{\partial A^{[1]}} = \left(W^{[2]}\right)^T \dfrac{\partial C}{\partial Z^{[2]}}$
$Z^{[1]}$	$\dfrac{\partial C}{\partial Z^{[1]}} = \dfrac{\partial C}{\partial A^{[1]}} \odot f_2'(Z^{[1]})$
$W^{[1]}$	$\dfrac{\partial C}{\partial W^{[1]}} = \dfrac{\partial C}{\partial Z^{[1]}}\left(A^{[0]}\right)^T$
$b^{[1]}$	$\dfrac{\partial C}{\partial b^{[1]}} = \dfrac{\partial C}{\partial Z^{[1]}} e_{m\times 1}$

Tabela 3.6: Sumarização das derivadas de C em relação aos termos da estrutura da rede em formato matricial.

3.3 Inicialização dos parâmetros

A inicialização dos parâmetros foi feita conforme descrito em Glorot e Bengio (2010). Descrevemos a seguir como é feita a inicialização dos parâmetros $W^{[i]}$ e $b^{[i]}$. Os vetores $b^{[i]}$ são inicializados com todos os elementos iguais a zero. Portanto, temos:

$$b^{[1]} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^p, \quad b^{[2]} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^n. \tag{3.13}$$

Para as matrizes $W^{[i]}$, cada elemento da matriz é escolhido aleatoriamente a partir de uma distribuição de probabilidade uniforme contínua parametrizada de acordo com o número de colunas na matriz, conforme descrito a seguir:

$$W^{[1]} \in \mathbb{R}^{p\times o},\ \text{com } w_{ij}^{[1]} \sim U_c\left[\frac{-1}{\sqrt{o}}, \frac{1}{\sqrt{o}}\right], \quad W^{[2]} \in \mathbb{R}^{n\times p},\ \text{com } w_{ij}^{[2]} \sim U_c\left[\frac{-1}{\sqrt{p}}, \frac{1}{\sqrt{p}}\right]. \tag{3.14}$$

3.4 Algoritmo

Tendo as expressões definidas acima, podemos definir um algoritmo a ser implementado, para fazer o treinamento da rede, isto é, aproximar os parâmetros $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$ que minimizam a função C, definido no Algoritmo 1.

Algoritmo 1: Algoritmo de treinamento de uma rede neural com uma camada intermediária

Entradas:

$X = A^{[0]} \in \mathbb{R}^{o \times m}$, matriz com dados de entrada;

$Y \in \mathbb{R}^{n \times m}$, matriz com saídas esperadas;

$p \in \mathbb{N}$, número de neurônios na camada intermediária;

$\alpha > 0 \in \mathbb{R}$, tamanho do passo;

$i \in \mathbb{N}$, número de iterações;

Passo 1: Inicializar parâmetros

Definir os valores iniciais para $W^{[1]} \in \mathbb{R}^{p \times o}, b^{[1]} \in \mathbb{R}^{p}, W^{[2]} \in \mathbb{R}^{n \times p}, b^{[2]} \in \mathbb{R}^{n}$, como definido nas Equações (3.13) e (3.14);

Passo 2: *Forward pass*

Calcular $Z^{[1]}, A^{[1]}, Z^{[2]}, A^{[2]}$ como definido nas Equações (2.3) e (2.4);

Passo 3: *Backpropagation*

Calcular $\frac{\partial C}{\partial W^{[2]}}, \frac{\partial C}{\partial b^{[2]}}, \frac{\partial C}{\partial W^{[1]}}, \frac{\partial C}{\partial b^{[1]}}$ conforme expressões da Tabela 3.6;

Passo 4: Atualização dos parâmetros

Atualizar cada parâmetro $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$, conforme Eq. (3.11);

Repetir i vezes os passos 2 a 4.

Saída:

$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$, aproximações encontradas.

