UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA

Vitor Monteiro Andrade Goulart

Treinamento de redes neurais com incorporação da técnica Backpropagation ao FDIPA

Juiz de Fora
2021

Vitor Monteiro Andrade Goulart

Treinamento de redes neurais com incorporação da técnica Backpropagation ao FDIPA

Dissertação apresentada ao Programa de Pós-graduação em Matemática da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Matemática. Área de concentração: Matemática Aplicada.

Orientador: Prof. Dr. Wilhelm Passarella Freire

Coorientador: Prof. Dr. Sandro Rodrigues Mazorche

Juiz de Fora

2021

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Monteiro Andrade Goulart, Vitor.
Treinamento de redes neurais com incorporação da técnica Backpropagation ao FDIPA / Vitor Monteiro Andrade Goulart. – 2021.
79 f. : il.

Orientador: Wilhelm Passarella Freire
Coorientador: Sandro Rodrigues Mazorche
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-graduação em Matemática, 2021.

1. Redes Neurais. 2. Aprendizado de Máquina. 3. *Backpropagation*. I. Freire, Wilhelm, orient. II. Mazorche, Sandro, coorient. III. Título.

Vítor Monteiro Andrade Goulart

Treinamento de Redes Neurais com Incorporação da Técnica Backpropagation ao FDIPA.

Dissertação apresentada ao Programa de Pós-graduação em Matemática da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Matemática. Área de concentração: Matemática Aplicada

Aprovada em 11 de fevereiro de 2022.

BANCA EXAMINADORA

Prof. Dr. Wilhelm Passarella Freire - Orientador

Universidade Federal de Juiz de Fora

Prof. Dr. Sandro Rodrigues Mazorche - Coorientador

Universidade Federal de Juiz de Fora

Prof. Dr. Jose Herskovits Norman

Instituto Militar de Engenharia

Prof. Dr. Leonardo Goliatt da Fonseca

Universidade Federal de Juiz de Fora

Juiz de Fora, 22/02/2022.

Documento assinado eletronicamente por **Wilhelm Passarella Freire**, **Professor(a)**, em 23/02/2022, às 11:28, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº

10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Leonardo Goliatt da Fonseca**, **Professor(a)**, em 23/02/2022, às 17:34, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Sandro Rodrigues Mazorche**, **Professor(a)**, em 23/02/2022, às 17:46, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Jose Herskovits Norman**, **Usuário Externo**, em 11/04/2022, às 14:49, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **0689714** e o código CRC **47EE40CC**.

AGRADECIMENTOS

A Deus em primeiro lugar, por ter me dado saúde e força para superar as dificuldades.

Aos meus pais, Adriene e Marco Antônio, e irmão, Arthur, pelo amor, incentivo e apoio incondicional.

À minha namorada Karine, que sempre esteve ao meu lado durante o meu percurso acadêmico. Agradeço pelo carinho, apoio, incentivo e amor.

Aos meus familiares e amigos que me apoiaram durante esse processo.

Aos meus amigos e colegas feitos dentro da Universidade Federal de Juiz de Fora (UFJF), com os quais pude ter uma conversa, seja sobre a vida, seja para discutir dúvidas quando precisasse. Destaco três nomes que eu sempre pude contar desde a minha graduação, Gabriel Brandão de Miranda, Jhuan Barbosa da Silva e Cedro e Raphael Cascelli dos Santos Souza. Todos muito talentosos e prestativos em ajudar seja em problemas técnicos, seja com problemas do dia a dia. Agradeço pelo privilégio de ser amigo desses.

Ao meu orientador Wilhelm Passarella Freire e ao meu coorientador Sandro Rodrigues Mazorche, pelo seu suporte, pelas suas correções e incentivos na elaboração deste trabalho. Agradeço também ao Luis Henrique Simplício Ribeiro pelas diversas ajudas para entender alguns dos assuntos desse trabalho.

Agradeço também aos membros da banca de avaliação final por aceitarem debater minha pesquisa.

Por fim, agradeço à CAPES e CNPq pelo apoio financeiro

RESUMO

As Redes Neurais Artificiais são modelos matemáticos e computacionais inspirados no funcionamento do cérebro humano. Elas são capazes de aprender e realizar tarefas como reconhecimento de padrões, classificação de imagens, detecção de fraudes em cartão de crédito entre outras aplicações. A estrutura de uma rede é composta de nós (que são os neurônios) ligados por arestas (que são as conexões) distribuídos em camadas. Essas conexões possuem valores (pesos) que representam o quanto aquela ligação é importante para a determinação do resultado final. A computação da rede é dada por uma série de composição de funções (funções de ativação) aplicadas ao produto dos pesos pelos valores atribuídos aos neurônios de cada camada. Para que a rede possa aprender, técnicas de otimização devem ser aplicadas para a determinação dos pesos ótimos da rede. Esse trabalho teve como principal objetivo incorporar a técnica *backpropagation* ao algoritmo de otimização FDIPA - *Feasible Directions Interior Point Algorithm* para a obtenção dos pesos ótimos de uma rede neural. Concluída essa tarefa, vários testes foram realizados para a comprovação da eficiência da proposta.

Palavras-chave: Redes Neurais, Otimização, Aprendizado de Máquina, *Backpropagation*.

ABSTRACT

Artificial Neural Networks are mathematical and computational models inspired by the functioning of the human brain. They are able to learn and perform tasks such as pattern recognition, image classification, credit card fraud detection and other applications. The structure of a network is composed of nodes (which are the neurons) connected by edges (which are the conections) distributed in layers. These conections have values (weights) that represent how important the connection is to the determination of the final result. The computation of the network is given by a series of composition of functions (activation functions) applied to the product of the weights by the values attributed to the neurons in each layer. In order to the network to learn, optimization techniques must be applied to determine the optimal network weights. The main objective of this work is to incorporate the *backpropagation* technique to the optimization algorithm FDIPA - *Feasible Directions Interior Point Algorithm* to obtain the optimal weights of a neural network. After completing this task, several tests were carried out to prove the efficiency of the proposal.

Keywords: Neural Networks, Optimization, Machine Learning, Backpropagation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Popularmente conhecido pelo seu termo em inglês *Machine Learning*, o Aprendizado de Máquina é um subcampo da Inteligência Artificial que estuda algoritmos computacionais que são capazes de aprender a realizar tarefas automaticamente através da experiência [1]. Dessa forma, ao invés de um programador desenvolver um algoritmo e deixar as ações que esse algoritmo irá realizar explicitamente no seu código, os algoritmos que usam aprendizado de máquina irão aprender a realizar aquela tarefa através da experiência, onde experiência nesse contexto é apresentada como um conjunto de dados conhecidos que dizem o que o algoritmo deveria fazer em um conjunto finito de situações conhecidas.

Uma subárea do aprendizado de máquina que vem chamando a atenção de boa parte dos pesquisadores é o Aprendizado Profundo. Ele é comumente utilizado quando o problema em questão envolve abstrações de alto nível de dados e para isso, uma arquitetura específica é utilizada que são as chamadas Redes Neurais Artificiais Profundas. Essas Redes Neurais existem desde os anos 50, porém, só recentemente obtiveram seu sucesso uma vez que houve o avanço do poder computacional e as capacidades de armazenamento de dados até o ponto em que o aprendizado profundo pudesse se tornar realidade na prática [2].

As Redes Neurais Artificiais (RNA), são modelos computacionais formados por neurônios e pesos podendo ser representados graficamente por um diagrama ou um modelo de grafo, onde os pontos ou nós são os neurônios e as arestas representam as ligações entre os neurônios da rede. Associada a essas ligações estão os pesos da rede, que são valores reais que representam a intensidade com que a informação será passada de um neurônio para o outro através dessa ligação. A rede neural pode ser vista como uma função matemática que possui seus parâmetros (os pesos da rede) e que, dada uma entrada, irá retornar uma saída. Entre os problemas que envolvem o aprendizado de máquina utilizando tais estruturas de redes neurais, uma delas, o treinamento de redes neurais, consiste em encontrar os pesos que minimizem uma função de erro que é relacionada com as respostas que a rede nos fornece. Por exemplo, se tivermos uma rede representada pela função $f : \mathbb{R}^k \times \mathbb{R}^n \longrightarrow \mathbb{R}^m$, onde $\mathbb{R}^k$ representa o domínio dos pesos e $\mathbb{R}^n$ representa o domínio da entrada da rede, o problema consiste em encontrar um conjunto de pesos w, que minimize uma função de erro $C(f(w, x))$, para um conjunto de dados x conhecido.

Para a minimização da função de erro, utiliza-se técnicas de Otimização que se trata de uma área da Matemática que busca encontrar os pontos do domínio de uma certa função, denominada função objetivo, que produzem os maiores ou menores valores funcionais, conforme o problema. Quando a função que se deseja otimizar é diferenciável, técnicas como o algoritmo de Descida do Gradiente [3], que utilizam as derivadas da função, podem ser utilizadas para resolver o problema. No caso de redes neurais, a função

custo da rede envolve uma série de composição de funções e multiplicação de matrizes que dificultam enormemente a obtenção das derivadas.

Em 1986, foi introduzido um algoritmo chamado de *Backpropagation* [4], que é utilizado para o cálculo do gradiente de uma função custo de uma rede neural de uma forma eficiente. O *backpropagation* é sem dúvida um dos algoritmos mais importantes na história das redes neurais e com o emprego desse algoritmo é possível realizar o treinamento de uma rede neural utilizando-se, por exemplo, a descida de gradiente.

O objetivo principal desse trabalho é adaptar o algoritmo FDIPA, sigla em inglês para algoritmo de pontos interiores e direções viáveis, para o treinamento de redes neurais, incorporando ao mesmo a técnica *backpropagation* a fim de obtermos um método eficiente e robusto para a minimização da função custo. O algoritmo FDIPA (Feasible Directions Interior Point Algorithm), desenvolvido por Herskovits [5] para solução de problemas de otimização não linear diferenciáveis com restrições. O FDIPA é um algoritmo robusto, eficiente e de fácil implementação. Ele basicamente requer a resolução de dois sistemas lineares com a mesma matriz. O FDIPA apresentou bons resultados para resolução de diversos problemas, o que nos motivou a aplicá-lo ao problema de treinamento de redes neurais. Outro aspecto é que o FDIPA requer somente o cálculo das derivadas parciais de primeira ordem da função objetivo. Escolhemos a técnica *backpropagation* amplamente utilizada para o cálculo das derivadas da função custo associada a redes neurais e fizemos a junção dos dois algoritmos, realizando as adaptações necessárias, com o objetivo de otimização dos pesos associados a uma rede arbitrária.

1.1 DESCRIÇÃO DO PROBLEMA

Como já mencionado anteriormente, o problema de treinamento de uma rede neural consiste em minimizar uma função custo. Para a definição da função custo, é necessário estabelecer a função que representa a computação da rede que depende da arquitetura da rede, isto é, depende da quantidade de camadas, da quantidade de neurônios por camadas, e das funções de ativação utilizadas. Além disso, a função custo depende de um conjunto de dados contendo entradas e, no caso do aprendizado supervisionado, saídas esperadas para cada uma dessas entradas.

O problema de treinamento de redes neurais agora consiste em minimizar a função custo com o objetivo de encontrar o melhor conjunto de pesos que faz com que as saídas obtidas sejam mais próximas possíveis das saídas conhecidas.

Os bons resultados numéricos apresentados pelo FDIPA em outros contextos, nos motivaram a utilizar este algoritmo para o treinamento de redes neurais. Entretanto o FDIPA resolve problemas com restrições o que não é o caso de redes neurais pois os pesos podem assumir quaisquer valores reais. Para contornarmos essa inconveniência, impusemos uma restrição ao tamanho do vetor de pesos.

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está estruturado em 8 capítulos sendo este o primeiro. No capítulo 2 faremos uma revisão bibliográfica de algumas técnicas utilizadas para o treinamento de redes neurais. No capítulo 3, recordamos a definição de neurônio artificial, descrevemos como ocorre a representação de redes neurais, explicamos como se define a função custo e como acontece o treinamento de uma rede neural. Além disso, descrevemos em detalhes o método de Descida de Gradiente Estocástico. Ainda neste capítulo, alguns teoremas e definições básicas importantes para o entendimento do assunto em tela são apresentados. No capítulo 4, os cálculos das derivadas feitos pelo *Backpropagation* são apresentados em detalhe. Pretende-se que esta seja outra contribuição deste trabalho pois trata-se de um tema de difícil compreensão. Ainda com o intuito de deixar bem claro como esses cálculos são feitos, foi apresentado um exemplo numérico que utiliza uma rede neural simples. No capítulo 5 apresentamos o algoritmo FDIPA em sua versão original, detalhamos como sua direção de busca é encontrada e descrevemos as expressões que fornecem explicitamente essa direção para o caso especial abordado nesse trabalho, ou seja, o caso em que precisamos de uma restrição somente. No capítulo 6, será apresentada a metodologia utilizada para a realização deste trabalho, ou seja, as fases de implementação dos algoritmos e etapas de testes. No capítulo 7, apresentaremos os resultados obtidos pelos algoritmos propostos neste trabalho e, finalmente, as conclusões são apresentadas no capítulo 8.

2 REVISÃO BIBLIOGRÁFICA

Atualmente, o aprendizado profundo vem sendo utilizado em diversas aplicações de diversas áreas, como carros autônomos [6], na área de cuidados da saúde [7], na área de finanças [8], dentre muitas outras. Assim, muitos pesquisadores vêm estudando formas de melhorar o treinamento das redes neurais, usadas no aprendizado profundo. Áreas como dos carros autônomos e cuidados da saúde por exemplo, envolvem vidas humanas e o erro cometido nas decisões deve ser muito próximo de zero para que essas tecnologias sejam viáveis e que as pessoas tenham confiança suficiente para utiliza-las.

Em [9] é mostrado que, em certas aplicações, o treinamento de uma rede neural utilizando um algoritmo genético apresenta melhores resultados do que com o emprego de técnicas tradicionais como o *backpropagation* aliado à Descida de Gradiente. Os autores destacam como ponto negativo das técnicas que usam o gradiente, a necessidade do cálculo do gradiente, cálculo esse que pode ser muito custoso no caso das redes profundas. Em [10], o autor chama atenção para o uso de algoritmos genéticos para treinamento de redes neurais aplicadas a problemas de classificação.

Em [11] é realizado o treinamento de redes neurais utilizando a técnica de otimização por enxame de partículas, um método utilizado para otimização de funções contínuas não lineares, inspirado em comportamento social de organismos vivos.

Em [12] é apresentada uma visão geral sobre os métodos de descida de gradiente bem como suas variações. O treinamento clássico é feito com a técnica de descida de gradiente, que consiste em minimizar uma função objetivo $C(W)$ parametrizada pelos pesos $W \in \mathbb{R}^n$ do modelo através da atualização desses pesos na direção oposta à direção do gradiente $\nabla_W C(W)$ da função objetivo. Uma taxa de aprendizado η determina o tamanho do passo que será dado na direção oposta do gradiente. A atualização dos pesos é realizada da seguinte forma:

$$W^{k+1} = W^k - \eta \nabla_W C(W^k). \tag{2.1}$$

Existem algumas variações da descida de gradiente em relação à quantidade de dados utilizados na definição da função custo e consequentemente no cálculo do gradiente. Dependendo da quantidade de amostras, ocorre um *trade-off* entre a acurácia da atualização dos pesos e o tempo computacional gasto nessa atualização.

Quando utiliza-se todo o conjunto de dados de uma só vez, dependendo de seu tamanho, o algoritmo pode ficar muito lento ou até mesmo intratável. Quando a Descida do Gradiente considera todo o conjunto de dados o algoritmo é denominado **Descida de Gradiente em Lote** (Batch Gradient Descent - BGD). O BGD converge localmente quando as funções custo são não-convexas.

O algoritmo de Descida de Gradiente Estocástica ou SGD (sigla em inglês para

Stochastic Gradient Descent), realiza a descida em cada amostra de treinamento $x^{(i)}$ com rótulo $y^{(i)}$ da seguinte forma

$$W^{k+1} = W^k - \eta \nabla_W C(W^k, x^{(i)}, y^{(i)}). \tag{2.2}$$

O SGD treina a rede mais rapidamente que o BGD com a diferença de realizar a atualização individualmente para cada amostra no conjunto de dados. Com isso, o SGD produz mais atualizações, o que causa uma variância mais alta. Em [12], o autor menciona que o SGD possui a vantagem de acabar com a redundância que o BGD pode ter e por conta disso, acaba aprendendo mais rápido. A parte estocástica do algoritmo ocorre através de um embaralhamento nos dados que é realizado em cada época do treinamento. Isso será melhor explicado no Capítulo 3.

A junção das principais características dos algoritmos BGD e SGD resulta no alagoritmo chamado *Mini-batch Gradient Descent*, ou em português Descida de Gradiente em mini-lote. Esse algoritmo realiza a atualização para cada mini-lote com n amostras da seguinte forma:

$$W^{k+1} = W^k - \eta \nabla_W C(W, x^{(i:i+n)}, y^{(i:i+n)}). \tag{2.3}$$

Assim como o SGD, ao final de cada época ocorre um embaralhamento nos dados. Iremos, por simplicidade, nos referir ao algoritmo Descida de Gradiente em mini-lote como SGD ao longo desse trabalho. Ao detalhar melhor esse algoritmo no Capítulo 3, já iremos nos referir a ele como SGD.

Alguns desafios enfrentados pelos algoritmos de descida citados acima, como escolha da taxa de aprendizagem e ficar preso em mínimos locais ruins, são explorados no trabalho [12].

Um problema que ocorre com o método SGD é a oscilação quando a busca se encontra em pontos onde a superfície se curva muito mais em uma dimensão do que em outra, algo que é comum em mínimos locais. Uma forma de resolver esse problema é com a utilização do método ***Momentum*** [13], que torna o aprendizado mais estável acelerando a convergência. Isso é feito através da inserção de um termo, o *momentum*, que quantifica o grau de importância da variação de peso do passo anterior comparado ao atual. A atualização é dada por:

$$W^{k+1} = W^k - v^{k+1} \tag{2.4}$$

onde $v^{k+1} = \gamma v^k + \eta \nabla_W C(W^k)$ e o termo do *momentum* γ é comumente colocado com o valor 0.9. Com essa atualização, o método acelera mais em direções mais relevantes. Em [12], o autor faz uma analogia com uma bola em um morro onde o *momentum*, nesse caso, empurra a bola morro abaixo. A bola irá acumular *momentum* a medida que ela rola no morro, ficando cada vez mais rápida naquela direção.

Ainda considerando a analogia do autor, um problema que pode acontecer com o método é um acumulo de aceleração na bola, enquanto ela desce o morro, fazendo com

que ela continue naquela direção, mesmo quando atinge o mínimo local, fazendo com que ela suba o outro morro. O método ***Nesterov Accelerated Gradient* (NAG)** [14] é um algoritmo que fornece ao *momentum* uma espécie de suavização tentando prever o instante que deve desacelerar olhando para o possível próximo ponto na sequência iterativa da descida de gradiente. A atualização com esse método acontece da seguinte forma:

$$W^{k+1} = W^k - v^{k+1} \tag{2.5}$$

onde $v^{k+1} = \gamma v^k + \eta \nabla_W C(W^k - \gamma v^k)$. Com essa atualização, previne-se uma aceleração muito alta ao longo das iterações.

O algoritmo **Adagrad** [15] é um algoritmo adaptativo que leva em consideração a geometria dos dados observados durante as iterações realizando passos iterativos com mais informações. Ao contrário dos outros algoritmos mencionados até o momento, o Adagrad realiza uma atualização na taxa de aprendizagem e ainda utiliza uma taxa diferente para cada peso a ser atualizado dentro de uma época. Esse procedimento fornece taxas de aprendizagem menores para características mais frequentes e taxas maiores para características menos frequentes. Seja g^k o gradiente da função custo no tempo k, ou seja, $g^k = \nabla_W C(W^k)$. Definimos G^k uma matriz diagonal onde:

$$G^k_{ii} = \sum_{j=1}^{k} (g^j_i)^2 \tag{2.6}$$

sendo g^j_i o gradiente da função custo no tempo j com respeito ao i-ésimo peso da rede. Por fim, a iteração do algoritmo é realizada da seguinte forma:

$$W^{k+1} = W^k - \eta (G^k + \epsilon I)^{(-1/2)} \nabla_W C(W^k) \tag{2.7}$$

onde $\epsilon > 0$ geralmente é tomado na ordem de 10^{-8}. Com isso, não há necessidade de grandes preocupações em relação a escolha da taxa de aprendizagem inicial η. Contudo, o acumulo dos gradientes ao quadrado em G^k irá tender a um número muito grande ao passar das épocas e, consequentemente, o termo que multiplica o gradiente irá tender a 0.

O algoritmo **Adadelta** [16] busca reduzir essa rápida diminuição da taxa de aprendizagem que ocorre no Adagrad. O algoritmo realiza tal feito, limitando a quantidade de termos considerados no somatório que define a matriz G^k. Ao invés de literalmente realizar o armazenamento de uma quantidade de termos g^k, o que é feito é uma espécie de amortecimento, calculando a média de $(g^k)^2$ no tempo k dando uma menor importância aos g^k mais antigos da seguinte forma:

$$E[(g^k)] = \gamma E[(g^{k-1})^2] + (1 - \gamma)(g^k)^2. \tag{2.8}$$

Para compatibilizarmos a notação utilizada pelos algoritmos Adagrad e Adadelta, chamando $v^{k+1} = \Delta W^k = -\eta g^k$ a atualização fica

$$W^{k+1} = W^k + \Delta W^k. \tag{2.9}$$

Denota-se por $RMS[g^k]$ o valor $\sqrt{E[(g^k)^2] + \epsilon}$. Define-se também um outro decaimento exponencial médio mas agora em relação ao quadrado da atualização dos parâmetros:

$$E[(\Delta W^k)^2] = \gamma E[(\Delta W^{k-1})^2] + (1-\gamma)(\Delta W^k)^2. \tag{2.10}$$

Com isso, define-se o erro quadrático médio das atualizações do parâmetro na forma:

$$RMS[\Delta W^k] = \sqrt{E[(\Delta W^k)^2] + \epsilon}. \tag{2.11}$$

Definidos esses termos, a regra de atualização do algoritmo Adadelta fica:

$$\begin{aligned} \Delta W^k &= -\frac{RMS[\Delta W^{k-1}]}{RMS[g^k]} g^k \\ W^{k+1} &= W^k + \Delta W^k. \end{aligned} \tag{2.12}$$

Dessa forma, não há necessidade de informar um valor de taxa de aprendizagem.

Outro algoritmo muito utilizado é o ***RMSprop*** que assim como o Adadelta, busca resolver o problema que ocorre com o decaimento rápido da taxa de aprendizagem do Adagrad. A atualização do *RMSprop* ocorre com a utlização de um amortecimento exponencial na soma dos quadrados dos termos de atualização da seguinte forma:

$$\begin{aligned} E[(g^k)^2] &= 0.9E[(g^{k-1})^2] + 0.1(g^k)^2 \\ W^{k+1} &= W^k - \frac{\eta}{\sqrt{E[(g^k)^2] + \epsilon} g^k}. \end{aligned} \tag{2.13}$$

Um outro algoritmo que computa taxas de aprendizagem adaptativas e que é muito utilizado é o algoritmo ***Adam*** (Adaptive Moment Estimation) [17]. Além de realizar um decaimento exponencial médio dos quadrados dos gradientes, assim como o Adadelta e o RMSprop, o Adam também realiza um decaimento semelhante para o termo dos gradientes passados, da seguinte forma:

$$\begin{aligned} m^k &= \beta_1 m^{k-1} + (1-\beta_1) g^k \\ v^k &= \beta_2 v^{k-1} + (1-\beta_2)(g^k)^2. \end{aligned} \tag{2.14}$$

onde $m^0 = v^0 = 0$. O termo m^k é uma estimativa do primeiro momento do gradiente, isto é, a média. O termo v^k é uma estimativa do segundo momento do gradiente, ou seja, da variância não centrada. Por conta de suas inicializações iguais a 0, existe uma tendência desses vetores em torno do ponto zero nas primeiras iterações e isso é intensificado quando as taxas de decaimento são pequenas, ou seja, quando β_1 e β_2 são próximas de 1. Um cálculo é realizado para correção desse viés nos dois momentos calculados:

$$\begin{aligned} \hat{m}^k &= \frac{m^k}{1-(\beta_1)^k} \\ \hat{v}^k &= \frac{v^k}{1-(\beta_2)^k}. \end{aligned} \tag{2.15}$$

Com esses termos definidos, a atualização dos pesos no algoritmo Adam fica da seguinte forma:

$$W^{k+1} = W^k - \frac{\eta}{\sqrt{\hat{v}^k} + \epsilon}\hat{m}^k. \tag{2.16}$$

Valores padrões de β_1 igual a 0.9, β_2 igual a 0.999 e ϵ igual a 10^{-8} são sugeridos pelos autores. Um estudo aprofundado sobre o método, como análise de convergência e alguns experimentos numéricos, são apresentados em [17], mostrando sua robustez na resolução de problemas na área de aprendizado de máquina.

Mais detalhes sobre os métodos mencionados acima podem ser encontrados nas referências apresentadas.

3 INTRODUÇÃO ÀS REDES NEURAIS E FUNDAMENTOS TEÓRICOS

Esse capítulo tem como objetivo apresentar as principais ideias a respeito de redes neurais importantes para o entendimento do trabalho.

O trabalho [18] foi uma das principais referências utilizadas para o assunto de redes neurais abordado neste capítulo. Nessa referência, o autor apresenta redes neurais de uma forma muito detalhada partindo dos princípios mais básicos, explicando de onde vem a inspiração para a construção e utilização de redes neurais. O trabalho abrange os principais tópicos em torno do assunto como, por exemplo, os processos de aprendizado e os diferentes tipos de arquiteturas utilizados na definição de redes neurais.

Neste capítulo, introduzimos a definição de neurônio artificial, explicamos como se representam redes neurais, deduzimos a expressão que define a função custo e também como se faz o treinamento de uma rede neural. Além disso, apresentamos em detalhes o método de Descida de Gradiente Estocástico. Ainda neste capítulo, relembramos o conceito de matriz Jacobiana, a fórmula de Sherman-Morrison, o produto de Hadamard, as condições de otimalidade de Karush-Kuhn-Tucker além da Regra da Cadeia em $\mathbb{R}^n$.

3.1 REDES NEURAIS

As Redes Neurais Artificiais (comumente chamadas de Redes Neurais) são estruturas matemáticas inspiradas no funcionamento das redes neurais biológicas do ser humano. O cérebro humano é uma espécie de supercomputador altamente complexo, não-linear e paralelo, capaz de realizar muitas tarefas, sejam simples ou complicadas, como por exemplo percepção e reconhecimento de padrões, em milésimos de segundos [18]. Um exemplo disso, é a visão humana como tarefa do processamento de informação. O sistema visual tem como função prover informação sobre os objetos e o ambiente à nossa volta para que possamos interagir com eles. O cérebro humano também realiza funções rotineiras que, por serem comuns, não paramos para pensar na complexidade da ação como, por exemplo, a de reconhecer rostos de pessoas já conhecidas mesmo em um ambiente que não seja familiar ou de olhar para a foto de um animal e reconhecer qual animal é aquele na foto.

A estrutura de uma rede neural é formada por nós, que são os neurônios, divididos em camadas e ligados por conexões, às quais são atribuídos pesos. Uma representação gráfica de uma rede neural com três camadas por ser vista na Figura 1. A primeira camada é chamada camada de entrada e possui dois neurônios, x_1 e x_2. A segunda camada é chamada camada oculta e possui os neurônios a_1^2, a_2^2 e a_3^2. A terceira e última camada é chamada camada de saída da rede e possui o neurônio a_1^3. No caso geral, em uma rede com N camadas, a primeira é a camada de entrada, a última é a camada de saída e as camadas intermediárias são as camadas ocultas.

Uma rede com várias camadas ocultas é chamada Rede Neural Profunda [19]. O problema básico que envolve redes neurais profundas é denominado Aprendizado Profundo que consiste na determinação dos pesos ótimos para essa rede. Os pesos ótimos são aqueles que, a partir de um conjunto de informações, fazem com que a rede produza uma resposta mais próxima possível daquela previamente conhecida.

Para entendermos melhor o funcionamento das redes neurais, iremos iniciar descrevendo o modelo dos neurônios que formam a base de uma rede neural.

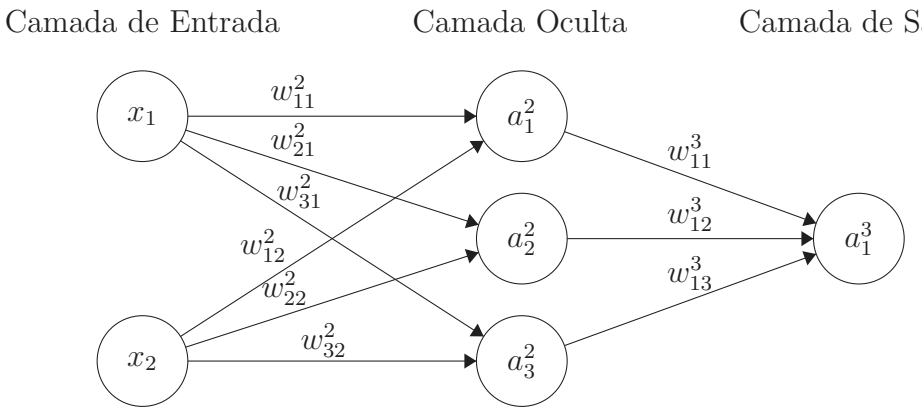


Figura 1 – Representação de uma Rede Neural Artificial.

3.1.1 Neurônio Artificial

Inspirado na estrutura e funcionamento do neurônio biológico, os neurônios artificiais são estruturas matemáticas que tentam reproduzir ações semelhantes às realizadas pelos neurônios humano. Em [18], é explicado o funcionamento do sistema nervoso humano e é feito uma correlação entre as características deste sistema humano com os componentes das redes neurais. De uma forma simplificada, um neurônio artificial é um objeto que recebe um conjunto de informações, trabalha essas informações utilizando funções matemáticas e passa o resultado para os neurônios da próxima camada

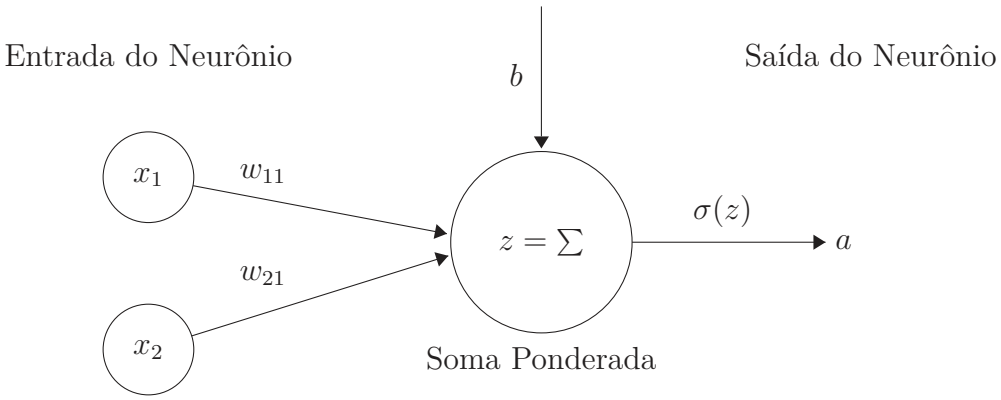


Figura 2 – Exemplo de um neurônio artificial.

A Figura 2 exemplifica um neurônio artificial. Esse neurônio poderia ser um dos neurônios da segunda camada da rede neural da Figura 1, onde agora destaca-se o *bias*, denotado pela letra b, que não havia sido representado anteriormente. O *bias*, ou viés em português, é um parâmetro que interfere na passagem da informação carregada por x_1 e x_2 para a camada seguinte. As entradas x_1 e x_2 são passadas para este neurônio através das conexões de pesos w_{11} e w_{21}. Com essas informações, o neurônio realiza uma soma ponderada da seguinte forma:

$$z = w_{11}x_1 + w_{21}x_2 + b. \tag{3.1}$$

Por fim, o neurônio aplica uma função não-linear, denominada função de ativação, gerando um resultado

$$a = \sigma(z). \tag{3.2}$$

Portanto, um neurônio que recebe as entradas x_1 e x_2, por meio das conexões com pesos w_{11} e w_{21}, respectivamente e com um *bias* b, tem como saída o seguinte valor:

$$a = \sigma(w_{11}x_1 + w_{21}x_2 + b). \tag{3.3}$$

3.1.2 Representações da Rede Neural

Nesta seção, apresentaremos alguns dos objetos matemáticos utilizados para representar as estruturas da rede neural da Figura 1. Com isso, poderemos escrever a saída da rede através de uma função matemática de forma mais clara. Vamos enumerar as camadas da nossa rede a partir do número 1 para a camada de entrada, numero 2 para a camada seguinte e assim sucessivamente. Na rede que estamos considerando, temos 3 camadas sendo a camada 1 a camada de entrada, a camada oculta é a camada 2 e a camada de saída é a camada 3.

Os pesos associados as ligações que conectam duas camadas podem ser representados por uma matriz. Supondo uma rede com N camadas, chamaremos de W^L, $L \in \{2, 3, \ldots, N\}$, a matriz que contém os pesos associados às ligações que conectam as camadas $L-1$ e L. Os elementos dessa matriz são os pesos w_{ij}^L onde, w_{ij}^L representa o peso da conexão que liga o neurônio i da camada L com o neurônio j da camada $L-1$. Com isso, a nossa rede neural possui as matrizes W^2 e W^3 definidas da seguinte forma:

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \\ w_{31}^2 & w_{32}^2 \end{bmatrix} \quad \text{e} \quad W^3 = \begin{bmatrix} w_{11}^3 & w_{12}^3 & w_{13}^3 \end{bmatrix}. \tag{3.4}$$

Podemos também representar os *biases* através de uma matriz, mais precisamente um vetor. Mesmo que não esteja representado na Figura 1, cada um dos neurônios das camadas 2 e 3 possui um *bias*. Dessa forma, chamaremos de B^L o vetor que contém os *biases* da camada L. Os elementos desse vetor são os *biases* b_i^L que representa o *bias* do

neurônio i, ou i-ésimo neurônio, da camada L. Daí, a rede neural da Figura 1 possui as matrizes B^2 e B^3 definidas como

$$B^2 = \begin{bmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \end{bmatrix} \quad \text{e} \quad B^3 = \begin{bmatrix} b_1^3 \end{bmatrix}. \tag{3.5}$$

Com essas estruturas definidas, podemos fazer a computação da rede através de produtos matriciais. Por exemplo, se tivermos uma entrada $X = [x_1, x_2]^T$, obtemos a soma ponderada dos neurônios da segunda camada da seguinte forma:

$$Z^2 = W^2X + B^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \\ w_{31}^2 & w_{32}^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \end{bmatrix} = \begin{bmatrix} w_{11}^2 x_1 + w_{12}^2 x_2 + b_1^2 \\ w_{21}^2 x_1 + w_{22}^2 x_2 + b_2^2 \\ w_{31}^2 x_1 + w_{32}^2 x_2 + b_3^2 \end{bmatrix}. \tag{3.6}$$

Assim, Z^L fica definida como a matriz que contém as somas ponderadas das ativações da camada $(L-1)$ com seus respectivos pesos com a adição do *bias* da camada L.

Seja $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ uma função de ativação. A saída A^L, da camada L da rede, é dada por $\sigma(Z^L)$, onde $A_{ij}^L = \sigma(Z_{ij}^L)$. No caso da nossa rede, por exemplo

$$Z^2 = \begin{bmatrix} z_1^2 \\ z_2^2 \\ z_3^2 \end{bmatrix} \quad \text{e} \quad A^2 = \sigma(Z^2) = \begin{bmatrix} \sigma(z_1^2) \\ \sigma(z_2^2) \\ \sigma(z_3^2) \end{bmatrix}. \tag{3.7}$$

Com isso, conseguimos uma forma geral para Z^L dada por

$$Z^L = W^L A^{L-1} + B^L \tag{3.8}$$

onde $A^1 = X$.

Finalmente, conseguimos escrever a saída da nossa rede neural através de uma função $y : \mathbb{R}^2 \longrightarrow \mathbb{R}$, para a entrada $X = [x_1, x_2]^T$, definida por:

$$\begin{aligned} y(X) = \sigma(W^3\sigma(W^2X + B^2) + B^3) = \sigma(W^3\sigma(Z^2) + B^3) = \\ = \sigma(W^3A^2 + B^3) = \sigma(Z^3) = A^3. \end{aligned} \tag{3.9}$$

3.1.3 Função de Ativação

No final da seção anterior, foi utilizada uma função genérica σ que chamamos de função de ativação. A função de ativação em uma rede neural é uma função aplicada na saída do neurônio. O seu papel nas redes neurais é inserir uma não linearidade na função saída y da rede que seria linear sem a aplicação das funções de ativação. A não linearidade

faz-se necessária pelo fato de que funções não lineares são capazes de separar o espaço $\mathbb{R}^n$ de diversas formas além de semi espaços que é o caso quando se tem apenas hiperplanos envolvidos. Dessa forma, a rede pode ser usada para resolver problemas não triviais que ocorrem mesmo na presença de um número pequeno de nós. Em [20] é apresentada uma análise de desempenho de uma rede utilizando-se diferentes funções de ativações.

Algumas das funções de ativação mais utilizadas são:

$$\text{Tangente hiperbólica} : \sigma(x) = \tanh(x) \tag{3.10}$$

$$\text{Função de Heaviside} : \sigma(x) = \begin{cases} 1, \text{ se x} > 0 \\ 0, \text{ caso contrário} \end{cases} \tag{3.11}$$

$$\text{Função logística ou sigmoide} : \sigma(x) = \frac{1}{1+e^{-x}} \tag{3.12}$$

$$\text{ReLu} : \sigma(x) = \max\{0, x\}. \tag{3.13}$$

De forma geral, a lei das funções de ativação podem mudar com a introdução de coeficientes. Por exemplo, as funções em (3.11) e (3.12) podem variar, dependendo do problema, de maneira a terem seu conjunto imagem estendido para o intervalo $[-1, +1]$. É possível entender melhor essa extensão analisando os gráficos dessas funções na Figura 3.

Entre as funções de ativação apresentadas, destacamos a função sigmoide, que é uma das mais utilizadas na construção de redes neurais artificiais. Essa função se caracteriza por apresentar um bom balanceamento entre comportamento linear e não-linear, além de ser uma função estritamente crescente.

3.1.4 Função Custo

Até esse momento, a rede neural que estamos consideramos recebe uma entrada X e retorna uma saída $y(X)$. Podemos atribuir às conexões da rede pesos e *biases* aleatórios, o que faria altamente improvável que a nossa rede tivesse um bom acerto no sentido de produzir uma saída próxima daquela já esperada. Uma das formas de treinamento de uma rede é o aprendizado supervisionado, onde a partir de um conjunto de dados conhecidos, determina-se os pesos e *biases* de modo que a saída da rede seja mais próxima possível da saída esperada. Nesse caso, o que é feito é uma comparação entre a saída $y(x^{(i)})$ que a rede está fornecendo para uma entrada $x^{(i)}$ e sua resposta esperada $a(x^{(i)})$. Essa comparação é realizada através de uma função C chamada função custo ou função de perda que exibe essa diferença entre a saída produzida pela rede e a saída desejada. Um exemplo de tal função é o erro quadrático médio definido por:

$$C(W, B, X, a(X)) = C(W, B) = \frac{1}{2n}\sum_{i=1}^{n} ||y(x^{(i)}) - a(x^{(i)})||^2. \tag{3.14}$$

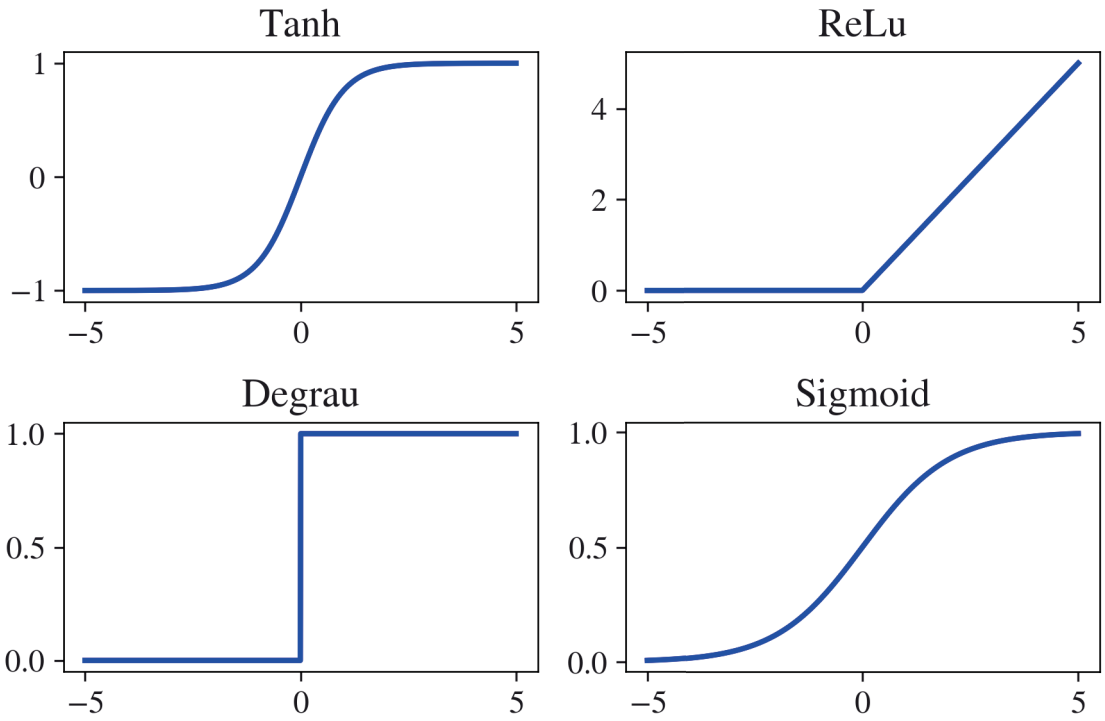


Figura 3 – Algumas funções de ativação.

onde $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, $a(X) = \{a(x^{(1)}), a(x^{(2)}), \ldots, a(x^{(n)})\}$ e $y(x^{(i)})$ é a saída da rede para a entrada $x^{(i)}$.

Os parâmetros X e $a(X)$ que aparecem na expressão $C(W, B, X, a(X))$ foram omitidos na expressão $C(W, B)$ para destacar que as variáveis da função custo C são na realidade W e B, no sentido explicitado pela fórmula (3.9).

Outro exemplo de função custo, é a função erro de entropia cruzada definida por:

$$C(W, B) = -\frac{1}{2n} \sum_{i=1}^{n} \left\langle y(x^{(i)}), \log a(x^{(i)}) \right\rangle . \tag{3.15}$$

onde $< \cdot, \cdot >$ representa o produto interno e *log* está sendo aplicado em cada coordenada de $a(x^{(i)})$.

A função erro de entropia cruzada, cujas ideias se baseiam no Princípio da Máxima Entropia [21, 22], pode ser definida, para duas distribuições de probabilidades **p** e **q**, da seguinte forma [23]:

$$H(p, q) = E_p[-\log q] = H(p) + D_{KL}(p \| q) \tag{3.16}$$

onde $H(p)$ é a entropia da distribuição de probabilidade **p**, e $D_{KL}(p\|q)$ é a divergência Kullback-Leibler da distribuição de probabilidade **q** dado **p**. Essas funções são definidas, no caso em que as distribuições são discretas, por:

$$H(p) = \sum_{i} p_i \log p_i \tag{3.17}$$

$$D_{KL}(p\|q) = \sum_i p_i \log \frac{p_i}{q_i}. \tag{3.18}$$

Neste caso, a função erro entropia cruzada fica:

$$H(p,q) = -\sum_i p_i \log q_i. \tag{3.19}$$

3.1.5 Treinamento de redes neurais

O treinamento de redes neurais que vamos tratar nesse trabalho pode ser considerando aprendizado supervisionado que acontece quando tem-se um conjunto finito de dados de entrada que chamaremos $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$ associado com as saídas chamadas $a(X) = \{a(x^{(1)}), a(x^{(2)}), \ldots, a(x^{(n)})\}$. A partir disso, a rede precisa aprender como fazer essa associação, isto é, quando a entrada $x^{(i)}$ é dada, a saída desejada é $a(x^{(i)})$. Por exemplo, em um problema de classificação podemos pensar que $x^{(i)} \in \mathbb{R}^m$ e $a(x^{(i)})$ pertence a um certo conjunto $\mathcal{S}$ e o objetivo do treinamento seria definir uma função $y : \mathbb{R}^m \longrightarrow \mathcal{S}$ de modo que $y(x^{(i)})$ seja próximo de $a(x^{(i)})$ para cada $i \in \{1, 2, \ldots, n\}$.

Consideremos uma rede neural com a camada de entrada tendo n neurônios e a camada de saída possuindo m neurônios e associada a ela a função $y : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. Seja dado um conjunto de treinamento $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(k)}\}$, onde $x^{(i)} \in \mathbb{R}^n$, e um conjunto associado $a(X) = \{a(x^{(1)}), a(x^{(2)}), a(x^{(k)})\}$, onde $a(x^{(i)}) \in \mathbb{R}^m$, $\forall i \in \{1, \ldots, k\}$. Com isso, conseguimos definir a função custo $C(W, B)$ que quantificará o quanto as saídas $y(x^{(i)})$ estão próximas dos dados de treinamento $a(x^{(i)})$. As variáveis da função custo C são os pesos W e os *biases* B da rede. O treinamento consiste em encontrar um conjunto de pesos W e *biases* B que minimizem a função custo $C(W, B)$.