Capítulo 4

Experimentos numéricos

Para avaliar o método proposto e exposto nas seções anteriores deste trabalho, apresentamos neste capítulo os resultados de experimentos numéricos computacionais realizados. Primeiramente, introduzimos o problema que será utilizado como base e os dados a serem utilizados. Em seguida, descrevemos o experimento e os resultados observados.

4.1 Problema-base: reconhecimento de números manuscritos

O problema a ser utilizado como prova de conceito neste trabalho será o de classificar uma imagem de acordo com o algarismo manuscrito representado na imagem. Como dados de entrada, utilizamos a base modificada do Instituto Nacional de Padrões e Tecnologia dos Estados Unidos[1], conhecida pela sigla *MNIST*(LECUN; CORTES, 2010). Essa base contém imagens rotuladas de dígitos manuscritos por trabalhadores do Departamento do Censo dos Estados Unidos e estudantes de ensino médio. Alguns exemplos das figuras da base de dados estão na Figura 4.1.

A base utilizada contém 70000 imagens de números escritos a mão. As imagens têm resolução de 28×28px, e estão em escala de cinza, para que cada pixel da imagem tenha apenas um valor correspondente: 0 para branco, 255 para preto, e valores inteiros no intervalo $(0, 255)$ para cinzas intermediários. As imagens estão rotuladas, ou seja, para cada imagem, sabemos qual dígito ela representa. Como estamos classificando algarismos, as possíveis classes estão no conjunto $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Diversas são as formas de aplicar os conceitos de redes neurais para problemas de classificação tomando imagens como dados de entrada. A abordagem de linearização a ser apresentada aqui, utilizando imagens em escala de cinza, é apenas uma delas, e é uma das mais simples que possibilita aproveitar a estrutura de redes neurais apresentada até aqui. Portanto, para que possamos trabalhar com a estrutura proposta neste trabalho, será feita uma linearização das imagens. Considerando a imagem como uma matriz em $\mathbb{R}^{28\times28}$, o vetor coluna representando a imagem i será formado pelos elementos da primeira linha, ordenados da esquerda para direita, seguidos dos elementos da segunda linha, e assim por diante até incluirmos os elementos da última linha da matriz da imagem, conforme diagrama da Figura 4.2.

[1] *Modified National Institute of Standards and Technology database*

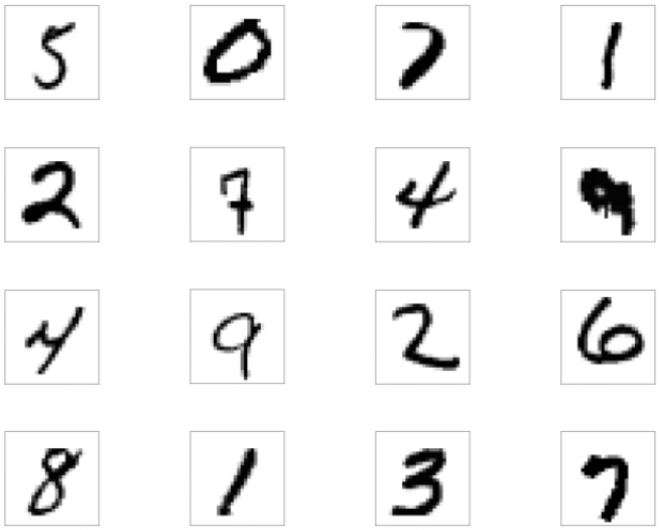

Figura 4.1: Exemplos de imagens da base de dados MNIST. Fonte: Imagens renderizadas a partir da base de dados *MNIST* usando funções gráficas da linguagem *Python*.

Cada elemento de entrada da rede será, então, um vetor no espaço $\mathbb{R}^{784}$. Sendo assim, a matriz de dados de entrada X que será fornecida para o modelo como dados de treinamento será uma matriz em $\mathbb{R}^{784\times m}$, sendo m a quantidade de elementos no conjunto de dados de treinamento, onde cada coluna será o resultado da linearização de uma imagem.