3.1.6 Simplificação da função da rede neural

Neste trabalho, vamos considerar uma rede neural contendo apenas os pesos como parâmetros. Isso é feito, transformando-se os *biases* em pesos da rede. Tomando como referencia a Equação (3.1), típica de um neurônio, definindo $x_3 = 1$ e $w_{31} = b$, esta equação pode ser reescrita como:

$$z = w_{11}x_1 + w_{21}x_2 + w_{31}x_3. \tag{3.20}$$

Dessa forma, um neurônio denominado x_3 com valor fixo igual a 1 foi adicionado na primeira camada. Foi então criado um novo peso na rede associado a conexão entre o neurônio x_3 e o neurônio da segunda camada. Com isso, aumentamos o número de pesos da rede mas eliminamos os *biases* como parâmetros explícitos.

Essa modificação deve ser realizada em todas as camadas da rede para que eliminemos todos os *biases*. O processo portanto, consiste em adicionar um neurônio na camada de entrada e em cada uma das camadas ocultas e manter o seu valor fixo igual a 1 durante todo o processo de treinamento e computação da rede. Com a adição desse novo

neurônio em todas as camadas, exceto na camada de saída, surgem novos pesos referentes as conexões entre esse novo neurônio de cada camada e todos os neurônios da camada seguinte.

3.2 DESCIDA DE GRADIENTE ESTOCÁSTICA

A Descida de Gradiente é um algoritmo iterativo de otimização de primeira ordem para encontrar mínimos de funções diferenciáveis. A ideia intuitiva do método é a utilização da direção oposta à do gradiente, que é uma direção de descida, para atualizar os termos da sequencia gerada pelo algoritmo. Uma formulação do algoritmo é apresentada a seguir.

Seja $F : \mathbb{R}^m \longrightarrow \mathbb{R}$ uma função diferenciável. Dado um ponto $w_n \in \mathbb{R}^m$, o algoritmo obtém o ponto w_{n+1} utilizando a seguinte regra de atualização:

$$w_{n+1} = w_n - \eta \nabla F(w_n) \tag{3.21}$$

onde $\eta > 0$ é um valor real que representa o tamanho do passo que será dado na direção $-\nabla F(w_n)$. As iterações do algoritmo acontecem até que um critério de parada seja atingido. Esse critério de parada pode ser definido por um número máximo de iterações ou a verificação do tamanho da direção em cada ponto, caso em que o algoritmo será interrompido quando $\|\nabla F(w_n)\| \leq \epsilon$ para algum $\epsilon > 0$ pré definido.

No contexto do treinamento de redes neurais, o algoritmo de Descida de Gradiente funciona da seguinte forma: o conjunto original de dados é repartido em p subconjuntos de mesmo tamanho chamados de mini-lotes. Escolhido um número natural N e um vetor de pesos inicial W_0, após p iterações, o algoritmo de Descida de Gradiente produz o vetor W_1 finalizando uma etapa denominada época 1. Na época 2, a partir de W_1, após p iterações, o algoritmo encontra o vetor W_2. Para $1 < j < N$, a época j consiste de, a partir, de W_{j-1} encontrar o vetor W_j. Em cada época j, a partir do ponto inicial W_{j-1}, são calculados p vetores de pesos utilizando-se os p mini-lotes.

Suponhamos que $\mathcal{D} = \{(X_1, a(X_1)), (X_2, a(X_2)), \ldots, (X_k, a(X_k))\}$ seja o conjunto de dados. Decidido que o conjunto $\mathcal{D}$ será repartido em p mini-lotes, teremos que o tamanho de cada mini-lote será $q = k/p$. Dessa forma, os mini-lotes podem ser representados por:

- $\mathcal{D}_1 = \{(X_1, a(X_1)), (X_2, a(X_2)), \ldots, (X_q, a(X_q))\}$;
- $\mathcal{D}_2 = \{(X_{q+1}, a(X_{q+1})), (X_{q+2}, a(X_{q+2})), \ldots, (X_{2q}, a(X_{2q}))\}$;
  $\ldots$
- $\mathcal{D}_p = \{(X_{(p-1)q+1}, a(X_{(p-1)q+1})), (X_{(p-1)q+2}, a(X_{(p-1)q+2})), \ldots, (X_k, a(X_k))\}$.

Com essa notação, em cada época j, a iteração i do algoritmo de Descida do Gradiente é realizada utilizando-se o mini-lote D_i. Como foi mostrado, a função custo de uma rede

neural é definida a partir de um conjunto de dados. Portanto, em cada época, teremos p funções custo diferentes, uma para cada um dos mini-lotes, que não se alteram de época para época. Todo esse processo é denominado algoritmo de Descida de Gradiente em Lote (Batch Gradient Descent).

Quando o conjunto de dados é muito grande, o aprendizado das redes pode ser muito lento utilizando a Descida de Gradiente em Lote pois o algoritmo pode ficar preso em mínimos locais das funções relacionadas aos diferentes mini-lotes. Uma alternativa para contornar esse problema é a utilização do algoritmo **Descida de Gradiente Estocástica** ou **SGD**, sigla em inglês para *Stochastic Gradient Descent* [24, 25]. A única alteração realizada em comparação ao algoritmo de Descida de Gradiente em Lote é, em cada uma das épocas, um embaralhamento da ordem do conjunto de dados originais $\mathcal{D}$ antes da separação em lotes. Feito isso, em cada época, o algoritmo terá p diferentes funções objetivos, que variam de época para época devido ao embaralhamento, para minimizar e consegue evitar de uma forma melhor os mínimos locais das funções objetivos relativas a alguns mini-lotes.

3.3 TEOREMAS E DEFINIÇÕES BÁSICAS

Nesta seção, apresentaremos alguns teoremas e definições utilizados neste trabalho. Um dos objetivos é deixar explicitado as notações que empregamos nos próximos capítulos.

3.3.1 Jacobiano e Regra da Cadeia

Definição 3.3.1. Seja uma função $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ continuamente diferenciável. Dado um ponto $a \in \mathbb{R}^n$, a *matriz jacobiana de f no ponto a*, indicada por $Jf(a)$ é definida por:

$$Jf(a) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(a) & \frac{\partial f_1}{\partial x_2}(a) & \dots & \frac{\partial f_1}{\partial x_n}(a) \\ \frac{\partial f_2}{\partial x_1}(a) & \frac{\partial f_2}{\partial x_2}(a) & \dots & \frac{\partial f_2}{\partial x_n}(a) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(a) & \frac{\partial f_m}{\partial x_2}(a) & \dots & \frac{\partial f_m}{\partial x_n}(a) \end{bmatrix} = \begin{bmatrix} \nabla f_1(a) \\ \nabla f_2(a) \\ \vdots \\ \nabla f_m(a) \end{bmatrix} \tag{3.22}$$

cujas linhas são constituídas pelas derivadas parciais das funções coordenadas de f.

Teorema 3.3.1. *(Regra da Cadeia [26]) Sejam $U \subset \mathbb{R}^n$ e $V \subset \mathbb{R}^m$ conjuntos abertos, $f : U \longrightarrow \mathbb{R}^m$ uma aplicação diferenciável no ponto $a \in U$, com $f(U) \subset V$ e $g : V \longrightarrow \mathbb{R}^p$ uma aplicação diferenciável no ponto $b = f(a) \in V$.*

Então, a aplicação composta $g \circ f : U \longrightarrow \mathbb{R}^p$ é diferenciável no ponto a e sua derivada é

$$J(g \circ f)(a) = Jg(b)Jf(a)$$

3.3.2 Fórmula de Sherman-Morrison

A fórmula de Sherman-Morrison [27, 28, 29], calcula a matriz inversa da soma de uma matriz inversível A e o produto tensorial, uv^t, dos vetores u e v.

Teorema 3.3.2. *(Fórmula de Sherman-Morrison) Sejam $A \in \mathbb{R}^{n\times n}$ uma matriz inversível e $u, v \in \mathbb{R}^{n\times 1}$. Então, $A + uv^t$ é inversível se e somente se $1 + v^tA^{-1}u \neq 0$ e a fórmula para o cálculo da inversa é dada por:*

$$(A + uv^t)^{-1} = A^{-1} - \frac{A^{-1}uv^tA^{-1}}{1 + v^tA^{-1}u}$$

3.3.3 Produto de Hadamard

O Produto de Hadamard é uma operação de multiplicação de matrizes especialmente utilizada pelo algoritmo *backpropagation*.

Definição 3.3.2. (Produto de Hadamard) Sejam A, $B \in \mathbb{R}^{m\times n}$. O produto de Hadamard entre as matrizes A e B, denotado por $A \odot B$, é definido como:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ b_{31} & b_{32} & \cdots & b_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ a_{31}b_{31} & a_{32}b_{32} & \cdots & a_{3n}b_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{bmatrix} \tag{3.23}$$

Em outras palavras, o Produto de Hadamard faz a multiplicação de matrizes elemento por elemento.

3.4 OTIMIZAÇÃO E CONDIÇÕES DE KARUSH-KUHN-TUCKER (KKT)

Otimização é uma área da Matemática que, dada uma função $F : X \subset \mathbb{R}^n \longrightarrow \mathbb{R}$, chamada função objetivo, procura pontos $x^* \in X$ tais que $F(x^*) \leq F(x)$ para todo $x \in X$. Se $X = \mathbb{R}^n$, o problema é dito irrestrito. Caso contrário, isto é, se X é um subconjunto próprio de $\mathbb{R}^n$, temos um problema com restrições.

Em geral, um problema de otimização é escrito da seguinte forma:

$$(P) \quad \begin{cases} \text{minimizar} & f(x) \\ \text{sujeito a} & g(x) \leq 0 \\ \text{e} & h(x) = 0, \end{cases} \tag{3.24}$$

onde $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ e $h : \mathbb{R}^n \longrightarrow \mathbb{R}^p$ com $g(x) = (g_1(x), g_2(x), \ldots, g_m(x))^t$ e $h(x) = (h_1(x), h_2(x), \ldots, h_p(x))^t$. A notação $g(x) \leq 0$ significa $g_i(x) \leq 0$ para todo $i \in \{1, 2, \ldots, m\}$. Uma restrição g_i será dita ativa no ponto x se $g_i(x) = 0$.

Supondo que x^* é uma solução de (P), que as funções f, g_i, h_j são continuamente diferenciáveis em x^* e que $\nabla h_1(x^*), \nabla h_2(x^*), \ldots, \nabla h_p(x^*)$ e $\nabla g_i(x^*)$ para as restrições

ativas são linearmente independentes, as condições necessárias de primeira ordem de Karush-Kuhn-Tucker (KKT) [30] garantem a existência de vetores $\lambda^* = (\lambda_1^*, \ldots, \lambda_m^*)$ e $\mu^* = (\mu_1^*, \ldots \mu_p^*)$, chamados multiplicadores de KKT, que satisfazem:

- $\nabla f(x^*) + Jg(x^*)^t\lambda^* + Jg(x^*)^t\mu^* = 0$ (**estacionariedade**)
- $G(x^*)\lambda^* = 0$ (**folga complementar**)
- $\lambda^* \geq 0$ (**viabilidade dual**)
- $g(x^*) \leq 0$ (**viabilidade primal**)
- $h(x^*) = 0$ (**viabilidade primal**)

onde $G(x^*)$ é uma matriz diagonal introduzida para simplificar a notação. Seus elementos são $G(x^*)_{ii} = g_i(x^*)$. Dessa forma, a condição de folga complementar é equivalente a $\lambda_i g_i(x^*) = 0$ para $i = 1, \ldots, m$.

4 *BACKPROPAGATION*

Dada uma rede neural, o objetivo é encontrar os pesos ou coeficientes ótimos da rede, que são aqueles que produzem o modelo mais eficiente no que diz respeito às respostas mais precisas possíveis para as entradas conhecidas de forma que novos dados possam produzir informações bem acuradas.

Esses pesos ótimos são obtidos através do treinamento da rede que consiste de, entre outras coisas, um processo de otimização, como já visto no capítulo 3. Uma das possibilidades é a utilização do método do gradiente, que requer o cálculo do gradiente da função custo associada a rede. O cálculo desse gradiente pode ser feito com a utilização de uma técnica chamada *backpropagation*, primeiramente introduzida no trabalho [4] como um novo procedimento de treinamento para redes neurais.

O *backpropagation* é uma técnica que se baseia na Regra da Cadeia. Porém, se o cálculo do gradiente for realizado de forma convencional, isto é, determinando-se as derivadas parciais $\frac{\partial C}{\partial w_{ij}}$ uma a uma, o processo computacional resultará extremamente custoso. Ao invés disso, o *backpropagation* pode ser realizado em duas etapas chamadas *forward* e *backward* que simplificam de maneira significativa o cálculo do ∇C.

Neste capítulo pretendemos detalhar as etapas do *backpropagation*, explicitando os cálculos envolvidos com o objetivo de prover uma apresentação mais amigável da técnica, que não é de fácil entendimento. Para isso, iremos considerar um exemplo mais simples que consiste de uma rede neural com 2 neurônios na entrada, 2 camadas ocultas com 3 neurônios cada e a camada de saída com 2 neurônios, como mostrado na Figura 4.

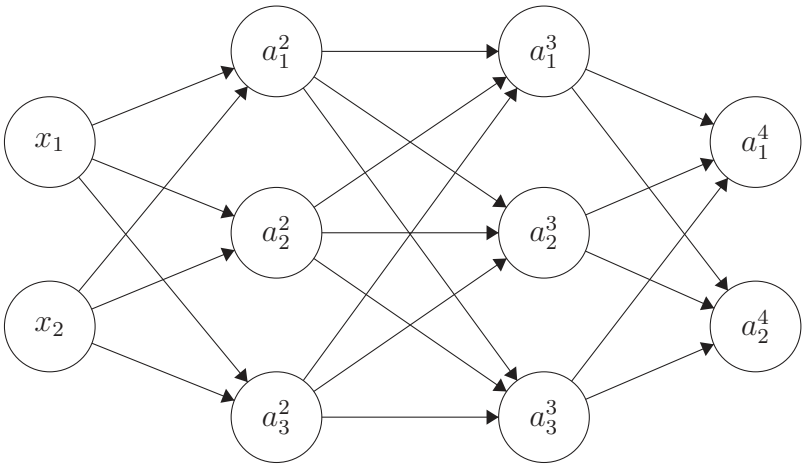


Figura 4 – Rede Neural Artificial utilizada como exemplo.

4.1 FASE *FORWARD*

A partir de um conjunto de pesos inicial e um conjunto de dados selecionados, na fase *forward*, é realizada a computação da rede, armazenando-se os vetores e matrizes

computados em cada uma das etapas intermediárias e o cálculo da função custo C, obtendo-se também ao final os pesos W, os vetores Z e os vetores A.

Considerando o exemplo utilizado, teremos o vetor de entrada

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \tag{4.1}$$

os pesos da rede

$$W^2 = \begin{bmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^2_{31} & w^2_{32} \end{bmatrix}, \quad W^3 = \begin{bmatrix} w^3_{11} & w^3_{12} & w^3_{13} \\ w^3_{21} & w^3_{22} & w^3_{23} \\ w^3_{31} & w^3_{32} & w^3_{33} \end{bmatrix}, \quad W^4 = \begin{bmatrix} w^4_{11} & w^4_{12} & w^4_{13} \\ w^4_{21} & w^4_{22} & w^4_{23} \end{bmatrix}, \tag{4.2}$$

os vetores Z^l e suas respectivas ativações A^l

$$Z^2 = W^2 x = \begin{bmatrix} w^2_{11}x_1 + w^2_{12}x_2 \\ w^2_{21}x_1 + w^2_{22}x_2 \\ w^2_{31}x_1 + w^2_{32}x_2 \end{bmatrix} = \begin{bmatrix} z^2_1 \\ z^2_2 \\ z^2_3 \end{bmatrix}, \tag{4.3}$$

$$A^2 = \sigma(Z^2) = \begin{bmatrix} \sigma(z^2_1) \\ \sigma(z^2_2) \\ \sigma(z^2_3) \end{bmatrix}, \tag{4.4}$$

$$Z^3 = W^3 A^2 = \begin{bmatrix} w^3_{11}\sigma(z^2_1) + w^3_{12}\sigma(z^2_2) + w^3_{13}\sigma(z^2_3) \\ w^3_{21}\sigma(z^2_1) + w^3_{22}\sigma(z^2_2) + w^3_{23}\sigma(z^2_3) \\ w^3_{31}\sigma(z^2_1) + w^3_{32}\sigma(z^2_2) + w^3_{33}\sigma(z^2_3) \end{bmatrix} = \begin{bmatrix} z^3_1 \\ z^3_2 \\ z^3_3 \end{bmatrix}, \tag{4.5}$$

$$A^3 = \sigma(Z^3) = \begin{bmatrix} \sigma(z^3_1) \\ \sigma(z^3_2) \\ \sigma(z^3_3) \end{bmatrix}, \tag{4.6}$$

$$Z^4 = W^4 A^3 = \begin{bmatrix} w^4_{11}\sigma(z^3_1) + w^4_{12}\sigma(z^3_2) + w^4_{13}\sigma(z^3_3) \\ w^4_{21}\sigma(z^3_1) + w^4_{22}\sigma(z^3_2) + w^4_{23}\sigma(z^3_3) \end{bmatrix} = \begin{bmatrix} z^4_1 \\ z^4_2 \end{bmatrix}, \tag{4.7}$$

$$A^4 = \sigma(Z^4) = \begin{bmatrix} \sigma(z^4_1) \\ \sigma(z^4_2) \end{bmatrix}. \tag{4.8}$$

A saída A^4 da rede do nosso exemplo é uma série de composições e multiplicações de matrizes o que torna o cálculo de sua derivada não trivial. Para deixar esse fato mais evidente podemos reescrever A^4 da seguinte forma:

$$A^4 = \sigma(W^4\sigma(W^3\sigma(W^2 x))). \tag{4.9}$$

Considerando o erro quadrático médio, nossa função custo será dada por

$$C(W, X) = \frac{1}{2n} \sum_{x \in X} ||A^4(W, x) - a(x)||^2 \tag{4.10}$$

onde o somatório é feito em relação a cada amostra x do conjunto de dados, $a(x)$ é a resposta esperada para x, n é a quantidade de amostras disponíveis nesse conjunto, W é o conjunto de pesos da rede e X é o conjunto de dados.

4.2 FASE *BACKWARD*

Na fase *backward* é realizado o cálculo do gradiente, iniciando-se na ultima camada, utilizando-se as derivadas encontradas nessa camada no cálculo das derivadas da penúltima camada e assim por diante até chegar na primeira camada.

Para mostrar o processo da fase *backward*, iremos realizar o cálculo das derivadas parciais separadamente para enxergarmos a fórmula final do gradiente. Para simplificar os cálculos, será considerado apenas um dos termos do somatório da função custo C e desprezaremos a constante $\frac{1}{n}$. Isso pode ser feito sem perda de generalidade, dada a propriedade da soma de derivadas e de multiplicação por uma constante. O somatório e a constante serão incluídos no cálculo final quando for considerado todo o conjunto de dados de uma só vez. Dessa forma, a função custo fica

$$C(W,x) = \frac{1}{2}||A^4(W,x) - a(x)||^2, \tag{4.11}$$

isto é, estamos calculando o custo da rede para apenas uma amostra $(x, a(x))$ do nosso conjunto de dados. Começando com os pesos da matriz W^4 temos:

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{11}^4} &= \frac{\partial}{\partial w_{11}^4}\left(\frac{1}{2}\left((\sigma(z_1^4) - a_1(x))^2 + (\sigma(z_2^4) - a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4) - a_1(x))\frac{\partial \sigma(z_1^4)}{\partial w_{11}^4} + (\sigma(z_2^4) - a_2(x))\frac{\partial \sigma(z_2^4)}{\partial w_{11}^4} = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{11}^4} + (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{11}^4} = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\sigma(z_1^3) + (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4)\cdot 0 = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\sigma(z_1^3)
\end{aligned} \tag{4.12}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{12}^4} &= \frac{\partial}{\partial w_{12}^4}\left(\frac{1}{2}\left((\sigma(z_1^4) - a_1(x))^2 + (\sigma(z_2^4) - a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4) - a_1(x))\frac{\partial \sigma(z_1^4)}{\partial w_{12}^4} + (\sigma(z_2^4) - a_2(x))\frac{\partial \sigma(z_2^4)}{\partial w_{12}^4} = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{12}^4} + (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{12}^4} = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\sigma(z_2^3) + (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4)\cdot 0 = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\sigma(z_2^3)
\end{aligned} \tag{4.13}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{13}^4} &= \frac{\partial}{\partial w_{13}^4}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{13}^4}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{13}^4} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{13}^4}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{13}^4} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\sigma(z_3^3)+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\cdot 0 = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\sigma(z_3^3)
\end{aligned} \tag{4.14}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{21}^4} &= \frac{\partial}{\partial w_{21}^4}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{21}^4}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{21}^4} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{21}^4}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{21}^4} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\cdot 0+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\cdot\sigma(z_1^3) = \\
&= (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\sigma(z_1^3)
\end{aligned} \tag{4.15}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{22}^4} &= \frac{\partial}{\partial w_{22}^4}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{22}^4}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{22}^4} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{22}^4}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{22}^4} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\cdot 0+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\cdot\sigma(z_2^3) = \\
&= (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\sigma(z_2^3)
\end{aligned} \tag{4.16}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{23}^4} &= \frac{\partial}{\partial w_{23}^4}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{23}^4}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{23}^4} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{23}^4}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{23}^4} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\cdot 0+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\cdot\sigma(z_3^3) = \\
&= (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\sigma(z_3^3)
\end{aligned} \tag{4.17}$$

De maneira resumida, temos:

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{11}^4} &= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\sigma(z_1^3) \\
\frac{\partial C(W,x)}{\partial w_{12}^4} &= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\sigma(z_2^3) \\
\frac{\partial C(W,x)}{\partial w_{13}^4} &= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4)\sigma(z_3^3) \\
\frac{\partial C(W,x)}{\partial w_{21}^4} &= (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4)\sigma(z_1^3) \\
\frac{\partial C(W,x)}{\partial w_{22}^4} &= (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4)\sigma(z_2^3) \\
\frac{\partial C(W,x)}{\partial w_{23}^4} &= (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4)\sigma(z_3^3)
\end{aligned} \tag{4.18}$$

Dessa forma, conseguimos escrever uma matriz $\frac{\partial C}{\partial W^4}$, onde cada elemento (i,j) é a derivada $\frac{\partial C(W,x)}{\partial w_{ij}^4}$, da seguinte forma:

$$\begin{aligned}
\frac{\partial C}{\partial W^4} &= \left(\begin{bmatrix} \sigma(z_1^4) - a_1(x) \\ \sigma(z_2^4) - a_2(x) \end{bmatrix} \odot \begin{bmatrix} \sigma'(z_1^4) \\ \sigma'(z_2^4) \end{bmatrix} \right) \begin{bmatrix} \sigma(z_1^3) & \sigma(z_2^3) & \sigma(z_3^3) \end{bmatrix} = \\
&= \left(\frac{\partial C}{\partial A^4} \odot \sigma'(Z^4) \right) \left(A^3 \right)^T
\end{aligned} \tag{4.19}$$

onde $\frac{\partial C}{\partial A^4}$ denota

$$\begin{bmatrix} \sigma(z_1^4) - a_1(x) \\ \sigma(z_2^4) - a_2(x) \end{bmatrix}. \tag{4.20}$$