Como foi mencionado acima, cada imagem está rotulada para indicar a resposta esperada. Para essa implementação, consideramos que cada saída esperada é um vetor da base canônica do espaço $\mathbb{R}^{10}$ em que cada posição está atrelada a uma classe, conforme a Definição 4.1.

Definição 4.1 (Vetor de classificação)**.** *Identificamos a classe de uma imagem com um vetor de classificação, um vetor canônico de* $\mathbb{R}^{10}$*. Cada posição representa uma das 10 classes possíveis. Assim, o elemento 1 estará na posição que corresponde à classe da imagem. Segue um exemplo de um vetor representando a classe "4":*

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \leftarrow \textit{classe 0} \\ \leftarrow \textit{classe 1} \\ \leftarrow \textit{classe 2} \\ \leftarrow \textit{classe 3} \\ \leftarrow \textit{classe 4} \\ \leftarrow \textit{classe 5} \\ \leftarrow \textit{classe 6} \\ \leftarrow \textit{classe 7} \\ \leftarrow \textit{classe 8} \\ \leftarrow \textit{classe 9.} \end{matrix}$$

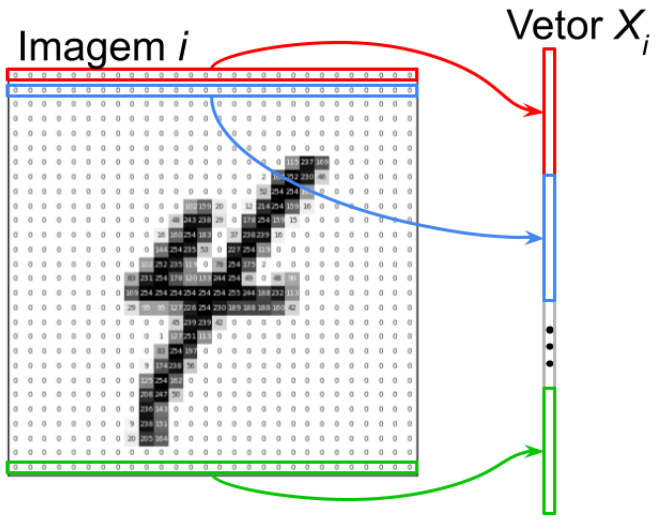

Figura 4.2: Diagrama mostrando como é feita a linearização da matriz da imagem. Fonte: Autoria própria.

4.2 Experimento computacional

Para o experimento computacional, a base foi obtida com o uso da biblioteca *Scikit-Learn*, da linguagem de programação *Python*[2]. O código utilizado para os experimentos está disponível para consulta no repositório do *GitHub*[3].

Os dados de entrada são disponibilizados no formato linearizado, e as classes correspondentes são informadas por algarismos. O pré-processamento aplicado nas bases incluiu as seguintes atividades:

- Nos dados de entrada: normalização dos valores para o intervalo $[0, 1]$, dividindo o valor de cada pixel por 255.
- Nas saídas esperadas: transformação de cada saída esperada que vem como algarismo para o vetor de classificação correspondente.

Após o pré-processamento, separamos as bases de treino e de teste. A base disponibilizada está embaralhada de forma que a distribuição de classes seja aproximadamente uniforme, e separada em 60000 elementos de treino e os restantes 10000 para teste. O algoritmo será aplicado na base de treino, com o objetivo de aproximar os parâmetros da rede, e a base de testes será mantida inédita para propósitos de avaliação da capacidade de generalização do modelo. Ao fim das iterações do Algoritmo 1, temos aproximações dos parâmetros $W^{[i]}$ e $b^{[i]}$ ótimos. Podemos utilizar esses parâmetros para calcular as saídas fornecidas pela rede para dados que não estavam contidos no conjunto de treinamento. Para essa avaliação utilizamos a acurácia conforme a Definição 1.3.