Iremos agora, realizar as derivadas com relação aos pesos de W^3

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{11}^3} &= \frac{\partial}{\partial w_{11}^3} \left(\frac{1}{2} \left((\sigma(z_1^4) - a_1(x))^2 + (\sigma(z_2^4) - a_2(x))^2 \right) \right) = \\
&= (\sigma(z_1^4) - a_1(x)) \frac{\partial \sigma(z_1^4)}{\partial w_{11}^3} + (\sigma(z_2^4) - a_2(x)) \frac{\partial \sigma(z_2^4)}{\partial w_{11}^3} = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4) \frac{\partial z_1^4}{\partial w_{11}^3} + (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4) \frac{\partial z_2^4}{\partial w_{11}^3} = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4) w_{11}^4 \sigma'(z_1^3)\sigma(z_1^2) + \\
&+ (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4) w_{21}^4 \sigma'(z_1^3)\sigma(z_1^2)
\end{aligned} \tag{4.21}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{12}^3} &= \frac{\partial}{\partial w_{12}^3} \left(\frac{1}{2} \left((\sigma(z_1^4) - a_1(x))^2 + (\sigma(z_2^4) - a_2(x))^2 \right) \right) = \\
&= (\sigma(z_1^4) - a_1(x)) \frac{\partial \sigma(z_1^4)}{\partial w_{12}^3} + (\sigma(z_2^4) - a_2(x)) \frac{\partial \sigma(z_2^4)}{\partial w_{12}^3} = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4) \frac{\partial z_1^4}{\partial w_{12}^3} + (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4) \frac{\partial z_2^4}{\partial w_{12}^3} = \\
&= (\sigma(z_1^4) - a_1(x))\sigma'(z_1^4) w_{11}^4 \sigma'(z_1^3)\sigma(z_2^2) + \\
&+ (\sigma(z_2^4) - a_2(x))\sigma'(z_2^4) w_{21}^4 \sigma'(z_1^3)\sigma(z_2^2)
\end{aligned} \tag{4.22}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{13}^3} &= \frac{\partial}{\partial w_{13}^3}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{13}^3}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{13}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{13}^3}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{13}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)w_{11}^4\sigma'(z_1^3)\sigma(z_3^2)+ \\
&+ (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)w_{21}^4\sigma'(z_1^3)\sigma(z_3^2)
\end{aligned} \tag{4.23}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{21}^3} &= \frac{\partial}{\partial w_{21}^3}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{21}^3}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{21}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{21}^3}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{21}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)w_{12}^4\sigma'(z_2^3)\sigma(z_1^2)+ \\
&+ (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)w_{22}^4\sigma'(z_2^3)\sigma(z_1^2)
\end{aligned} \tag{4.24}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{22}^3} &= \frac{\partial}{\partial w_{22}^3}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{22}^3}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{22}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{22}^3}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{22}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)w_{12}^4\sigma'(z_2^3)\sigma(z_2^2)+ \\
&+ (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)w_{22}^4\sigma'(z_2^3)\sigma(z_2^2)
\end{aligned} \tag{4.25}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{23}^3} &= \frac{\partial}{\partial w_{23}^3}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{23}^3}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{23}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{23}^3}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{23}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)w_{12}^4\sigma'(z_2^3)\sigma(z_3^2)+ \\
&+ (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)w_{22}^4\sigma'(z_2^3)\sigma(z_3^2)
\end{aligned} \tag{4.26}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{31}^3} &= \frac{\partial}{\partial w_{31}^3}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{31}^3}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{31}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{31}^3}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{31}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)w_{13}^4\sigma'(z_3^3)\sigma(z_1^2)+ \\
&+ (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)w_{23}^4\sigma'(z_3^3)\sigma(z_1^2)
\end{aligned} \tag{4.27}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{32}^3} &= \frac{\partial}{\partial w_{32}^3}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{32}^3}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{32}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{32}^3}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{32}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)w_{13}^4\sigma'(z_3^3)\sigma(z_2^2)+ \\
&+ (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)w_{23}^4\sigma'(z_3^3)\sigma(z_2^2)
\end{aligned} \tag{4.28}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w_{33}^3} &= \frac{\partial}{\partial w_{33}^3}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) = \\
&= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{33}^3}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{33}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{33}^3}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{33}^3} = \\
&= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)w_{13}^4\sigma'(z_3^3)\sigma(z_3^2)+ \\
&+ (\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)w_{23}^4\sigma'(z_3^3)\sigma(z_3^2)
\end{aligned} \tag{4.29}$$

De maneira resumida, temos:

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w^3_{11}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{11}\sigma'(z^3_1)\sigma(z^2_1)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{21}\sigma'(z^3_1)\sigma(z^2_1)\\
\frac{\partial C(W,x)}{\partial w^3_{12}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{11}\sigma'(z^3_1)\sigma(z^2_2)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{21}\sigma'(z^3_1)\sigma(z^2_2)\\
\frac{\partial C(W,x)}{\partial w^3_{13}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{11}\sigma'(z^3_1)\sigma(z^2_3)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{21}\sigma'(z^3_1)\sigma(z^2_3)\\
\frac{\partial C(W,x)}{\partial w^3_{21}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{12}\sigma'(z^3_2)\sigma(z^2_1)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{22}\sigma'(z^3_2)\sigma(z^2_1)\\
\frac{\partial C(W,x)}{\partial w^3_{22}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{12}\sigma'(z^3_2)\sigma(z^2_2)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{22}\sigma'(z^3_2)\sigma(z^2_2)\\
\frac{\partial C(W,x)}{\partial w^3_{23}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{12}\sigma'(z^3_2)\sigma(z^2_3)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{22}\sigma'(z^3_2)\sigma(z^2_3)\\
\frac{\partial C(W,x)}{\partial w^3_{31}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{13}\sigma'(z^3_3)\sigma(z^2_1)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{23}\sigma'(z^3_3)\sigma(z^2_1)\\
\frac{\partial C(W,x)}{\partial w^3_{32}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{13}\sigma'(z^3_3)\sigma(z^2_2)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{23}\sigma'(z^3_3)\sigma(z^2_2)\\
\frac{\partial C(W,x)}{\partial w^3_{33}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)w^4_{13}\sigma'(z^3_3)\sigma(z^2_3)+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)w^4_{23}\sigma'(z^3_3)\sigma(z^2_3)
\end{aligned} \tag{4.30}$$

Da mesma forma que foi feito com W^4, iremos escrever uma matriz que chamaremos de $\frac{\partial C}{\partial W^3}$ contendo as derivadas parciais de C em relação aos pesos de W^3 da seguinte forma:

$$\begin{aligned}
\frac{\partial C}{\partial W^3} &= \left(\left(\begin{bmatrix} w^4_{11} & w^4_{21}\\ w^4_{12} & w^4_{22}\\ w^4_{13} & w^4_{23}\end{bmatrix}\left(\begin{bmatrix}\sigma(z^4_1)-a_1(x)\\ \sigma(z^4_2)-a_2(x)\end{bmatrix}\odot\begin{bmatrix}\sigma'(z^4_1)\\ \sigma'(z^4_2)\end{bmatrix}\right)\right)\odot\begin{bmatrix}\sigma'(z^3_1)\\ \sigma'(z^3_2)\\ \sigma'(z^3_3)\end{bmatrix}\right)\begin{bmatrix}\sigma(z^2_1)\\ \sigma(z^2_2)\\ \sigma(z^2_3)\end{bmatrix}^T =\\
&= \left(\left(\left(W^4\right)^T\left(\frac{\partial C}{\partial A^4}\odot\sigma'(Z^4)\right)\right)\odot\sigma'(Z^3)\right)\left(A^2\right)^T.
\end{aligned} \tag{4.31}$$

Finalmente, iremos agora, calcular as derivadas com relação aos pesos de W^2:

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w^2_{11}} &= \frac{\partial}{\partial w^2_{11}}\left(\frac{1}{2}\left((\sigma(z^4_1)-a_1(x))^2+(\sigma(z^4_2)-a_2(x))^2\right)\right)=\\
&= (\sigma(z^4_1)-a_1(x))\frac{\partial\sigma(z^4_1)}{\partial w^2_{11}}+(\sigma(z^4_2)-a_2(x))\frac{\partial\sigma(z^4_2)}{\partial w^2_{11}}=\\
&= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\frac{\partial z^4_1}{\partial w^2_{11}}+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\frac{\partial z^4_2}{\partial w^2_{11}}=\\
&= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\left(w^4_{11}\sigma'(z^3_1)w^3_{11}\sigma'(z^2_1)x_1+w^4_{12}\sigma'(z^3_2)w^3_{21}\sigma'(z^2_1)x_1+w^4_{13}\sigma'(z^3_3)w^3_{31}\sigma'(z^2_1)x_1\right)\\
&+ (\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\left(w^4_{21}\sigma'(z^3_1)w^3_{11}\sigma'(z^2_1)x_1+w^4_{22}\sigma'(z^3_2)w^3_{21}\sigma'(z^2_1)x_1+w^4_{23}\sigma'(z^3_3)w^3_{31}\sigma'(z^2_1)x_1\right)
\end{aligned} \tag{4.32}$$

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w^2_{12}} &= \frac{\partial}{\partial w^2_{12}}\left(\frac{1}{2}\left((\sigma(z^4_1)-a_1(x))^2+(\sigma(z^4_2)-a_2(x))^2\right)\right)=\\
&= (\sigma(z^4_1)-a_1(x))\frac{\partial\sigma(z^4_1)}{\partial w^2_{12}}+(\sigma(z^4_2)-a_2(x))\frac{\partial\sigma(z^4_2)}{\partial w^2_{12}}=\\
&= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\frac{\partial z^4_1}{\partial w^2_{12}}+(\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\frac{\partial z^4_2}{\partial w^2_{12}}=\\
&= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\left(w^4_{11}\sigma'(z^3_1)w^3_{11}\sigma'(z^2_1)x_2+w^4_{12}\sigma'(z^3_2)w^3_{21}\sigma'(z^2_1)x_2+w^4_{13}\sigma'(z^3_3)w^3_{31}\sigma'(z^2_1)x_2\right)\\
&+ (\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\left(w^4_{21}\sigma'(z^3_1)w^3_{11}\sigma'(z^2_1)x_2+w^4_{22}\sigma'(z^3_2)w^3_{21}\sigma'(z^2_1)x_2+w^4_{23}\sigma'(z^3_3)w^3_{31}\sigma'(z^2_1)x_2\right)
\end{aligned} \tag{4.33}$$

$$\frac{\partial C(W,x)}{\partial w_{21}^2} = \frac{\partial}{\partial w_{21}^2}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) =$$
$$= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{21}^2}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{21}^2} =$$
$$= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{21}^2}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{21}^2} =$$
$$= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\left(w_{11}^4\sigma'(z_1^3)w_{12}^3\sigma'(z_2^2)x_1+w_{12}^4\sigma'(z_2^3)w_{22}^3\sigma'(z_2^2)x_1+w_{13}^4\sigma'(z_3^3)w_{32}^3\sigma'(z_2^2)x_1\right)$$
$$+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\left(w_{21}^4\sigma'(z_1^3)w_{12}^3\sigma'(z_2^2)x_1+w_{22}^4\sigma'(z_2^3)w_{22}^3\sigma'(z_2^2)x_1+w_{23}^4\sigma'(z_3^3)w_{32}^3\sigma'(z_2^2)x_1\right) \tag{4.34}$$

$$\frac{\partial C(W,x)}{\partial w_{22}^2} = \frac{\partial}{\partial w_{22}^2}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) =$$
$$= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{22}^2}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{22}^2} =$$
$$= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{22}^2}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{22}^2} =$$
$$= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\left(w_{11}^4\sigma'(z_1^3)w_{12}^3\sigma'(z_2^2)x_2+w_{12}^4\sigma'(z_2^3)w_{22}^3\sigma'(z_2^2)x_2+w_{13}^4\sigma'(z_3^3)w_{32}^3\sigma'(z_2^2)x_2\right)$$
$$+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\left(w_{21}^4\sigma'(z_1^3)w_{12}^3\sigma'(z_2^2)x_2+w_{22}^4\sigma'(z_2^3)w_{22}^3\sigma'(z_2^2)x_2+w_{23}^4\sigma'(z_3^3)w_{32}^3\sigma'(z_2^2)x_2\right) \tag{4.35}$$

$$\frac{\partial C(W,x)}{\partial w_{31}^2} = \frac{\partial}{\partial w_{31}^2}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) =$$
$$= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{31}^2}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{31}^2} =$$
$$= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{31}^2}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{31}^2} =$$
$$= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\left(w_{11}^4\sigma'(z_1^3)w_{13}^3\sigma'(z_3^2)x_1+w_{12}^4\sigma'(z_2^3)w_{23}^3\sigma'(z_3^2)x_1+w_{13}^4\sigma'(z_3^3)w_{33}^3\sigma'(z_3^2)x_1\right)$$
$$+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\left(w_{21}^4\sigma'(z_1^3)w_{13}^3\sigma'(z_3^2)x_1+w_{22}^4\sigma'(z_2^3)w_{23}^3\sigma'(z_3^2)x_1+w_{23}^4\sigma'(z_3^3)w_{33}^3\sigma'(z_3^2)x_1\right) \tag{4.36}$$

$$\frac{\partial C(W,x)}{\partial w_{32}^2} = \frac{\partial}{\partial w_{32}^2}\left(\frac{1}{2}\left((\sigma(z_1^4)-a_1(x))^2+(\sigma(z_2^4)-a_2(x))^2\right)\right) =$$
$$= (\sigma(z_1^4)-a_1(x))\frac{\partial\sigma(z_1^4)}{\partial w_{32}^2}+(\sigma(z_2^4)-a_2(x))\frac{\partial\sigma(z_2^4)}{\partial w_{32}^2} =$$
$$= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\frac{\partial z_1^4}{\partial w_{32}^2}+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\frac{\partial z_2^4}{\partial w_{32}^2} =$$
$$= (\sigma(z_1^4)-a_1(x))\sigma'(z_1^4)\left(w_{11}^4\sigma'(z_1^3)w_{13}^3\sigma'(z_3^2)x_2+w_{12}^4\sigma'(z_2^3)w_{23}^3\sigma'(z_3^2)x_2+w_{13}^4\sigma'(z_3^3)w_{33}^3\sigma'(z_3^2)x_2\right)$$
$$+(\sigma(z_2^4)-a_2(x))\sigma'(z_2^4)\left(w_{21}^4\sigma'(z_1^3)w_{13}^3\sigma'(z_3^2)x_2+w_{22}^4\sigma'(z_2^3)w_{23}^3\sigma'(z_3^2)x_2+w_{23}^4\sigma'(z_3^3)w_{33}^3\sigma'(z_3^2)x_2\right) \tag{4.37}$$

De maneira resumida, temos:

$$\begin{aligned}
\frac{\partial C(W,x)}{\partial w^2_{11}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\left(w^4_{11}\sigma'(z^3_1)w^3_{11}\sigma'(z^2_1)x_1 + w^4_{12}\sigma'(z^3_2)w^3_{21}\sigma'(z^2_1)x_1 + w^4_{13}\sigma'(z^3_3)w^3_{31}\sigma'(z^2_1)x_1\right)\\
&+ (\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\left(w^4_{21}\sigma'(z^3_1)w^3_{11}\sigma'(z^2_1)x_1 + w^4_{22}\sigma'(z^3_2)w^3_{21}\sigma'(z^2_1)x_1 + w^4_{23}\sigma'(z^3_3)w^3_{31}\sigma'(z^2_1)x_1\right)\\
\frac{\partial C(W,x)}{\partial w^2_{12}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\left(w^4_{11}\sigma'(z^3_1)w^3_{11}\sigma'(z^2_1)x_2 + w^4_{12}\sigma'(z^3_2)w^3_{21}\sigma'(z^2_1)x_2 + w^4_{13}\sigma'(z^3_3)w^3_{31}\sigma'(z^2_1)x_2\right)\\
&+ (\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\left(w^4_{21}\sigma'(z^3_1)w^3_{11}\sigma'(z^2_1)x_2 + w^4_{22}\sigma'(z^3_2)w^3_{21}\sigma'(z^2_1)x_2 + w^4_{23}\sigma'(z^3_3)w^3_{31}\sigma'(z^2_1)x_2\right)\\
\frac{\partial C(W,x)}{\partial w^2_{21}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\left(w^4_{11}\sigma'(z^3_1)w^3_{12}\sigma'(z^2_2)x_1 + w^4_{12}\sigma'(z^3_2)w^3_{22}\sigma'(z^2_2)x_1 + w^4_{13}\sigma'(z^3_3)w^3_{32}\sigma'(z^2_2)x_1\right)\\
&+ (\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\left(w^4_{21}\sigma'(z^3_1)w^3_{12}\sigma'(z^2_2)x_1 + w^4_{22}\sigma'(z^3_2)w^3_{22}\sigma'(z^2_2)x_1 + w^4_{23}\sigma'(z^3_3)w^3_{32}\sigma'(z^2_2)x_1\right)\\
\frac{\partial C(W,x)}{\partial w^2_{22}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\left(w^4_{11}\sigma'(z^3_1)w^3_{12}\sigma'(z^2_2)x_2 + w^4_{12}\sigma'(z^3_2)w^3_{22}\sigma'(z^2_2)x_2 + w^4_{13}\sigma'(z^3_3)w^3_{32}\sigma'(z^2_2)x_2\right)\\
&+ (\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\left(w^4_{21}\sigma'(z^3_1)w^3_{12}\sigma'(z^2_2)x_2 + w^4_{22}\sigma'(z^3_2)w^3_{22}\sigma'(z^2_2)x_2 + w^4_{23}\sigma'(z^3_3)w^3_{32}\sigma'(z^2_2)x_2\right)\\
\frac{\partial C(W,x)}{\partial w^2_{31}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\left(w^4_{11}\sigma'(z^3_1)w^3_{13}\sigma'(z^2_3)x_1 + w^4_{12}\sigma'(z^3_2)w^3_{23}\sigma'(z^2_3)x_1 + w^4_{13}\sigma'(z^3_3)w^3_{33}\sigma'(z^2_3)x_1\right)\\
&+ (\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\left(w^4_{21}\sigma'(z^3_1)w^3_{13}\sigma'(z^2_3)x_1 + w^4_{22}\sigma'(z^3_2)w^3_{23}\sigma'(z^2_3)x_1 + w^4_{23}\sigma'(z^3_3)w^3_{33}\sigma'(z^2_3)x_1\right)\\
\frac{\partial C(W,x)}{\partial w^2_{32}} &= (\sigma(z^4_1)-a_1(x))\sigma'(z^4_1)\left(w^4_{11}\sigma'(z^3_1)w^3_{13}\sigma'(z^2_3)x_2 + w^4_{12}\sigma'(z^3_2)w^3_{23}\sigma'(z^2_3)x_2 + w^4_{13}\sigma'(z^3_3)w^3_{33}\sigma'(z^2_3)x_2\right)\\
&+ (\sigma(z^4_2)-a_2(x))\sigma'(z^4_2)\left(w^4_{21}\sigma'(z^3_1)w^3_{13}\sigma'(z^2_3)x_2 + w^4_{22}\sigma'(z^3_2)w^3_{23}\sigma'(z^2_3)x_2 + w^4_{23}\sigma'(z^3_3)w^3_{33}\sigma'(z^2_3)x_2\right)
\end{aligned} \tag{4.38}$$

Dessa forma, analogamente ao que foi feito com relação à W^4 e W^3, é possível escrever a matriz $\frac{\partial C}{\partial W^2}$ contendo as derivadas parciais de C em relação aos pesos de W^2 da seguinte forma:

$$\frac{\partial C}{\partial W^2} = \left(\left[\left(W^3\right)^T\left(\left[\left(W^4\right)^T\left(\frac{\partial C}{\partial A^4}\odot\sigma'(Z^4)\right)\right]\odot\sigma'(Z^3)\right)\right]\odot\sigma'(Z^2)\right)\left(A^1\right)^T. \tag{4.39}$$

onde $A^1 = x$.

Conforme os cálculos feitos anteriormente, encontramos as derivadas parciais da função custo C contidas nas matrizes que chamamos de $\frac{\partial C}{\partial W^L}$, para $L = 2, 3, 4$. Observando as Equações (4.19), (4.31), (4.39) definiremos um termo que chamaremos de δ^L em cada equação:

$$\begin{aligned}
\frac{\partial C}{\partial W^4} &= \underbrace{\left(\frac{\partial C}{\partial A^4}\odot\sigma'(Z^4)\right)}_{\delta^4}\left(A^3\right)^T = \delta^4\left(A^3\right)^T\\
\frac{\partial C}{\partial W^3} &= \underbrace{\left(\left(\left(W^4\right)^T\left(\frac{\partial C}{\partial A^4}\odot\sigma'(Z^4)\right)\right)\odot\sigma'(Z^3)\right)}_{\delta^3}\left(A^2\right)^T =\\
&= \underbrace{\left(\left(\left(W^4\right)^T\delta^4\right)\odot\sigma'(Z^3)\right)}_{\delta^3}\left(A^2\right)^T = \delta^3\left(A^2\right)^T\\
\frac{\partial C}{\partial W^2} &= \underbrace{\left(\left(\left(W^3\right)^T\left(\left(\left(W^4\right)^T\left(\frac{\partial C}{\partial A^4}\odot\sigma'(Z^4)\right)\right)\odot\sigma'(Z^3)\right)\right)\odot\sigma'(Z^2)\right)}_{\delta^2}\left(A^1\right)^T =\\
&= \underbrace{\left(\left(\left(W^3\right)^T\delta^3\right)\odot\sigma'(Z^2)\right)}_{\delta^2}\left(A^1\right)^T = \delta^2\left(A^1\right)^T
\end{aligned} \tag{4.40}$$

De modo geral, uma rede com N camadas, inicia-se o cálculo da derivada com

$$\frac{\partial C}{\partial W^N} = \underbrace{\left(\frac{\partial C}{\partial A^N} \odot \sigma'(Z^N)\right)}_{\delta^N} \cdot \left(A^{N-1}\right)^T \tag{4.41}$$

e para as outras camadas $L = N-1, N-2, \ldots, 2$, a derivada é dada por:

$$\frac{\partial C}{\partial W^L} = \underbrace{\left(\left(\left(W^{L+1}\right)^T \cdot \delta^{L+1}\right) \odot \sigma'(Z^L)\right)}_{\delta^L} \left(A^{L-1}\right)^T. \tag{4.42}$$

4.3 CÁLCULO CONSIDERANDO O CONJUNTO DE DADOS COMPLETO

Iremos agora, considerar o nosso conjunto de dados completo e não apenas uma amostra. Seja então X a matriz contendo nosso conjunto de dados com n amostras distribuídas em suas colunas da seguinte forma:

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(n)} \end{bmatrix} \tag{4.43}$$

onde cada coluna $x^{(i)} = (x_1^{(i)}, x_2^{(i)})$ é a amostra i do conjunto de treinamento. Usando a matriz X, os vetores e matrizes definidos anteriormente agora ficam:

$$Z^2 = W^2 X = \begin{bmatrix} z_1^{2(1)} & z_1^{2(2)} & \ldots & z_1^{2(n)} \\ z_2^{2(1)} & z_2^{2(2)} & \ldots & z_2^{2(n)} \\ z_3^{2(1)} & z_3^{2(2)} & \ldots & z_3^{2(n)} \end{bmatrix}, \tag{4.44}$$

$$A^2 = \sigma(Z^2) = \begin{bmatrix} \sigma(z_1^{2(1)}) & \sigma(z_1^{2(2)}) & \ldots & \sigma(z_1^{2(n)}) \\ \sigma(z_2^{2(1)}) & \sigma(z_2^{2(2)}) & \ldots & \sigma(z_2^{2(n)}) \\ \sigma(z_3^{2(1)}) & \sigma(z_3^{2(2)}) & \ldots & \sigma(z_3^{2(n)}) \end{bmatrix}, \tag{4.45}$$

$$Z^3 = W^3 A^2 = \begin{bmatrix} z_1^{3(1)} & z_1^{3(2)} & \ldots & z_1^{3(n)} \\ z_2^{3(1)} & z_2^{3(2)} & \ldots & z_2^{3(n)} \\ z_3^{3(1)} & z_3^{3(2)} & \ldots & z_3^{3(n)} \end{bmatrix}, \tag{4.46}$$

$$A^3 = \sigma(Z^3) = \begin{bmatrix} \sigma(z_1^{3(1)}) & \sigma(z_1^{3(2)}) & \ldots & \sigma(z_1^{3(n)}) \\ \sigma(z_2^{3(1)}) & \sigma(z_2^{3(2)}) & \ldots & \sigma(z_2^{3(n)}) \\ \sigma(z_3^{3(1)}) & \sigma(z_3^{3(2)}) & \ldots & \sigma(z_3^{3(n)}) \end{bmatrix}, \tag{4.47}$$

$$Z^4 = W^4 A^3 = \begin{bmatrix} z_1^{4(1)} & z_1^{4(1)} & \ldots & z_1^{4(n)} \\ z_2^{4(1)} & z_2^{4(1)} & \ldots & z_2^{4(n)} \end{bmatrix}, \tag{4.48}$$

$$A^4 = \sigma(Z^4) = \begin{bmatrix} \sigma(z_1^{4(1)}) & \sigma(z_1^{4(2)}) & \ldots & \sigma(z_1^{4(n)}) \\ \sigma(z_2^{4(1)}) & \sigma(z_2^{4(2)}) & \ldots & \sigma(z_2^{4(n)}) \end{bmatrix}. \tag{4.49}$$

Associado ao conjunto de dados X, temos a matriz $a(X)$ que contém as saídas esperadas para as amostras em X

$$a(X) = \begin{bmatrix} a_1(x^{(1)}) & a_1(x^{(2)}) & \dots & a_1(x^{(n)}) \\ a_2(x^{(1)}) & a_2(x^{(2)}) & \dots & a_2(x^{(n)}) \end{bmatrix}, \tag{4.50}$$

e com isso, definimos nossa função custo C

$$C(W, X) = \frac{1}{2n}||A^4(W, X) - a(X)||^2. \tag{4.51}$$

Algoritmo 1: Backpropagation

Dados: Conjunto de dados (X, y)
Resultado: $\nabla_w C$

$\nabla_w C \leftarrow$ lista de matrizes de zeros nas dimensões das matrizes de pesos W;
ativacoes $\leftarrow$ lista vazia;
ativacao $\leftarrow X$;
ativacoes $\leftarrow$ ativacoes $\cup$ ativacao;
$zs \leftarrow$ lista vazia;
para *cada W nas matrizes de pesos* **faça**
    $z \leftarrow W *$ ativacao
    $zs \leftarrow zs \cup z$;
    ativacao $\leftarrow f(z)$; `/* f é a função de ativação. */`
    ativacoes $\leftarrow$ ativacoes $\cup$ ativacao;
delta $\leftarrow$ (ativacoes$[-1] - y) \odot f'(zs[-1])$; `/* [p] representa a posição p da lista. -1 é a ultima posição, -2 a penúltima e assim por diante. */`
$\nabla_w C[-1] \leftarrow$ delta $\cdot$ (ativacoes$[-2])^T$;
para *l de 2 até numero de camadas* **faça**
    $z \leftarrow zs[-l]$;
    $sp \leftarrow f'(z)$; `/* f' é a derivada da função de ativação */`
    delta $\leftarrow ((W[-l+1])^T \cdot$ delta$) \odot sp$;
    $\nabla_w C[-l] \leftarrow$ delta $*$ (ativacoes$[-l-1])^T$;

Definindo $\frac{\partial C}{\partial A^4}$ da seguinte forma:

$$\frac{\partial C}{\partial A^4} = \begin{bmatrix} (\sigma(z_1^{4(1)}) - a_1(x^{(1)})) & (\sigma(z_1^{4(2)}) - a_1(x^{(2)})) & \dots & (\sigma(z_1^{4(N)}) - a_1(x^{(n)})) \\ (\sigma(z_2^{4(1)}) - a_2(x^{(1)})) & (\sigma(z_2^{4(1)}) - a_2(x^{(2)})) & \dots & (\sigma(z_2^{4(N)}) - a_2(x^{(n)})) \end{bmatrix} \tag{4.52}$$

e realizando as contas como anteriormente, chegamos na equação para a derivada de C com relação aos pesos de W^4:

$$\frac{\partial C}{\partial W^4} = \underbrace{\left(\frac{\partial C}{\partial A^4} \odot \sigma'(Z^4)\right)}_{\delta^4} \cdot \left(A^3\right)^T \tag{4.53}$$

onde

$$\delta^4 = \begin{bmatrix} (\sigma(z_1^{4(1)}) - a_1(x^{(1)}))\sigma'(z_1^{4(1)}) & (\sigma(z_1^{4(2)}) - a_1(x^{(2)}))\sigma'(z_1^{4(2)}) & \dots & (\sigma(z_1^{4(n)}) - a_1(x^{(n)}))\sigma'(z_1^{4(n)}) \\ (\sigma(z_2^{4(1)}) - a_2(x^{(1)}))\sigma'(z_2^{4(1)}) & (\sigma(z_2^{4(2)}) - a_2(x^{(2)}))\sigma'(z_2^{4(2)}) & \dots & (\sigma(z_2^{4(n)}) - a_2(x^{(n)}))\sigma'(z_2^{4(n)}) \end{bmatrix}. \tag{4.54}$$

Nessa derivada, observa-se que os termos $\frac{\partial C}{\partial A^4}$ e $\sigma'(Z^4)$ têm dimensão $2 \times n$ e o termo $(A^3)^T$ tem dimensão $n \times 3$ e portanto, em termos de dimensão, os cálculos fazem sentido. Observe que cada coluna da matriz δ^4 é resultado de um produto de Hadamard entre as colunas da matriz $\frac{\partial C}{\partial A^4}$ e da matriz $\sigma'(Z^4)$, que corresponde a contribuição de cada amostra do conjunto de dados. Por outro lado, os elementos de cada linha i de δ^4 correspondem à contribuição, no produto de Hadamard, do elemento de índice i das amostras. Dessa forma, os elementos da matriz $\frac{\partial C}{\partial W^4}$ ficam:

$$\begin{aligned}
\frac{\partial C(W,X)}{\partial w_{11}^4} &= \sum_{j=1}^{n}(\sigma(z_1^{4(j)}) - a_1(x^{(j)}))\sigma'(z_1^{4(j)})\sigma(z_1^{3(j)}) = \sum_{j=1}^{n}\frac{\partial C(W,x^{(j)})}{\partial w_{11}^4},\\
\frac{\partial C(W,X)}{\partial w_{12}^4} &= \sum_{j=1}^{n}(\sigma(z_1^{4(j)}) - a_1(x^{(j)}))\sigma'(z_1^{4(j)})\sigma(z_2^{3(j)}) = \sum_{j=1}^{n}\frac{\partial C(W,x^{(j)})}{\partial w_{12}^4},\\
\frac{\partial C(W,X)}{\partial w_{13}^4} &= \sum_{j=1}^{n}(\sigma(z_1^{4(j)}) - a_1(x^{(j)}))\sigma'(z_1^{4(j)})\sigma(z_3^{3(j)}) = \sum_{j=1}^{n}\frac{\partial C(W,x^{(j)})}{\partial w_{13}^4},\\
\frac{\partial C(W,X)}{\partial w_{21}^4} &= \sum_{j=1}^{n}(\sigma(z_2^{4(j)}) - a_2(x^{(j)}))\sigma'(z_2^{4(j)})\sigma(z_1^{3(j)}) = \sum_{j=1}^{n}\frac{\partial C(W,x^{(j)})}{\partial w_{21}^4},\\
\frac{\partial C(W,X)}{\partial w_{22}^4} &= \sum_{j=1}^{n}(\sigma(z_2^{4(j)}) - a_2(x^{(j)}))\sigma'(z_2^{4(j)})\sigma(z_2^{3(j)}) = \sum_{j=1}^{n}\frac{\partial C(W,x^{(j)})}{\partial w_{22}^4},\\
\frac{\partial C(W,X)}{\partial w_{23}^4} &= \sum_{j=1}^{n}(\sigma(z_2^{4(j)}) - a_2(x^{(j)}))\sigma'(z_2^{4(j)})\sigma(z_3^{3(j)}) = \sum_{j=1}^{n}\frac{\partial C(W,x^{(j)})}{\partial w_{23}^4}.
\end{aligned} \tag{4.55}$$

Com isso, obtivemos a derivada de maneira análoga como anteriormente feito para uma única amostra, obtendo como resultado final a soma das derivadas considerando cada amostra do conjunto de dados. Porém, os cálculos foram feitos de forma matricial, sem a necessidade de realizar o somatório explicitamente, o que seria mais custoso do ponto de vista computacional.

Os demais cálculos são feitos de forma análoga de tal forma que a matriz resultante, $\frac{\partial C}{\partial W^L}$ na camada L, independentemente de utilizarmos uma única amostra ou o conjunto de dados completos, terá a mesma dimensão, digamos, $p \times q$, pois, δ^L tem dimensão $p \times n$ e $(A^L)^T$ tem dimensão $n \times q$ onde, como já visto, n é o número de amostras.

O Algoritmo 1 apresenta um pseudocódigo do *backpropagation* considerando-se o erro quadrático médio como função custo.

4.4 ENTROPIA CRUZADA E SOFTMAX

No caso em que a função de entropia cruzada é escolhida como função de custo, usa-se a função de ativação *softmax* $s : \mathbb{R}^m \longrightarrow [0,1]^m$ definida por:

$$s(z) = (s_1(z), s_2(z), \ldots, s_m(z)) \text{ onde } s_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}} \quad \text{para } i = 1, 2, \ldots, m \tag{4.56}$$

com $\sum_{i=1}^{m} s_i(z) = 1$ na ultima camada para que a saída seja uma função de probabilidade.

No nosso exemplo, $m = 2$ pois temos apenas 2 neurônios na camada de saída da rede. Assim, a saída da rede, é dada por

$$A^4 = s(Z^4) = \begin{bmatrix} s_1(Z^4) \\ s_2(Z^4) \end{bmatrix}. \tag{4.57}$$

e a função custo por

$$C(W, x) = -(a_1(x) \cdot \ln s_1(Z^4) + a_2(x) \cdot \ln s_2(Z^4)) \tag{4.58}$$

Para uma rede qualquer, com N camadas, as derivadas da função custo C em relação aos pesos de W^N, relativos a ultima camada, podem ser escritas como

$$\frac{\partial C}{\partial W^N} = \frac{\partial C}{\partial Z^N}\frac{\partial Z^N}{\partial W^N} = \frac{\partial C}{\partial Z^N}(A^N)^T \tag{4.59}$$

da mesma forma que tínhamos anteriormente. Porém, agora, o termo $\frac{\partial C}{\partial Z^N}$, que representa δ^N, será calculado de forma diferente. Iniciaremos calculando a derivada de $s_i(Z^4)$ e, sem perda de generalidade, denotaremos Z^N por z e z_i^N por z_i, para qualquer i.

Sejam

$$f_i(z) = e^{z_i} \quad \text{e} \quad g(z) = \sum_{k=1}^{m} e^{z_k}. \tag{4.60}$$

Daí, as derivadas de f e g com respeito a z_j são dadas por

$$\frac{\partial f_i(z)}{\partial z_j} = \frac{\partial e^{z_i}}{\partial z_j} = e^{z_i}\frac{\partial z_i}{\partial z_j} = \begin{cases} e^{z_i}, \text{ se } i = j \\ 0, \text{ se } i \neq j \end{cases}, \tag{4.61}$$

$$\frac{\partial g(z)}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\sum_{k=1}^{m} e^{z_k}\right) = \frac{\partial}{\partial z_j}\left(\sum_{k=1,k\neq j}^{m} e^{z_k} + e^{z_j}\right) = e^{z_j}. \tag{4.62}$$

No caso em que $i = j$ obtemos:

$$\begin{aligned} \frac{\partial s_i(z)}{\partial z_j} &= \frac{e^{z_i}\sum_{k=1}^{m} e^{z_k} - e^{z_i}e^{z_i}}{\left(\sum_{k=1}^{m} e^{z_k}\right)^2} = \frac{e^{z_i}\left(\sum_{k=1}^{m} e^{z_k} - e^{z_i}\right)}{\left(\sum_{k=1}^{m} e^{z_k}\right)^2} = \\ &= \frac{e^{z_i}}{\sum_{k=1}^{m} e^{z_k}}\frac{\sum_{k=1}^{m} e^{z_k} - e^{z_i}}{\sum_{k=1}^{m} e^{z_k}} = s_i(z)(1 - s_i(z)). \end{aligned} \tag{4.63}$$

Se $i \neq j$ temos:

$$\frac{\partial s_i(z)}{\partial z_j} = \frac{0 \cdot \sum_{k=1}^{m} e^{z_k} - e^{z_i}e^{z_j}}{\left(\sum_{k=1}^{m} e^{z_k}\right)^2} = -\frac{e^{z_i}}{\sum_{k=1}^{m} e^{z_k}}\frac{e^{z_j}}{\sum_{k=1}^{m} e^{z_k}} = -s_i(z)s_j(z) \tag{4.64}$$

Iremos agora, utilizar a derivada de $s_i(z)$ para calcular $\frac{\partial C}{\partial Z^N}$. Para isso, precisamos calcular a derivada de C com relação a z_i:

$$\begin{aligned}
\frac{\partial C}{\partial z_i} &= \frac{\partial}{\partial z_i}\left(-\sum_{k=1}^{m} a_k(x)\cdot \log s_k(z)\right) = -\sum_{k=1}^{m} a_k(x)\cdot \frac{\partial \log s_k(z)}{\partial z_i} = \\
&= -\sum_{k=1}^{m} a_k(x)\cdot \frac{1}{s_k(z)}\frac{\partial s_k(z)}{\partial z_i} = -\sum_{k=1}^{m} \frac{a_k(x)}{s_k(z)}\frac{\partial s_k(z)}{\partial z_i} = \\
&= -\frac{a_i(x)}{s_i(z)}\frac{\partial s_i(z)}{\partial z_i} - \sum_{k=1,k\neq i}^{m} \frac{a_k(x)}{s_k(z)}\frac{\partial s_k(z)}{\partial z_i} = \\
&= -\frac{a_i(x)}{\cancel{s_i(z)}}\cancel{s_i(z)}(1-s_i(z)) - \sum_{k=1,k\neq i}^{m} \frac{a_k(x)}{\cancel{s_k(z)}}\left(-\cancel{s_k(z)}s_i(z)\right) = \\
&= -a_i(x)(1-s_i(z)) + \sum_{k=1,k\neq i}^{m} a_k(x)s_i(z) = a_i(x)s_i(z) + \sum_{k=1,k\neq i}^{m} a_k(x)s_i(z) - a_i(x) = \\
&= \sum_{k=1}^{m} a_k(x)s_i(z) - a_i(x) = s_i(z)\underbrace{\sum_{k=1}^{m} a_k(x)}_{=1} - a_i(x) = s_i(z) - a_i(x).
\end{aligned} \tag{4.65}$$

Portanto,

$$\frac{\partial C}{\partial Z^N} = s(Z^N) - a(x) \tag{4.66}$$

onde $a(x) = (a_1(x), a_2(x), \ldots, a_m(x))$. Com isso, temos que as derivadas parciais de C com relação aos pesos de W^N serão dadas na matriz $\frac{\partial C}{\partial W^N}$ obtida da seguinte forma:

$$\frac{\partial C}{\partial W^N} = \underbrace{\left(s(Z^N) - a(x)\right)}_{\delta^N}\left(A^N\right)^T. \tag{4.67}$$

A única mudança com a utilização da função de ativação *softmax* e a entropia cruzada como função custo é no cálculo da derivada na ultima camada da rede. Os demais cálculos do *backpropagation* ocorrerão da mesma forma como apresentado anteriormente.

4.5 EXEMPLO

Consideremos uma rede neural com 3 neurônios na camada de entrada, 1 camada oculta com 3 neurônios e 1 neurônio na camada de saída, conforme representado na Figura 5. Nessa rede já estão adicionados os neurônios com valores constantes iguais a 1 para a remoção do *bias*.

Seja o conjunto inicial de pesos

$$\begin{aligned}
(W^2)^{(1)} &= \begin{bmatrix} 1.62 & -0.61 & -0.53 \\ -1.07 & 0.87 & -2.30 \\ 1.74 & -0.76 & 0.32 \end{bmatrix} \\
(W^3)^{(1)} &= \begin{bmatrix} -0.25 & 1.46 & -2.06 \end{bmatrix}.
\end{aligned} \tag{4.68}$$

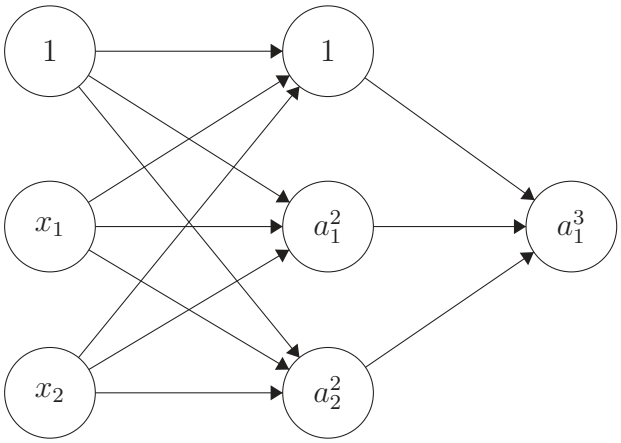


Figura 5 – Rede Neural para exemplificação do *backpropagation*.

Para simplificação de notação escreveremos $(W^2)^{(1)} = W^2$ e $(W^3)^{(1)} = W^3$ e, de forma geral, quando não houver perigo de confusão, na iteração k, $(W^i)^{(k)} = W^i$.

Utilizaremos nessa rede a função de ativação sigmoide $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$, definida por

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{4.69}$$

Consideremos o seguinte conjunto de dados

$$\{(x^{(1)}, a(x^{(1)})), (x^{(2)}, a(x^{(2)}))\} = \{((2, 1.5), 1), ((0, 1), 0)\}$$

ou seja, quando a entrada é $x_1 = 2$ e $x_2 = 1.5$ a saída esperada é $a_1^3 = 1$ e quando a entrada é $x_1 = 0$ e $x_2 = 1$ a saída esperada é $a_1^3 = 0$.