Um acerto é definido como o caso em que a saída da rede classificou corretamente a imagem. Como estamos utilizando neurônios com funções de ativação e não *perceptrons*,

[2]Foi utilizada a função `sklearn.datasets.fetch_openml('mnist_784')`.
[3]https://github.com/dudinhadnm/ProjetoMatematicaIndustrial

a saída da rede não necessariamente será um vetor da base canônica de $\mathbb{R}^{10}$, como é a saída esperada. Para contornar isso, definimos uma função auxiliar, $R(v)$ para mapear a saída da rede para um vetor da base canônica, ou seja, um vetor em que todas as entradas são 0, com exceção de uma entrada que tem valor 1, sinalizando a classe escolhida.

Definição 4.2 (Função de classificação)**.** *Seja v um vetor resultante da rede neural. Definimos a função auxiliar de classificação, $R : \mathbb{R}^{10} \mapsto \mathbb{R}^{10}$, como a função que mapeia a saída da rede para um vetor da base canônica definida elemento a elemento da seguinte forma:*

$$[R(v)]_i = \begin{cases} 1, & \text{se } v_i = max(v), \\ 0, & \text{caso contrário.} \end{cases} \tag{4.1}$$

No caso de um empate, a função seleciona o menor i, tal que $v_i = max(v)$, para receber o valor 1.

Definição 4.3 (Acerto)**.** *Seja $F(X_i)$ a saída da rede neural para o dado de entrada X_i, Y_i a saída esperada correspondente, e $R(\cdot)$ a função auxiliar da Definição 4.2. Definimos um acerto de classificação da rede casos em que*

$$R(F(X_i)) = Y_i. \tag{4.2}$$

4.3 Resultados

Após a separação das bases de treino e teste, a distribuição das classes foi verificada, conforme Tabela 4.1. Vemos que as classes se encontram aproximadamente balanceadas, com cada classe representando algo próximo de 10% de cada base. Isso é importante para que a rede tenha exemplos de cada classe de forma igualitária, para que não sofra nenhum viés inesperado.

Classe	0	1	2	3	4	5	6	7	8	9
Treino	0.099	0.112	0.099	0.102	0.097	0.09	0.099	0.104	0.098	0.099
Teste	0.098	0.114	0.103	0.101	0.098	0.089	0.096	0.103	0.097	0.101

Tabela 4.1: Distribuição das classes nos conjuntos de dados.

Para encontrar os hiperparâmetros a serem utilizados, executamos um processo de *grid search* considerando os seguintes valores possíveis para os hiperparâmetros:

- p (neurônios da camada intermediária): $\{10, 20, 30, 40, 50, 70, 80, 90, 100, 120, 130, 150\}$;
- α (tamanho do passo): $\{0.1, 0.5, 1, 2, 3, 4\}$.

A combinação de hiperparâmetros que trouxe uma melhor performance no contexto de validação foi $p = 80$, $\alpha = 3$. Esses valores foram usados no treinamento final da base, usando a base de treino completa no Algoritmo 1. Ao fim das 600 iterações obtivemos aproximações das matrizes e vetores, $W^{[i]}$ e $b^{[i]}$, que foram utilizados no contexto de testes, usando a base de testes, considerada inédita para o algoritmo.