Iniciaremos com a primeira amostra mostrando os cálculos para Z^2 e A^2:

$$Z^{2(1)} = W^2 x^{(1)} = \begin{bmatrix} 1.62 & -0.61 & -0.53 \\ -1.07 & 0.87 & -2.30 \\ 1.74 & -0.76 & 0.32 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1.5 \end{bmatrix} = \begin{bmatrix} -0.39 \\ -2.78 \\ 0.70 \end{bmatrix} \tag{4.70}$$

$$A^{2(1)} = \sigma(Z^{2(1)}) = \sigma\left(\begin{bmatrix} -0.39 \\ -2.78 \\ 0.70 \end{bmatrix}\right) = \begin{bmatrix} \sigma(-0.39) \\ \sigma(-2.78) \\ \sigma(0.70) \end{bmatrix} = \begin{bmatrix} 0.40 \\ 0.06 \\ 0.67 \end{bmatrix} \tag{4.71}$$

Por conta do *bias*, consideramos só os dois últimos termos de $A^{2(1)}$, sendo o primeiro fixado igual a 1. Com isso

$$A^{2(1)} = \begin{bmatrix} 1 \\ 0.06 \\ 0.67 \end{bmatrix} = \begin{bmatrix} 1 \\ a_1^{2(1)} \\ a_2^{2(1)} \end{bmatrix}. \tag{4.72}$$

Agora, o cálculo de Z^3 e A^3:

$$Z^{3(1)} = W^3 A^{2(1)} = \begin{bmatrix} -0.25 & 1.46 & -2.06 \end{bmatrix} \begin{bmatrix} 1 \\ 0.06 \\ 0.67 \end{bmatrix} = -1.54 \tag{4.73}$$

$$A^{3(1)} = \sigma(Z^{3(1)}) = \sigma(-1.54) = 0.18 = y(x^{(1)}). \tag{4.74}$$

Para a segunda amostra $x^{(2)}$, temos:

$$Z^{2(2)} = W^2 x^{(2)} = \begin{bmatrix} 1.62 & -0.61 & -0.53 \\ -1.07 & 0.87 & -2.30 \\ 1.74 & -0.76 & 0.32 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.09 \\ -3.37 \\ 2.06 \end{bmatrix} \tag{4.75}$$

$$A^{2(2)} = \sigma(Z^{2(2)}) = \sigma\left(\begin{bmatrix} 1.09 \\ -3.37 \\ 2.06 \end{bmatrix}\right) = \begin{bmatrix} \sigma(1.09) \\ \sigma(-3.37) \\ \sigma(2.06) \end{bmatrix} = \begin{bmatrix} 0.75 \\ 0.03 \\ 0.89 \end{bmatrix} \tag{4.76}$$

Novamente, por conta do *bias*, de $A^{2(2)}$ aproveitamos só os dois últimos termos, sendo o primeiro igualado a 1 e com isso, obtemos:

$$A^{2(2)} = \begin{bmatrix} 1 \\ 0.03 \\ 0.89 \end{bmatrix} = \begin{bmatrix} 1 \\ a_1^{2(2)} \\ a_2^{2(2)} \end{bmatrix}. \tag{4.77}$$

Agora, o cálculo de Z^3 e A^3:

$$Z^{3(2)} = W^3 A^{2(2)} = \begin{bmatrix} -0.25 & 1.46 & -2.06 \end{bmatrix} \begin{bmatrix} 1 \\ 0.03 \\ 0.89 \end{bmatrix} = -2.04 \tag{4.78}$$

$$A^{3(2)} = \sigma(Z^{3(2)}) = \sigma(-2.04) = 0.12 = y(x^{(2)}). \tag{4.79}$$

Com a saída da rede neural calculada para cada amostra do conjunto de dados, considerando o erro quadrático médio, obtemos a função custo:

$$\begin{aligned} C(w,x) &= \frac{1}{2 \cdot 2}\left(\|y(x^{(1)}) - a(x^{(1)})\|^2 + \|y(x^{(2)}) - a(x^{(2)})\|^2\right) = \\ &= \frac{1}{4}\left(\|0.18 - 1\|^2 + \|0.12 - 0\|^2\right) = \frac{1}{4}\left(0.67 + 0.01\right) = 0.17 \end{aligned} \tag{4.80}$$

A seguir, calcularemos as derivadas de C com respeito aos pesos da rede considerando a amostra $x^{(1)}$. A derivada da função sigmoide, $\sigma' : \mathbb{R} \longrightarrow \mathbb{R}$ é dada por

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)). \tag{4.81}$$

Em primeiro lugar, calcularemos $\frac{\partial C}{\partial W^3}$ como na Equação (4.41):

$$\begin{aligned}
\frac{\partial C}{\partial W^3} &= \underbrace{\left(\frac{\partial C}{\partial A^3} \odot \sigma'(Z^{3(1)})\right)}_{\delta^3} \cdot \left(A^{2(1)}\right)^T = \underbrace{\left((y(x^{(1)}) - a(x^{(1)})) \odot \sigma'(Z^{3(1)})\right)}_{\delta^3} \cdot (A^{2(1)})^T \\
&= ((0.18 - 1) \odot \sigma'(-1.54)) \cdot \begin{bmatrix} 1 & 0.06 & 0.67 \end{bmatrix} = \\
&= (-0.82 \odot 0.15) \cdot \begin{bmatrix} 1 & 0.06 & 0.67 \end{bmatrix} = \\
&= \underbrace{-0.12}_{\delta^3} \cdot \begin{bmatrix} 1 & 0.06 & 0.67 \end{bmatrix} = \\
&= \begin{bmatrix} -0.12 & -0.01 & -0.08 \end{bmatrix}
\end{aligned} \tag{4.82}$$

Agora, o cálculo de $\frac{\partial C}{\partial W^2}$ utilizando δ^3 seguindo a Equação (4.42):

$$\begin{aligned}
\frac{\partial C}{\partial W^2} &= \underbrace{\left(\left((W^3)^T \cdot \delta^3\right) \odot \sigma'(Z^{2(1)})\right)}_{\delta^2} \left(A^{1(1)}\right)^T = \\
&= \underbrace{\left(\left(\left(\begin{bmatrix} -0.25 & 1.46 & -2.06 \end{bmatrix}\right)^T \cdot (-0.12)\right) \odot \begin{bmatrix} \sigma'(-0.39) \\ \sigma'(-2.78) \\ \sigma'(0.70) \end{bmatrix}\right)}_{\delta^2} \left(\begin{bmatrix} 1 \\ 2 \\ 1.5 \end{bmatrix}\right)^T = \\
&= \underbrace{\left(\begin{bmatrix} 0.03 \\ -0.18 \\ 0.25 \end{bmatrix} \odot \begin{bmatrix} 0.24 \\ 0.06 \\ 0.22 \end{bmatrix}\right)}_{\delta^2} \begin{bmatrix} 1.0 & 2.0 & 1.5 \end{bmatrix} = \\
&= \begin{bmatrix} 0.01 \\ -0.01 \\ 0.06 \end{bmatrix} \begin{bmatrix} 1.0 & 2.0 & 1.5 \end{bmatrix} = \begin{bmatrix} 0.01 & 0.02 & 0.02 \\ -0.01 & -0.02 & -0.02 \\ 0.06 & 0.12 & 0.09 \end{bmatrix}
\end{aligned} \tag{4.83}$$

Considerando agora a amostra $x^{(2)}$:

$$\begin{aligned}
\frac{\partial C}{\partial W^3} &= \underbrace{\left((y(x^{(2)}) - a(x^{(2)})) \odot \sigma'(Z^{3(2)})\right)}_{\delta^3} \cdot (A^{2(2)})^T \\
&= ((0.12 - 0) \odot \sigma'(-2.04)) \cdot \begin{bmatrix} 1 & 0.03 & 0.89 \end{bmatrix} = \\
&= (0.12 \odot 0.10) \cdot \begin{bmatrix} 1 & 0.03 & 0.89 \end{bmatrix} = \\
&= \underbrace{0.01}_{\delta^3} \cdot \begin{bmatrix} 1 & 0.03 & 0.89 \end{bmatrix} = \\
&= \begin{bmatrix} 0.01 & 0.0 & 0.01 \end{bmatrix}
\end{aligned} \tag{4.84}$$

$$
\begin{aligned}
\frac{\partial C}{\partial W^2} &= \underbrace{\left(\left(\left(W^3\right)^T \cdot \delta^3\right) \odot \sigma'(Z^{2(2)})\right)}_{\delta^2} \left(A^{1(2)}\right)^T = \\
&= \underbrace{\left(\left(\left(\begin{bmatrix} -0.25 & 1.46 & -2.06 \end{bmatrix}\right)^T \cdot 0.01\right) \odot \begin{bmatrix} \sigma'(1.09) \\ \sigma'(-3.37) \\ \sigma'(2.06) \end{bmatrix}\right)}_{\delta^2} \left(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right)^T = \\
&= \underbrace{\left(\begin{bmatrix} 0.0 \\ 0.01 \\ -0.02 \end{bmatrix} \odot \begin{bmatrix} 0.19 \\ 0.03 \\ 0.10 \end{bmatrix}\right)}_{\delta^2} \left(\begin{bmatrix} 1 \\ 2 \\ 1.5 \end{bmatrix}\right)^T = \\
&= \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}.
\end{aligned} \tag{4.85}
$$

A derivada final será dada pela soma das derivadas encontradas, multiplicada pelo fator $\frac{1}{2n}$, com $n = 2$ no nosso exemplo:

$$
\frac{\partial C}{\partial W^3} = \frac{1}{4}\left(\begin{bmatrix} -0.12 & -0.01 & -0.08 \end{bmatrix} + \begin{bmatrix} 0.01 & 0.0 & 0.01 \end{bmatrix}\right) = \begin{bmatrix} -0.03 & -0.00 & -0.02 \end{bmatrix} \tag{4.86}
$$

$$
\frac{\partial C}{\partial W^2} = \frac{1}{4}\left(\begin{bmatrix} 0.01 & 0.02 & 0.02 \\ -0.01 & -0.02 & -0.02 \\ 0.06 & 0.12 & 0.09 \end{bmatrix} + \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}\right) = \begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \\ 0.02 & 0.03 & 0.02 \end{bmatrix}. \tag{4.87}
$$

Adotando uma taxa de aprendizagem $\eta = 0.1$, o primeiro passo do SGD fica:

$$
\begin{aligned}
(W^3)^{(2)} &= (W^3)^{(1)} - \eta \frac{\partial C}{\partial W^3} = \\
&= \begin{bmatrix} -0.25 & 1.46 & -2.06 \end{bmatrix} - 0.1 \begin{bmatrix} -0.03 & -0.00 & -0.02 \end{bmatrix} = \\
&= \begin{bmatrix} -0.247 & 1.46 & -2.058 \end{bmatrix}
\end{aligned} \tag{4.88}
$$

$$
\begin{aligned}
(W^2)^{(2)} &= (W^2)^{(1)} - \eta \frac{\partial C}{\partial W^2} = \\
&= \begin{bmatrix} 1.62 & -0.61 & -0.53 \\ -1.07 & 0.87 & -2.30 \\ 1.74 & -0.76 & 0.32 \end{bmatrix} - 0.1 \begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \\ 0.02 & 0.03 & 0.02 \end{bmatrix} = \\
&= \begin{bmatrix} 1.62 & -0.61 & -0.53 \\ -1.07 & 0.87 & -2.30 \\ 1.738 & -0.763 & 0.318 \end{bmatrix}.
\end{aligned} \tag{4.89}
$$

Fica exemplificado o cálculo da derivada pelo *backpropagation* e mostrada uma iteração do SGD.

Os cálculos das derivadas feitos acima consideraram as amostras de forma separada. Usualmente, com o objetivo de otimizar esses cálculos, para cada mini-lote, considera-se todo o conjunto de dados. Dessa forma, temos

$$X = [x^{(1)}, x^{(2)}] = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 1.5 & 1 \end{bmatrix} \tag{4.90}$$

Daí

$$Z^2 = W^2 X = \begin{bmatrix} 1.62 & -0.61 & -0.53 \\ -1.07 & 0.87 & -2.30 \\ 1.74 & -0.76 & 0.32 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 1.5 & 1 \end{bmatrix} = \begin{bmatrix} -0.39 & 1.09 \\ -2.78 & -3.37 \\ 0.70 & 2.06 \end{bmatrix} \tag{4.91}$$

$$A^2 = \sigma(Z^2) = \begin{bmatrix} 0.40 & 0.75 \\ 0.06 & 0.03 \\ 0.67 & 0.89 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0.06 & 0.03 \\ 0.67 & 0.89 \end{bmatrix} \tag{4.92}$$

$$Z^3 = W^3 A^2 = \begin{bmatrix} -0.25 & 1.46 & -2.06 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0.06 & 0.03 \\ 0.67 & 0.89 \end{bmatrix} = \begin{bmatrix} -1.54 & -2.04 \end{bmatrix} \tag{4.93}$$

$$A^3 = \sigma(Z^3) = \begin{bmatrix} 0.18 & 0.12 \end{bmatrix} \tag{4.94}$$

$$\begin{aligned} \frac{\partial C}{\partial W^3} &= \frac{1}{4} \underbrace{\left(\frac{\partial C}{\partial A^3} \odot \sigma'(Z^3) \right)}_{\delta^3} \cdot \left(A^2\right)^T = \frac{1}{4} \underbrace{\left((A^3 - a(X)) \odot \sigma'(Z^3) \right)}_{\delta^3} \cdot \left(A^2\right)^T = \\ &= \frac{1}{4} \underbrace{\left(\left(\begin{bmatrix} 0.18 & 0.12 \end{bmatrix} - \begin{bmatrix} 1 & 0 \end{bmatrix} \right) \odot \sigma'(\begin{bmatrix} -1.54 & -2.04 \end{bmatrix}) \right)}_{\delta^3} \cdot \left(A^2\right)^T = \\ &= \frac{1}{4} \underbrace{\left(\begin{bmatrix} -0.82 & 0.12 \end{bmatrix} \odot \begin{bmatrix} 0.15 & 0.10 \end{bmatrix} \right)}_{\delta^3} \cdot \left(\begin{bmatrix} 1 & 1 \\ 0.06 & 0.03 \\ 0.67 & 0.89 \end{bmatrix} \right)^T = \\ &= \frac{1}{4} \underbrace{\begin{bmatrix} -0.12 & 0.01 \end{bmatrix}}_{\delta^3} \cdot \begin{bmatrix} 1 & 0.06 & 0.67 \\ 1 & 0.03 & 0.89 \end{bmatrix} = \begin{bmatrix} -0.03 & 0.00 & -0.02 \end{bmatrix} \end{aligned} \tag{4.95}$$

$$
\begin{aligned}
\frac{\partial C}{\partial W^2} &= \frac{1}{4}\underbrace{\left(\left(\left(W^3\right)^T \cdot \delta^3\right) \odot \sigma'(Z^2)\right)}_{\delta^2}\left(A^1\right)^T = \\
&= \frac{1}{4}\underbrace{\left(\left(\begin{bmatrix} -0.25 \\ 1.46 \\ -2.06 \end{bmatrix} \cdot \begin{bmatrix} -0.12 & 0.01 \end{bmatrix}\right) \odot \sigma'\left(\begin{bmatrix} -0.39 & 1.09 \\ -2.78 & -3.37 \\ 0.70 & 2.06 \end{bmatrix}\right)\right)}_{\delta^2}\left(A^1\right)^T = \\
&= \frac{1}{4}\underbrace{\left(\begin{bmatrix} 0.03 & 0.00 \\ -0.18 & 0.01 \\ 0.25 & -0.02 \end{bmatrix} \odot \begin{bmatrix} 0.24 & 0.19 \\ 0.06 & 0.03 \\ 0.22 & 0.10 \end{bmatrix}\right)}_{\delta^2}\left(A^1\right)^T = \\
&= \frac{1}{4}\underbrace{\begin{bmatrix} 0.01 & 0.00 \\ -0.01 & 0.00 \\ 0.06 & 0.00 \end{bmatrix}}_{\delta^2}\begin{bmatrix} 1 & 2 & 1.5 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.02 & 0.03 & 0.02 \end{bmatrix}.
\end{aligned}
\tag{4.96}
$$

Nota-se que as derivadas obtidas utilizando-se as amostras separadamente ou o conjunto de dados de uma só vez são iguais.

5 FDIPA - UM ALGORITMO DE PONTOS INTERIORES E DIREÇÕES VIÁVEIS

Neste capítulo, apresentamos as principais características do FDIPA. Explicamos como são obtidos os sistemas lineares que definem a direção de busca do método e também como se define o passo dado nessa direção.

No caso geral, o problema de otimização de redes neurais é irrestrito. Entretanto, o FDIPA resolve problemas com restrições. Para que pudéssemos então aplicar o FDIPA ao contexto de redes neurais, foi preciso a introdução de uma única restrição ao problema original. Isto levou a uma simplificação das fórmulas que definem a direção de busca do algoritmo. Esses detalhes serão também explicados neste capítulo. Um estudo mais profundo a respeito do FDIPA pode ser encontrado em [5].

Iniciaremos o capítulo introduzindo o problema de otimização não linear diferenciável e a caracterização de suas soluções para, então, apresentarmos o FDIPA.

5.1 PROBLEMA DE OTIMIZAÇÃO NÃO LINEAR DIFERENCIÁVEL

Neste trabalho, consideraremos apenas restrições de desigualdade. Um problema de otimização não linear diferenciável com restrições de desigualdade pode ser escrito como:

$$(P) \quad \begin{cases} \text{minimizar} & f(x) \\ \text{sujeito a} & g(x) \leq 0 \end{cases} \tag{5.1}$$

onde $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, $g(x) = (g_1(x), g_2(x), \ldots, g_m(x))^t$, são funções de classe C^1 em $\mathbb{R}^n$. A restrição $g(x) \leq 0$ significa que cada função coordenada g_i de g satisfaz a essa condição, isto é, $g_i(x) \leq 0$ para todo $i \in \{1, 2, \ldots, m\}$.

Para que o problema seja considerado de otimização não linear, pelo menos uma das funções deve ser não linear.

Definição 5.1.1. Uma restrição g_i será dita ativa no ponto x se $g_i(x) = 0$.

Seja $I(x) = \{i | g_i(x) = 0\}$ o conjunto das restrições ativas no ponto x. Dizemos que x é um ponto regular se os vetores no conjunto $\{\nabla g_i(x)\}_{i \in I(x)}$ são linearmente independentes.

Consideremos a função lagrangiana, associada ao problema (P):

$$L(x, \lambda) = f(x) + \lambda^t g(x) \tag{5.2}$$

onde $\lambda \in \mathbb{R}^m$ é uma variável auxiliar chamada variável dual ou multiplicador de Lagrange. A matriz hessiana da função lagrangiana (5.2) é dada por:

$$H(x, \lambda) = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x) \tag{5.3}$$

O próximo teorema explicita as condições necessárias de otimalidade de primeira ordem de KKT para o problema (P). Esse teorema é uma versão simplificada das condições de KKT apresentadas no capítulo 3. No teorema, $G(x)$ denota a matriz diagonal tal que $G(x)_{ii} = g_i(x)$ para todo $i \in \{1, 2, \ldots, m\}$.

Teorema 5.1.1. *Seja x^* um ponto regular para o problema (P)* (5.1)*. Se x^* é um mínimo local de (P) então existe $\lambda^* \in \mathbb{R}^m$ tal que*

$$\begin{aligned} \nabla f(x^*) + Jg(x^*)^t\lambda^* &= 0 \\ G(x^*)\lambda^* &= 0 \\ \lambda^* &\geq 0 \\ g(x^*) &\leq 0 \end{aligned}$$

5.2 O ALGORITMO FDIPA

O FDIPA [5] é um algoritmo de otimização que converge globalmente para pontos de Karush-Kuhn-Tucker sendo simples de implementar, apresentando um bom desempenho no que diz respeito a robustez e eficiência. O método se baseia na solução de dois sistemas de equações lineares com a mesma matriz em cada iteração seguida de uma busca linear inexata. A resolução desses sistemas lineares é feita de modo a respeitar, em cada iteração, as restrições de desigualdade apresentadas no Teorema 5.1.1. Com isso, há a garantia de que o ponto solução será viável. Ou seja, a partir de um ponto inicial no interior da região viável, o FDIPA gera uma sequência de pontos que são interiores à região de viabilidade. Em cada iteração, primeiramente, resolve-se um sistema de equações lineares envolvendo as variáveis primal e dual obtendo-se uma direção de descida não necessariamente viável. Em seguida, o sistema linear é perturbado para produzir uma deflexão na direção obtida no sentido do interior da região de viabilidade de forma a se obter uma direção de descida que seja também viável. Por fim, uma busca linear é feita para se obter um novo ponto interior que melhore o objetivo. No que segue, mostraremos passo a passo como o FDIPA obtém sua direção de busca e detalharemos como a busca linear é realizada pelo algoritmo.

Consideremos o seguinte sistema linear extraído do Teorema 5.1.1:

$$\begin{cases} \nabla f(x) + Jg(x)^t\lambda = 0 & (5.4) \\ G(x)\lambda = 0 & (5.5) \end{cases}$$

Definindo

$$y = \begin{pmatrix} x \\ \lambda \end{pmatrix} \quad \text{e} \quad \Phi(y) = \begin{pmatrix} \nabla f(x) + Jg(x)^t\lambda \\ G(x)\lambda \end{pmatrix} \tag{5.6}$$

encontramos o jacobiano de $\Phi(y)$:

$$J\Phi(y) = \begin{pmatrix} H(x,\lambda) & Jg(x)^t \\ \Lambda Jg(x) & G(x) \end{pmatrix} \tag{5.7}$$

onde Λ é uma matriz diagonal de tal que $\Lambda_{ii} = \lambda_i$ para todo $i \in \{1, 2, \ldots, m\}$.

Considerando o ponto $y^k = (x^k, \lambda^k)$ na iteração k, encontra-se $y^{k+1} = (x^{k+1}, \lambda^{k+1})$ com uma iteração de Newton para resolver o sistema de equações lineares $\Phi(y) = 0$ definido pelas Equações (5.4) e (5.5) que pode ser escrito da seguinte forma:

$$J\Phi(y^k)(y^{k+1} - y^k) = -\Phi(y^k). \tag{5.8}$$

A Equação (5.8) pode ser reescrita, obtendo-se:

$$\begin{pmatrix} B^k & Jg(x^k)^t \\ \Lambda^k Jg(x^k) & G(x^k) \end{pmatrix} \begin{pmatrix} x - x^k \\ \lambda - \lambda^k \end{pmatrix} = - \begin{pmatrix} \nabla f(x^k) + Jg(x^k)^t \lambda^k \\ G(x^k)\lambda^k \end{pmatrix} \tag{5.9}$$

onde B^k é uma matriz de ordem n podendo ser a própria matriz hessiana $H(x^k, \lambda^k)$ (5.3). Para a garantia da convergência global do algoritmo [5], a matriz B^k escolhida deve ser simétrica e definida positiva podendo inclusive ser uma aproximação da Hessiana obtida por alguma técnica quasi-Newton. Por vezes, nos referimos a $y^{k+1} = (x^{k+1}, \lambda^{k+1})$ apenas como $y = (x, \lambda)$.

Pondo $d = x - x^k$, podemos reescrever o sistema (5.9) da seguinte forma:

$$\begin{cases} B^k d \quad\quad + Jg(x^k)^t \lambda = -\nabla f(x^k) & (5.10) \\ \Lambda^k Jg(x^k) d + \quad G(x^k)\lambda = 0. & (5.11) \end{cases}$$

A solução do sistema definido pelas Equações (5.10) e (5.11) nos fornece uma tupla (d_1^k, λ_1^k) onde d_1^k é uma direção e λ_1^k uma estimativa para λ. Em [5] foi provado que $(d_1^k)^t \nabla f(x^k) < 0$, isto é, d_1^k é uma direção de descida para a função f. Entretanto, não há nenhuma garantia, até o momento, que a direção d_1^k seja viável.

De fato, caso o ponto x^k esteja "próximo" da curva $g_i(x^k) = 0$ para algum i, a direção d_1^k pode deixar de ser viável pois nesses caso, d_1^k tende a uma direção tangente ao conjunto viável. Com efeito, se reescrevermos a Equação (5.11) temos que :

$$\lambda_i \nabla g_i(x^k) d_1^k + g_i(x^k)\lambda_i = 0, \qquad i = 1, 2, ..., m \tag{5.12}$$

e isso implica que $\nabla g_i(x^k) d_1^k = 0$ para todo i tal que $g_i(x^k) = 0$, ou seja, d_1^k é tangente a curva $g_i(x) = 0$ e portanto, é uma direção que sai da região viável.

Uma solução para esse problema, é realizar o cálculo de uma nova direção de restauração para impedir que a direção original seja inviável. Para isso, é definido um novo sistema linear em d e $\overline{\lambda}$ a partir do sistema das Equações (5.10) e (5.11) adicionando um termo no lado direito da Equação (5.11) com um fator escalar $\rho \in \mathbb{R}_+^*$ e um vetor $\omega \in (\mathbb{R}_+^*)^m$. O sistema fica

$$\begin{cases} B^k d \quad\quad + Jg(x^k)^t \overline{\lambda} = -\nabla f(x^k) & (5.13) \\ \Lambda^k Jg(x^k) d + \quad G(x^k)\overline{\lambda} = -\rho_k \Lambda^k \omega^k. & (5.14) \end{cases}$$

Agora, a Equação (5.14) é equivalente a

$$\lambda_i^k \nabla g_i(x^k)^t d + g_i(x)\overline{\lambda}_i = -\rho_k \lambda_i^k \omega_i^k, \quad i = 1, 2, \ldots, m \tag{5.15}$$

e com isso, $\nabla g_i^t(x^k)d = -\rho_k \omega_i^k$ para as restrições ativas em x^k. Sendo $\rho_k > 0$ e $\omega_i^k > 0$ para todo i, temos que $-\rho_k \omega_i^k < 0$ e, portanto, a direção d calculada pelo novo sistema é viável uma vez que $\nabla g_i^t(x^k)d < 0$. Esse novo sistema, definido pelas Equações (5.13) e (5.14) produz uma direção d que é viável mas que pode não ser de descida para a função f. Para resolver esse problema, o sistema perturbado é desacoplado, dando origem aos seguintes sistemas:

$$\begin{cases} B^k d_1^k + Jg(x^k)^t \lambda_1^k = -\nabla f(x^k) & (5.16) \\ \Lambda^k Jg(x^k) d_1^k + G(x^k)\lambda_1^k = 0 & (5.17) \end{cases}$$

cujas soluções são d_1^k e λ_1^k e

$$\begin{cases} B^k d_2^k + Jg(x^k)^t \lambda_2^k = 0 & (5.18) \\ \Lambda^k Jg(x^k) d_2^k + G(x^k)\lambda_2^k = -\Lambda^k \omega^k & (5.19) \end{cases}$$

que tem como solução d_2^k e λ_2^k. Com isso, definimos a direção d e o multiplicador $\overline{\lambda}$ na iteração k como $d^k = d_1^k + \rho_k d_2^k$ e $\overline{\lambda}^k = \lambda_1^k + \rho_k \lambda_2^k$.

Apresentaremos agora a condição para que d^k seja também de descida. Já sabemos que $(d_1^k)^t \nabla f(x^k) < 0$. Da definição de d^k obtém-se:

$$(d^k)^t \nabla f(x^k) = (d_1^k)^t \nabla f(x^k) + \rho_k (d_2^k)^t \nabla f(x^k). \tag{5.20}$$

Se tivermos $(d_2^k)^t \nabla f(x^k) \leq 0$ então teremos que $(d^k)^t \nabla f(x^k) < 0$, $\forall \rho_k \in \mathbb{R}_+^*$. Portanto, devemos escolher ρ_k adequadamente no caso em que $(d_2^k)^t \nabla f(x^k) > 0$. Neste caso, impondo-se a desigualdade:

$$(d^k)^t \nabla f(x^k) \leq \xi (d_1^k)^t \nabla f(x^k) < 0 \text{ com } \xi \in (0, 1) \tag{5.21}$$

obtém-se:

$$(d_1^k)^t \nabla f(x^k) + \rho_k (d_2^k)^t \nabla f(x^k) \leq \xi (d_1^k)^t \nabla f(x^k) < 0 \tag{5.22}$$

e, isolando ρ_k, vem:

$$\rho_k \leq (\xi - 1) \frac{(d_1^k)^t \nabla f(x^k)}{(d_2^k)^t \nabla f(x^k)}. \tag{5.23}$$

Escolhendo ρ_k de modo a respeitar a desigualdade em (5.23), para qualquer $\xi \in (0, 1)$, conclui-se que d^k será uma direção de descida para a função f.

Por fim, com a direção d^k calculada na iteração k, atualiza-se o ponto x^{k+1} segundo a regra:

$$x^{k+1} = x^k + t_k d^k \tag{5.24}$$

onde t_k é um valor que representa um tamanho de passo que será dado na direção d^k e é selecionado como o maior termo na sequência $\{1, \nu, \nu^2, \nu^3, \dots\}$ que satisfaz

$$\begin{aligned} &f(x^x + t_k d^k) \leq f(x^k) + \eta t_k \nabla f(x^k)(d^k)^t \text{ e } g_i(x^k + t_k d^k) < 0, && \text{se } \overline{\lambda}_i^k \geq 0 \\ &f(x^x + t_k d^k) \leq f(x^k) + \eta t_k \nabla f(x^k)(d^k)^t \text{ e } g_i(x^k + t_k d^k) \leq g_i(x^k), && \text{se } \overline{\lambda}_i^k < 0. \end{aligned} \tag{5.25}$$

A matriz Λ que define os sistemas lineares é uma matriz diagonal cujos elementos são formados pelas coordenadas do vetor $\lambda \in (\mathbb{R}_+^*)^m$. Os valores de $\epsilon, \xi, \eta, \varphi$ e ν são fixados antes do algoritmo iniciar. Além disso, usando-se diferentes regras para a atualização da matriz B e dos vetores ω e λ obtém-se uma família de algoritmos. Existem diversas técnicas de atualização da matriz B. Uma delas é a atualização BFGS (Broyden–Fletcher–Goldfarb–Shanno) [31]. Outras técnicas podem ser vistas em [32], [5] e [33].

Para um melhor entendimento dos cálculos envolvidos, as dimensões das matrizes e vetores são:

- $x^k \in \mathbb{R}^{n\times 1}$;
- $B^k \in \mathbb{R}^{n\times n}$;
- $d_i^k \in \mathbb{R}^{n\times 1}$, para $i = 1, 2$;
- $Jg(x^k) \in \mathbb{R}^{m\times n}$;
- $\lambda_i^k \in \mathbb{R}^{m\times 1}$, para $i = 1, 2$;
- $\nabla f(x^k) \in \mathbb{R}^{n\times 1}$;
- $\Lambda^k \in \mathbb{R}^{m\times m}$;
- $G(x^k) \in \mathbb{R}^{m\times m}$;
- $\omega \in \mathbb{R}^{m\times 1}$.

Abaixo, é apresentado uma versão bem simplificada de um pseudocódigo do algoritmo FDIPA:

5.3 O CASO PARTICULAR EM TELA

Para esse trabalho, algumas particularidades foram consideradas no algoritmo FDIPA. Por simplicidade, a matriz B foi adotada igual à matriz identidade. Vale destacar que o problema original é irrestrito e portanto, para a utilização do FDIPA, houve a necessidade da incorporação de uma restrição ao problema original. Consequentemente, $m = 1$ em (5.1). Dessa forma, conseguimos realizar manipulações nos sistemas lineares

Algoritmo 2: FDIPA

Dados: $x \in \Omega^0$, $\lambda \in (\mathbb{R}^*_+)^m$, $\omega \in (\mathbb{R}^*_+)^m$, $B \in \mathbb{R}^{n\times n}$ simétrica definida positiva.

Resultado: x^k

incia $\epsilon \in (0,1)$, $\xi \in (0,1)$, $\eta \in (0,1)$, $\varphi > 0$, $\nu \in (0,1)$, tol $\in (0,1)$ e $MAX\_ITER \in \mathbb{N}$;
k $\leftarrow 0$;
$x^1 \leftarrow x$;
enquanto $k < MAX\_ITER$ **faça**
  k $\leftarrow$ k + 1;
  Calcule d_1^k e λ_1^k resolvendo o sistema das equações (5.16) e (5.17);
  se $d_1^k < \epsilon$ **então**
    retorne x^k;
  Calcule d_2^k e λ_2^k resolvendo o sistema das equações (5.18) e (5.19);
  se $d_2^k(x^k) > 0$ **então**
    $\rho_k \leftarrow \min\left\{\varphi\|d_1^k\|^2, (\xi-1)\frac{(d_1^k)^t\nabla f(x^k)}{(d_2^k)^t\nabla f(x^k)}\right\}$;
  senão
    $\rho_k \leftarrow \varphi\|d_1^k\|^2$;
  $d^k \leftarrow d_1^k + \rho_k d_2^k$;
  $\overline{\lambda}^k \leftarrow \lambda_1^k + \rho_k\lambda_2^k$;
  Calcule t_k o maior termo na sequência $\{1, \nu, \nu^2, \nu^3, \dots\}$ que satisfaz (5.25);
  se $t < tol$ **então**
    retorne x^k;
  $x^k \leftarrow x^k + t_k d^k$;
  Atualiza B^k simétrica definida positiva, $\omega \in (\mathbb{R}^*_+)^m$ e $\lambda \in (\mathbb{R}^*_+)^m$;
retorne x^k;

para chegar em expressões diretas para as direções d_1 e d_2. Estamos no caso em que $g : \mathbb{R}^n \longrightarrow \mathbb{R}$, $B^k = I$, $\Lambda^k = \lambda \in \mathbb{R}$ e $G(x^k) = g(x^k)$ para toda iteração k.

Para o primeiro sistema, começamos isolando λ_1^k da Equação (5.17) e obtemos:

$$\lambda_1^k = -\frac{\Lambda^k}{G(x^k)} Jg(x^k) d_1^k. \tag{5.26}$$

Substituindo (5.26) em (5.16) e isolando d_1^k temos:

$$\begin{aligned} B^k d_1^k + Jg(x^k)^t\left(-\frac{\Lambda^k}{G(x^k)} Jg(x^k) d_1^k\right) = -\nabla f(x^k) \implies \\ \left(B^k - \frac{\Lambda^k}{G(x^k)} Jg(x^k)^t Jg(x^k)\right) d_1^k = -\nabla f(x^k) \implies \\ d_1^k = -\left(B^k - \frac{\Lambda^k}{G(x^k)} Jg(x^k)^t Jg(x^k)\right)^{-1} \nabla f(x^k). \end{aligned} \tag{5.27}$$

Essa inversa foi calculada através da formula de Sherman-Morrison considerando a matriz $A = B^k = I$, $u = -\frac{\Lambda^k}{G(x^k)} Jg(x^k)^t$ e $v = Jg(x^k)^t$. Por simplicidade, realizamos as cálculos

abaixo suprimindo-se o índice k, obtendo:

$$\begin{aligned}\left(B - \frac{\Lambda}{G(x)}Jg(x)^tJg(x)\right)^{-1} &= B^{-1} - \frac{B^{-1}(-\frac{\Lambda}{G(x)}Jg(x)^t)(Jg(x)^t)^tB^{-1}}{1+(Jg(x)^t)^t(B)^{-1}(-\frac{\Lambda}{G(x)}Jg(x)^t)} = \\ &= I^{-1} + \frac{I^{-1}\frac{\Lambda}{G(x)}Jg(x)^tJg(x)I^{-1}}{1-\frac{\Lambda}{G(x)}Jg(x)I^{-1}Jg(x)^t} = \\ &= I + \frac{\frac{\Lambda}{G(x)}Jg(x)^tJg(x)}{1-\frac{\Lambda}{G(x)}Jg(x)Jg(x)^t}.\end{aligned} \tag{5.28}$$

Multiplicando a ultima fração por $\frac{G(x)}{\Lambda}$ obtemos:

$$\left(B - \frac{\Lambda}{G(x)}Jg(x)^tJg(x)\right)^{-1} = I + \frac{Jg(x)^tJg(x)}{\frac{G(x)}{\Lambda} - Jg(x)Jg(x)^t}. \tag{5.29}$$

A direção d_1^k fica então dada por:

$$d_1^k = -\left(I + \frac{Jg(x)^tJg(x)}{\frac{G(x)}{\Lambda} - Jg(x)Jg(x)^t}\right)\nabla f(x) = -\nabla f(x) - \frac{Jg(x)^tJg(x)\nabla f(x)}{\frac{G(x)}{\Lambda} - Jg(x)Jg(x)^t}. \tag{5.30}$$

Iremos agora, encontrar uma expressão direta para a direção d_2^k trabalhando com o segundo sistema de equações lineares. Assim como feito anteriormente, vamos primeiro isolar o termo λ_2^k da Equação (5.19):

$$\lambda_2^k = -\frac{\Lambda^k}{G(x^k)}\omega^k - \frac{\Lambda^k}{G(x^k)}Jg(x^k)d_2^k. \tag{5.31}$$

Substituindo (5.31) em (5.18) e isolando d_2^k obtemos:

$$\begin{aligned} B^kd_2^k + Jg(x^k)^t\left(-\frac{\Lambda^k}{G(x^k)}\omega^k - \frac{\Lambda^k}{G(x^k)}Jg(x^k)d_2^k\right) &= 0 \\ B^kd_2^k - \frac{\Lambda^k}{G(x^k)}Jg(x^k)^t\omega^k - \frac{\Lambda^k}{G(x^k)}Jg(x^k)^tJg(x^k)d_2^k &= 0 \\ B^kd_2^k - \frac{\Lambda^k}{G(x^k)}Jg(x^k)^tJg(x^k)d_2^k &= \frac{\Lambda^k}{G(x^k)}Jg(x^k)^t\omega^k \\ \left(B^k - \frac{\Lambda^k}{G(x^k)}Jg(x^k)^tJg(x^k)\right)d_2^k &= \frac{\Lambda^k}{G(x^k)}Jg(x^k)^t\omega^k \\ d_2^k = \left(B^k - \frac{\Lambda^k}{G(x^k)}Jg(x^k)^tJg(x^k)\right)^{-1}&\frac{\Lambda^k}{G(x^k)}Jg(x^k)^t\omega^k \end{aligned} \tag{5.32}$$

e como já calculamos essa inversa em (5.29) temos que a direção d_2^k é dada por:

$$\begin{aligned} d_2^k &= \left(I + \frac{Jg(x)^tJg(x)}{\frac{G(x)}{\Lambda} - Jg(x)Jg(x)^t}\right)\frac{\Lambda}{G(x)}Jg(x)^t\omega = \\ &= \frac{\Lambda}{G(x)}Jg(x)^t\omega + \frac{\Lambda}{G(x)}\frac{Jg(x)^tJg(x)Jg(x)^t\omega}{\frac{G(x)}{\Lambda} - Jg(x)Jg(x)^t}. \end{aligned} \tag{5.33}$$

Obtivemos uma expressão explícita para a direção d_1^k, que depende essencialmente do gradiente da função f, bem como para a direção d_2^k.

As direções d_1 e d_2 são vetores coluna de dimensão $n \times 1$. Na Formula (5.30) que fornece a direção d_1^k aparece a multiplicação $Jg(x)^t Jg(x)\nabla f(x)$. Perceba que, $Jg(x)$ tem dimensão $1 \times n$. Com isso, se fizermos o calculo de forma sequencial iremos primeiro multiplicar $Jg(x)^t Jg(x)$ cujo resultado nos dará uma matriz de dimensão $n \times n$ que, em termos computacionais, é mais custoso, principalmente no caso de redes neurais onde n, o número de neurônios da rede, é muito grande. Podemos então realizar primeiro a multiplicação $Jg(x)\nabla f(x)$ cujo resultado é um número real dado que $\nabla f(x)$ tem dimensão $n \times 1$. Finalmente, multiplicamos esse escalar obtido por $Jg(x)^t$. O mesmo pode ser feito para a direção d_2^k, de forma que nenhuma matriz é criada durante as iterações do algoritmo.

Para a função particular utilizada neste trabalho, qual seja, a função $g : \mathbb{R}^n \longrightarrow \mathbb{R}$ definida por

$$g(x) = \|x\|^2 - r^2 \tag{5.34}$$

o jacobiano de g fica

$$Jg(x) = 2x. \tag{5.35}$$

As expressões das direções podem então ser escritas da seguinte forma:

$$d_1^k = -\nabla f(x) - \frac{(2x)^t 2x \nabla f(x)}{\frac{\|x\|^2 - r^2}{\lambda} - 2x(2x)^t} = -\nabla f(x) - \frac{4(x^t x)\nabla f(x)}{\frac{\|x\|^2 - r^2}{\lambda} - 4\|x\|^2}. \tag{5.36}$$

$$\begin{aligned} d_2^k &= \frac{\lambda}{\|x\|^2 - r^2}(2x)^t\omega + \frac{\lambda}{\|x\|^2 - r^2}\frac{(2x)^t 2x(2x)^t\omega}{\frac{\|x\|^2 - r^2}{\lambda} - 2x(2x)^t} = \\ &= \frac{2\lambda}{\|x\|^2 - r^2}x^t\omega + \frac{\lambda}{\|x\|^2 - r^2}\frac{8(x^t x x^t)\omega}{\frac{\|x\|^2 - r^2}{\lambda} - 4\|x\|^2} = \\ &= \frac{2\lambda}{\|x\|^2 - r^2}x^t\omega + \frac{\lambda}{\|x\|^2 - r^2}\frac{8(x^t\|x\|^2)\omega}{\frac{\|x\|^2 - r^2}{\lambda} - 4\|x\|^2}. \end{aligned} \tag{5.37}$$

6 METODOLOGIA

Este capítulo apresenta a metodologia adotada na produção desse trabalho. Serão detalhados os passos para o desenvolvimento dos algoritmos, as etapas de realização dos testes, bem como os parâmetros utilizados.

6.1 IMPLEMENTAÇÃO DOS ALGORITMOS

Para a produção deste trabalho, todos os algoritmos foram implementados em linguagem Python. Em primeiro lugar foi implementado o algoritmo responsável pela implantação das redes neurais que aparecem no trabalho. Depois, implementamos o algoritmo *backpropagation* para o cálculo das derivadas e em seguida, o algoritmo de treinamento SGD. Feito isso, ficamos em condições de implementar o algoritmo FDIPA.

Em um primeiro momento, foi implementada a versão básica do FDIPA conforme descrita no capítulo 5. Porém, a representação dos pesos em uma rede neural, que correspondem às variáveis a serem otimizadas, não é compatível com a estrutura dos dados com os quais o FDIPA lida. Em um segundo momento foi feita uma adaptação nas matrizes que representam os pesos da rede neural para a compatibilização com o FDIPA.

Figura 6 – Base de dados MNIST [1]

A implementação do algoritmo SGD teve como meta a comparação dos resultados obtidos por uma algoritmo já consagrado com os resultados obtidos através do emprego de um novo algoritmo, no caso o FDIPA, que foi o principal objetivo do trabalho.

[1] Imagem retirada de: https://commons.wikimedia.org/wiki/File:MnistExamples.png

6.2 TESTES

Para realização dos testes, necessitamos de um conjunto de dados para o treinamento da rede neural. Escolhemos a base de dados MNIST [34], sigla em inglês para "Banco de Dados Modificado do Instituto Nacional de Padrões e Tecnologia". A base de dados MNIST consiste em um conjunto de imagens digitalizadas de dígitos manuscritos, juntamente com a sua correta classificação. Na Figura 6 é apresentado um subconjunto dessa base de dados. A base de dados MNIST é dividida em duas partes, a primeira contém um total de 60 mil imagens para serem usadas como dados de treinamento e a segunda contém 10 mil imagens usadas como dados de teste. Cada imagem está na escala em cinza e tem tamanho 28×28 pixels. Para serem utilizados pela rede neural, as imagens são disponibilizadas como um vetor de tamanho $784 = 28 \times 28$ onde em cada posição do vetor temos a intensidade de escala em cinza daquele pixel. Na Figura 7 temos um exemplo da representação em uma matriz de uma amostra na base de dados. O vetor usado como entrada pelo algoritmo da rede neural é uma transformação da matriz, mostrada na Figura 7, através de uma "linearização".

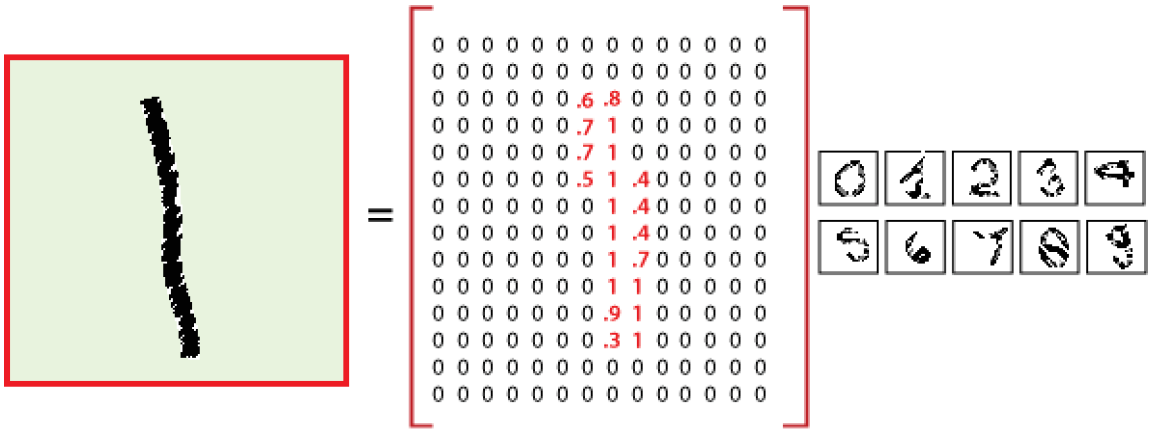


Figura 7 – Representação da imagem em uma matriz. [2]

Alguns parâmetros foram mantidos constantes ao longo dos testes realizados. São eles:

- Tamanho do mini-lote = 128;
- Número de épocas = 100;
- Número de camadas ocultas na rede = 2;

[2] Imagem retirada de: https://www.javatpoint.com/tensorflow-mnist-dataset-in-cnn

- Número de neurônios nas camadas ocultas = 32 neurônios em cada camada oculta.

Com esses parâmetros, obtemos 27324 pesos na rede para serem treinados. Também foram escolhidos constantes os seguintes parâmetros do FDIPA: $\varphi = 0.1$, $\xi = 0.8$, $\epsilon = 10^{-6}$, $\omega = 1$, $\lambda = 1$, $\nu = 0.9$, por terem sido os melhores parâmetros quando da resolução de diversos problemas da literatura pelo FDIPA. Além disso, a matriz B^k foi mantida igual a identidade em todas as iterações.

Foi incorporado ao problema irrestrito constituído da minimização da função custo uma restrição ao tamanho dos pesos definida pela função $g : \mathbb{R}^n \longrightarrow \mathbb{R}$ tal que:

$$g(x) = \|x\|^2 - r^2. \tag{6.1}$$

Como a nossa restrição é do tipo menor ou igual a 0 temos que:

$$g(x) \leq 0 \implies \|x\|^2 - r^2 \leq 0 \implies \|x\|^2 \leq r^2 \implies \|x\| \leq r. \tag{6.2}$$

Um estudo sobre a influência do parâmetro r nas soluções será também apresentado.

No pseudocódigo apresentado no Algoritmo 2, um dos critérios de parada é o número máximo de iterações que foi representado por MAX_ITER. Um estudo sobre a influência desse hiperparâmetro será apresentado.

Outro hiperparâmetro que iremos investigar para o FDIPA será o valor t que representa o tamanho do passo do algoritmo na direção d, obtida em cada iteração. Na versão básica do algoritmo, inicia-se com o valor $t = 1$ e multiplica-se t por um número $\nu \in (0, 1)$ até que o próximo ponto seja viável. Entretanto, no nosso contexto, a escolha inicial $t = 1$ acarretou muitas iterações do algoritmo até que um ponto viável fosse encontrado. Isso motivou estudos sobre a escolha inicial para o tamanho do passo, conforme será apresentado no Capítulo 7.

Foi utilizado um algoritmo de Evolução Diferencial [35] para obter os melhores hiperparâmetros do FDIPA e do SGD. Com relação ao FDIPA, os hiperparâmetros de interesse foram o raio r da função g, o número de iterações e o tamanho do passo t. Com relação ao SGD, o único hiperparâmetro estudado foi a taxa de aprendizagem η.

Com essa finalidade, definimos uma função que calcula a média da acurácia em uma estratégia de 3-*fold* e retorna o valor 1 após a subtração da média da acurácia.Os parâmetros do algoritmo evolução diferencial utilizados foram:

- Número de indivíduos na população (5 para o FDIPA) e (8 para o SGD);
- número máximo de gerações que a população é evoluída: 20;
- constante de mutação : 0.9;
- constante de recombinação : 0.9;

- tolerância para convergência : 10^{-5}.