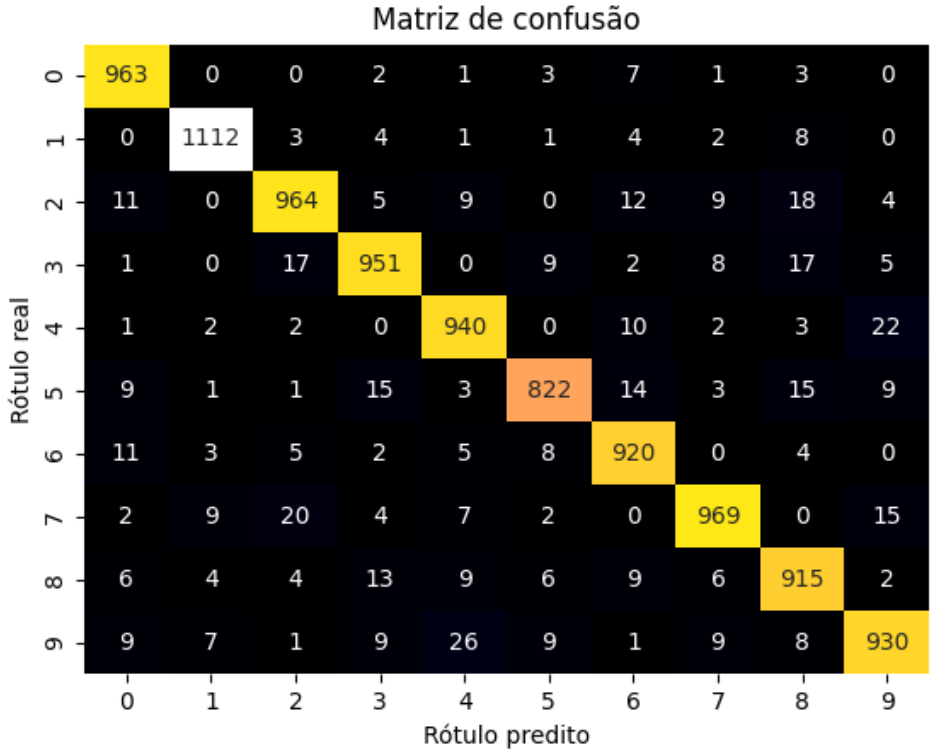

Figura 4.3: Matriz de confusão do modelo aplicado à base inédita.

A rede, quando aplicada aos dados de treino, obteve uma acurácia de $a = 0.9511$. Por sua vez, quando aplicamos a rede aos dados de teste, obtivemos uma acurácia de $a = 0.9486$. Assim, vemos que a performance da rede em ambas as bases foi próxima, e ambas tiveram um desempenho consideravelmente bom. O problema de classificação de algarismos manuscritos é um problema complexo, e mesmo com uma rede neural com apenas uma camada intermediária conseguimos uma boa quantidade de acertos, sem precisar de um pré-processamento extenso dos dados.

Na Figura 4.3 vemos a matriz de confusão resultante da base de testes. Ela mostra a relação entre a saída esperada de cada elemento ("Rótulo real" no eixo vertical) e a classificação feita pela rede neural ("Rótulo predito" no eixo horizontal). Os valores da matriz representam quantas imagens com o rótulo real da linha foram classificadas com o rótulo da coluna pelo algoritmo. Por exemplo, na Figura 4.3 o valor 9 na última linha e primeira coluna da matriz indica que 9 imagens que têm o rótulo real "9" foram classificadas com o rótulo "0" pela rede. Para um classificador perfeito as entradas fora da diagonal principal serão todas nulas. No caso da Figura 4.3 vemos que muitos dos elementos fora da diagonal principal são nulos, e a maioria dos resultados está concentrado na diagonal principal, indicando muitas classificações corretas, o que corrobora a medida de acurácia apresentada anteriormente. Além disso, não é possível observar nenhuma concentração clara de classificações incorretas, sugerindo que não houve um ponto de confusão específico. Os erros da rede estão bem distribuídos, e nenhuma classe em especial necessita de atenção especial.

Conclusões

No estudo desenvolvido para essa monografia, foi apresentada a modelagem matemática de uma rede neural *feedforward*, viabilizando o detalhamento da estrutura dessa rede neural artificial, o desenvolveimento dos cálculos necessários para a atualização dos parâmetros e a implementação de um algoritmo de treinamento da rede. Com isso, foi possível atingir o principal objetivo que motivou este estudo, ou seja, o detalhamento da modelagem matemática básica que está por trás do treinamento desse modelo de aprendizagem de máquina.

Utilizando uma estrutura com apenas uma camada intermediária, foram realizados os cálculos das derivadas parciais a fim de aplicar o método *backpropagation*. Primeiramente, foram analisadas as expressões elemento a elemento para apenas um dado de entrada, e, posteriormente, foram agregados os resultados para permitir cálculos utilizando matrizes de entrada, de forma a otimizar a implementação computacional. Após o cálculo dessas expressões, foi apresentado um algoritmo para o treinamento da rede. O algoritmo em questão foi implementado na linguagem *Python*, e sua performance foi avaliada utilizando o conjunto de dados *MNIST* para resolver um problema de reconhecimento de imagem, fazendo a classificação de dígitos manuscritos. Com os experimentos foi possível concluir que a estrutura, mesmo que simples, com apenas uma camada intermediária, é capaz classificar as imagens do problema, com uma boa capacidade de generalização, apresentando resultados satisfatórios em bases de dados inéditas, com as quais foi possível obter uma acurácia de 94,86%.

Para oportunidades futuras, com o objetivo de expandir o trabalho desenvolvido até o momento, reserva-se a intenção de estender a modelagem proposta aqui para uma quantidade arbitrária de camadas intermediárias, possibilitando mais liberdade à estrutura da rede. Além disso, outros passos para aprimorar este trabalho incluem a utilização de outras funções de ativação, como a *ReLu* e a *softmax*, de outras funções custo, como a *log-loss*, e o estudo de métodos de aceleração, em especial o método proposto por Nesterov (1983).

Referências Bibliográficas

CHOLLET, F. *Deep Learning with Python*. [S.l.]: Manning, 2017. ISBN 978-16-172-9443-3.

GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017. ISBN 978-14-919-6229-9.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.). *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 249–256. Disponível em: ⟨https://proceedings.mlr.press/v9/glorot10a.html⟩.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. ⟨http://www.deeplearningbook.org⟩.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Y.; CORTES, C. MNIST handwritten digit database. 2010. Disponível em: ⟨http://yann.lecun.com/exdb/mnist/⟩.

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

MITCHELL, T. M. *Machine Learning*. [S.l.]: McGraw-Hill Education, 1997. ISBN: 978-0070428072.

NESTEROV, Y. A method for solving the convex programming problem with convergence rate o(1/kˆ2). *Proceedings of the USSR Academy of Sciences*, v. 269, p. 543–547, 1983.

NESTEROV, Y. *Introductory Lectures on Convex Programming. Volume I: Basic Course*. 1998. Disponível em: ⟨https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.693.855&rep=rep1&type=pdf⟩.

NIELSEN, M. A. misc, *Neural Networks and Deep Learning*. Determination Press, 2015. Disponível em: ⟨http://neuralnetworksanddeeplearning.com/⟩.

RIBEIRO, A. A.; KARAS, E. W. *Otimização contínua: Aspectos teóricos e computacionais*. [S.l.]: Cengage Learning, 2013. ISBN 978-85-221-1501-3.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386–408, 1958. ISSN 0033-295X. Disponível em: ⟨http://dx.doi.org/10.1037/h0042519⟩.

RUMELHART, D. E. et al. Backpropagation: The basic theory. In: _____. Hillsdale, NJ, US: Lawrence Erlbaum Associates, Inc, 1995. (Developments in connectionist theory.), p. 1–34. ISBN 0-8058-1258-X (Hardcover); 0-8058-1259-8 (Paperback).

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, v. 323, n. 6088, p. 533–536, Oct 1986. ISSN 1476-4687. Disponível em: ⟨https://doi.org/10.1038/323533a0⟩.

STEWART, J. *Cálculo*. São Paulo - SP: Cengage Learning, 2013. v. 2. Tradução da 7a edição norte-americana.

SWAMYNATHAN, M. *Mastering Machine Learning with Python in Six Steps. A Practical Implementation Guide to Predictive Data Analytics Using Python*. [S.l.]: Apress, 2017. ISBN 978-1-4842-2866-1.

TIBSHIRANI, R. *Gradient Descent: Convergence Analysis*. 2013. Disponível em: ⟨https://www.stat.cmu.edu/~ryantibs/convexopt-F13/scribes/lec6.pdf⟩.