Após a obtenção dos melhores hiperparâmetros, foram realizadas algumas simulações com objetivo de investigar a influência desses no algoritmo. Para isso, no caso do FDIPA, foram fixados dois hiperparâmetros, como sendo o valor ótimo obtido pelo evolução diferencial, e feita a variação do terceiro. Esses resultados são apresentados no Capítulo 7.

Toda implementação e testes foram feitos utilizando-se um notebook com processador Intel® Core™ i7-7500U CPU @ 2.70GHz × 4, 16Gb de memória RAM e sistema operacional Linux Ubuntu 20.04.3 LTS. A função de evolução diferencial foi rodada em paralelo para a obtenção dos hiperparâmetros dos dois modelos. Os demais testes não foram feitos de forma paralela. O código desenvolvido para esse trabalho pode ser encontrado integralmente em <https://github.com/vitormgou/FDIPA-ANN>.

7 RESULTADOS

Para o FDIPA, o algoritmo Evolução Diferencial foi rodado duas vezes para duas sementes diferentes. Na primeira vez, deixamos livres o tamanho do passo t, o número de iterações e o raio r para serem otimizados. Com a semente igual a 1 para o algoritmo Evolução Diferencial, obtivemos $r = 49$, $t = 0.23111951$ e número de iterações igual a 2. O algoritmo levou 431970.4 segundos, isto é, aproximadamente 120 horas, e a acurácia obtida com esse modelo foi de 0.955740037 no conjunto de teste.

Na segunda vez, deixamos fixo o número de iterações em 2. Com a semente igual a 2, obtivemos $r = 47.40463817$ e $t = 0.3239329$. O algoritmo levou 602558.4 segundos. A acurácia obtida com esse modelo foi de 0.953859976 no conjunto de teste.

Feito isso, para o FDIPA, foi considerado o primeiro conjunto de hiperparâmetros obtidos, isto é, $r = 49$, $t = 0.23111951$ e número de iterações igual a 2.

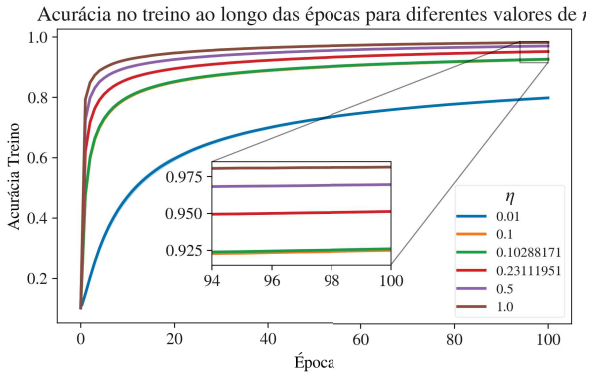
Para o algoritmo SGD, foi encontrado o hiperparâmetro $\eta = 0.10288171$ e o tempo gasto foi de 82831.3 segundos ou aproximadamente 23 horas. A acurácia obtida com esse modelo no conjunto de teste foi de 0.889.

Para os resultados nas tabelas e gráficos, serão utilizados algumas abreviações dos termos. Máximo será referido como máx, mínimo como mín, o desvio padrão como std.

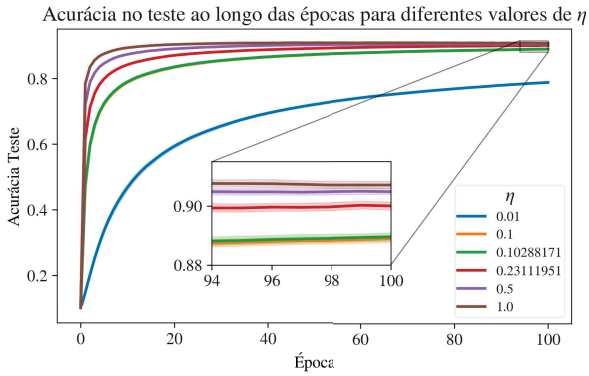
7.1 RESULTADOS SGD

Para o SGD, foram escolhidos 6 valores de taxa de aprendizagem η para variar e coletar os resultados. Os valores de η escolhidos foram $0.01, 0.1, 0.10288171, 0.23111951, 0.5$ e 1.0. O valor de $\eta = 1$ foi o que se saiu melhor, em média, na acurácia no treino e na acurácia no conjunto de teste como pode ser visto na Tabela 1. Para todos os valores de η houve um valor baixo de desvio padrão indicando que o método não foi afetado pela escolha da semente inicial que altera a inicialização dos pesos da rede. Isso pode ser observado nas Tabelas 1 e 3. Nas Tabelas 2 e 4, é possível observar que o valor de $\eta = 1$ obteve o melhor resultado da maior acurácia e do menor custo nos conjuntos de treino e de teste. Em todas as Tabelas referentes aos resultados do SGD, observamos uma tendência de melhora no resultado com o aumento da taxa de aprendizado η. Também é possível observar que não existe diferença entre o tempo gasto no treinamento por diferentes valores de η utilizados. Esse fato também pode ser observado pelos gráficos na Figura 8. As Figuras 8 (a) e (b) mostram que o treinamento aumenta a acurácia no conjunto de treino e de teste e pondo em destaque o fato de que, quanto maior a taxa de aprendizagem η, mais rápido a acurácia aumenta, chegando a valores de mais altos ao fim das 100 épocas de treinamento. De forma análoga, as Figuras 8 (e) e (f) apresentam a queda da função de custo nos conjuntos de treino e de teste, ocorrendo a queda da função mais rapidamente nos valores de η mais altos. Para todos os valores de η, o treinamento foi bem comportado para diferentes

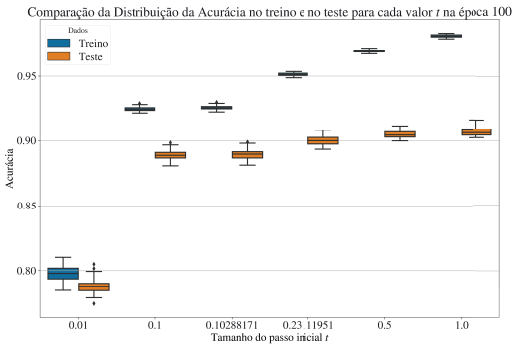
sementes apresentando pouca variância tanto no conjunto de treino quanto no conjunto de teste, como podemos ver na Figura 8 (c). Para todos os valores de η considerados, a norma do peso não se alterou muito ao longo das épocas, como pode ser visto na Figura 8 (d) e na Tabela 5. Como o valor da taxa de aprendizado η é um escalar que multiplica a direção d, interpretando-a como um tamanho de passo, os resultados obtidos disponíveis na Tabela 5 mostram que, com um tamanho de passo maior, em cada iteração o passo dado na direção do gradiente é maior e, portanto, pode haver uma mudança maior na posição do peso da rede em $\mathbb{R}^n$ se comparado a tamanho de passos menores. E com isso, a norma do peso terá uma maior variação ao longo das épocas.



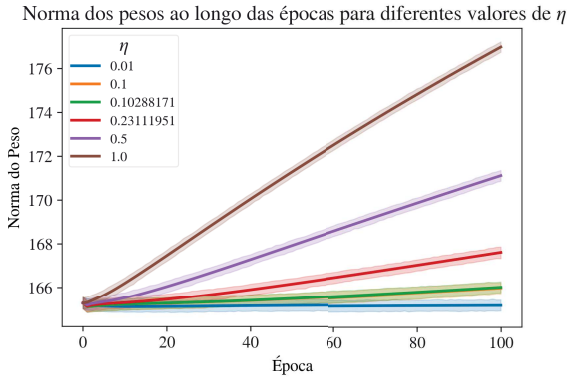
(a) Acurácia no Treino



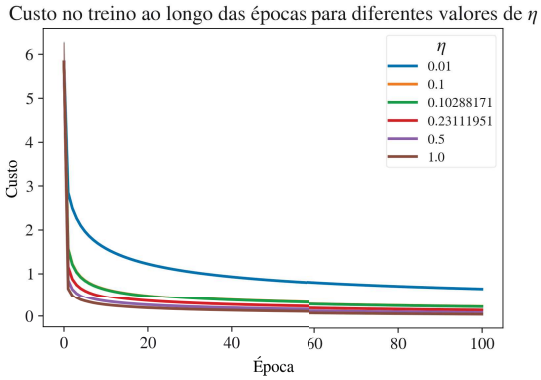
(b) Acurácia no Teste



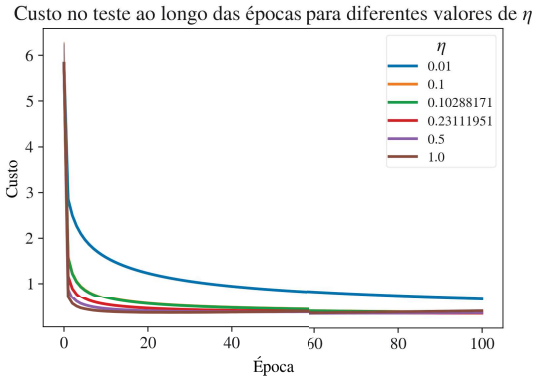
(c) Acurácia na época 100



(d) Norma dos pesos da rede



(e) Custo no Treino



(f) Custo no Teste

Figura 8 – Acurácia, custo e norma do peso para diferentes valores de η.

Tabela 1 – Médias da Acurácia e do tempo, em segundos, com o desvio padrão para diferentes valores de η.

η	Acurácia Treino (± std)	Acurácia Teste (± std)	Tempo (± std)
0.010	0.797 (± 0.006)	0.788 (± 0.007)	215.498 (± 42.665)
0.100	0.925 (± 0.002)	0.889 (± 0.004)	195.325 (± 0.843)
0.103	0.926 (± 0.002)	0.889 (± 0.004)	196.345 (± 1.532)
0.231	0.951 (± 0.001)	0.900 (± 0.004)	196.086 (± 0.773)
0.500	0.969 (± 0.001)	0.905 (± 0.003)	196.170 (± 1.381)
1.000	0.981 (± 0.001)	0.907 (± 0.003)	196.161 (± 0.685)

Tabela 2 – Valores máximos da Acurácia e do tempo, em segundos, para diferentes valores de η.

η	Acurácia Treino Máx	Acurácia Teste Máx	Tempo Máx
0.010	0.810	0.805	349.478
0.100	0.929	0.898	198.422
0.103	0.930	0.899	202.088
0.231	0.953	0.908	197.693
0.500	0.972	0.912	202.348
1.000	0.983	0.916	197.620

Tabela 3 – Médias da função Custo e do tempo, em segundos, com o desvio padrão para diferentes valores de η.

η	Custo Treino (± std)	Custo Teste (± std)	Tempo (± std)
0.010	0.639 (± 0.016)	0.670 (± 0.019)	215.498 (± 42.67)
0.100	0.253 (± 0.005)	0.375 (± 0.012)	195.325 (± 0.843)
0.103	0.250 (± 0.005)	0.374 (± 0.012)	196.345 (± 1.532)
0.231	0.172 (± 0.005)	0.353 (± 0.012)	196.086 (± 0.773)
0.500	0.114 (± 0.004)	0.367 (± 0.014)	196.170 (± 1.381)
1.000	0.073 (± 0.004)	0.407 (± 0.017)	196.161 (± 0.685)

Tabela 4 – Valores mínimos da função Custo e do tempo, em segundos, para diferentes valores de η.

η	Custo Treino Mín	Custo Teste Mín	Tempo Mín
0.010	0.600	0.621	194.481
0.100	0.237	0.347	194.060
0.103	0.234	0.346	194.975
0.231	0.164	0.324	194.590
0.500	0.106	0.334	194.669
1.000	0.065	0.377	194.562

O tempo total para rodar os 180 testes do SGD, com 30 sementes para cada um dos 6 valores de η, foi de $35.867,54$ segundos, ou $9,9632$ horas.

Tabela 5 – Norma do Peso inicial e final para diferentes valores de η.

η	Norma do peso na época 1				Norma do peso na época 100			
	Mínimo	Máximo	Média	Std	Mínimo	Máximo	Média	Std
0.010	163.674	167.078	165.264	0.745	163.622	167.054	165.207	0.746
0.100	163.674	167.078	165.264	0.745	164.433	167.703	165.971	0.739
0.103	163.674	167.078	165.264	0.745	164.467	167.732	166.004	0.738
0.231	163.674	167.078	165.264	0.745	166.084	169.170	167.606	0.725
0.500	163.674	167.078	165.264	0.745	169.774	172.503	171.114	0.689
1.000	163.674	167.078	165.264	0.745	175.829	178.407	176.984	0.679

7.2 RESULTADOS FDIPA

7.2.1 Variação do Raio

Foram escolhidos 5 valores de raio para avaliar, sendo eles: 2, 10, 50, 100 e 1000. Na Figura 9 podemos ver as medidas de acurácia e custo, no treino e no teste, e a norma dos pesos ao longo das épocas para os diferentes raios.

Para a métrica da acurácia, observamos que nos dados de treino e de teste o comportamento é igual para cada raio. O que muda de um gráfico para o outro nesse caso são as grandezas e a velocidade com que se alcança a assíntota horizontal como pode ser visto nas Figuras 9 (a) e (b). Na acurácia do treino, é possível ver que os raios 50 e 100, que obtiveram melhores resultados, estão muito próximo de 1 de acurácia enquanto no conjunto de teste eles estão menos próximos do valor 1, como pode ser observado na Figura 9 (a) e nas Tabelas 6 e 7. Observou-se um comportamento interessante em relação ao raio. Apesar da quantidade de raios testados ter sido pequena, podemos ver que a limitação adicionada pela restrição pode gerar baixa acurácia quando os raios são muito pequenos, como no caso 2 e 10 ou quando os raios são muito grandes, como no caso 1000, onde a acurácia levou mais tempo para atingir sua assíntota. Destaca-se que nesses casos, a acurácia foi menor que as obtidas com o raio igual a 50 ou 100. Ainda em relação a acurácia, nas Tabelas 6 e 7 observa-se que o modelo com raio igual a 100 se saiu melhor nos dados de treino e pior nos dados teste, se comparado ao modelo com raio igual a 50.

A acurácia final para os diferentes modelos nos dados de treino e de teste podem ser observados no *box plot* da acurácia na época 100 na Figura 9 (c). Observa-se que os melhores modelos, o com raio 50 e com raio 100, estão muito próximos no comportamento, tanto na acurácia obtida, quanto no desvio padrão muito baixo entre as execuções independentes. O modelo com raio igual a 2 obteve resultados muito ruins não variando muito e tendo um resultado próximo de 0.1 na acurácia do treino e do teste, o que mostra que esse modelo está basicamente chutando um valor de resposta, considerando que temos 10 classes no nosso problema. Os modelos com raio igual a 10 e 1000 apresentaram o maior desvio padrão, tanto na acurácia quanto no custo, como pode ser visto na Figura 9 e nas Tabelas 6 e 8, sendo que o modelo com raio igual a 10 obteve um desempenho por volta de 0.2 de

acurácia no conjunto de treino e de teste, enquanto que o modelo com raio igual a 1000 obteve acurácia por volta de 0.87 no conjunto de treino e de 0.82 no conjunto de teste.

Na Figura 9 (d) é mostrada a norma dos pesos da rede ao longo das épocas e podemos observar que a busca é realizada basicamente sobre a fronteira de uma bola com raio por volta de 45% do raio do modelo (por exemplo, para o modelo de raio 1000, o raio médio se manteve em torno de 450 durante todas as épocas). Além disso, os desvios padrões apresentados são proporcionais aos raios, isto é, apesar de eles serem maiores para raios maiores eles são proporcionais ao tamanho desses raios como pode ser visto na Tabela 10.

O tempo total para rodar os 150 testes desse caso, com 30 sementes para cada um dos 5 raios, foi de $136.858,50$ segundos ou aproximadamente 38 horas.

Tabela 6 – Médias da Acurácia e do tempo, em segundos, com o desvio padrão para diferentes raios.

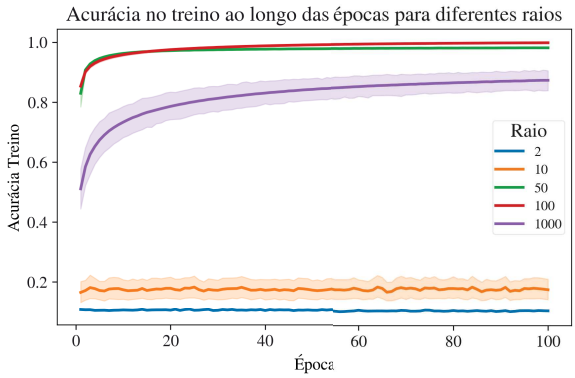
Raio	Acurácia Treino (± std)	Acurácia Teste (± std)	Tempo (± std)
2	0.103 (± 0.006)	0.102 (± 0.008)	955.47 (± 85.87)
10	0.173 (± 0.095)	0.173 (± 0.095)	851.52 (± 292.92)
50	0.981 (± 0.001)	0.958 (± 0.002)	1082.86 (± 196.96)
100	0.998 (± 0.002)	0.945 (± 0.007)	904.93 (± 215.34)
1000	0.873 (± 0.091)	0.820 (± 0.085)	767.16 (± 86.07)

Tabela 7 – Valores máximos da Acurácia e do tempo em segundos para diferentes raios.

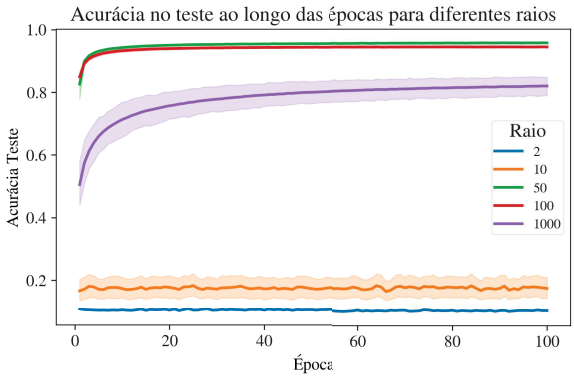
Raio	Acurácia Treino Máx	Acurácia Teste Máx	Tempo Máx
2	0.113	0.116	1133.570
10	0.389	0.382	1921.342
50	0.983	0.960	1251.527
100	1.000	0.956	1245.924
1000	0.997	0.936	899.549

Tabela 8 – Médias da função Custo e do tempo, em segundos, com o desvio padrão para diferentes raios.

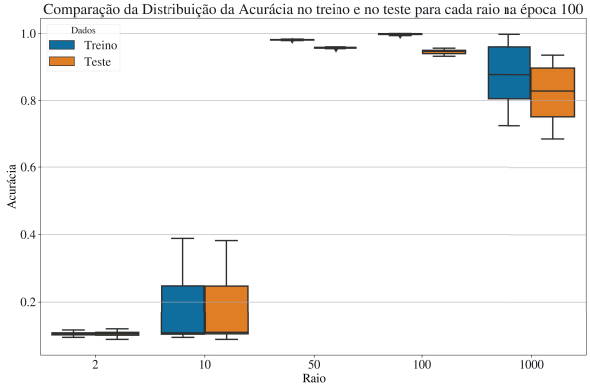
Raio	Custo Treino (± std)	Custo Teste (± std)	Tempo (± std)
2	2.308 (± 0.002)	2.308 (± 0.002)	955.474 (± 85.864)
10	2.120 (± 0.244)	2.121 (± 0.245)	851.518 (± 292.917)
50	0.089 (± 0.004)	0.152 (± 0.005)	1082.864 (± 196.958)
100	0.018 (± 0.009)	0.224 (± 0.033)	904.932 (± 215.335)
1000	0.413 (± 0.281)	0.613 (± 0.240)	767.162 (± 86.077)



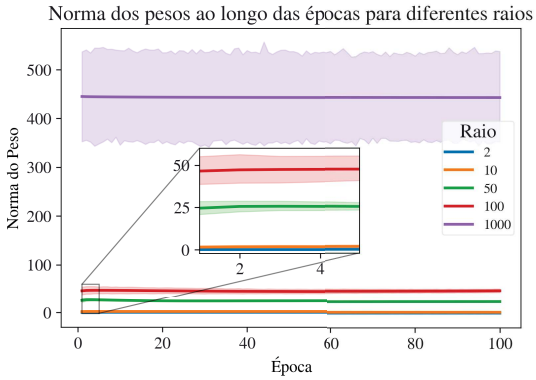
(a) Acurácia no Treino



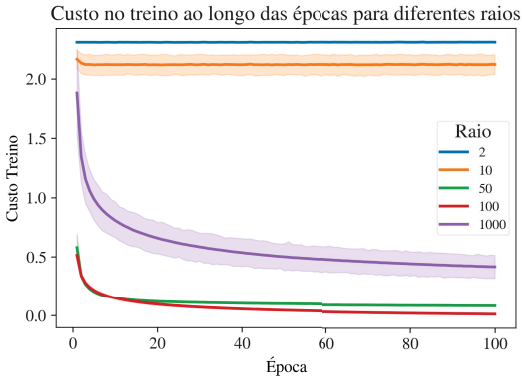
(b) Acurácia no Teste



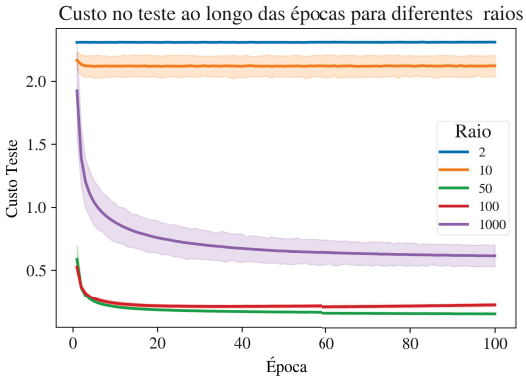
(c) Acurácia na época 100



(d) Norma dos pesos da rede



(e) Custo no Treino



(f) Custo no Teste

Figura 9 – Acurácia, custo e norma do peso para diferentes valores de raio.

Tabela 9 – Valores mínimos da função Custo e do tempo, em segundos, para diferentes raios.

Raio	Custo Treino Mín	Custo Teste Mín	Tempo Mín
2	2.304	2.303	721.276
10	1.663	1.663	706.431
50	0.082	0.139	707.712
100	0.007	0.181	702.953
1000	0.017	0.293	704.418

Tabela 10 – Norma do Peso inicial e final para diferentes raios.

Raio	Norma do peso na época 1				Norma do peso na época 100			
	Mínimo	Máximo	Média	Std	Mínimo	Máximo	Média	Std
2	0.059	0.180	0.115	0.031	0.049	0.196	0.120	0.027
10	0.153	4.912	1.574	1.867	0.163	5.108	1.829	2.049
50	7.112	42.872	24.597	10.726	23.397	24.401	23.849	0.238
100	13.544	87.725	46.513	24.126	36.896	69.426	45.838	10.149
1000	58.217	887.715	445.026	268.621	74.648	877.801	442.598	261.676

7.2.2 Variação do Tamanho do Passo

Para a variação do tamanho do passo t foram escolhidos 5 valores para serem avaliados, sendo eles: 0.01, 0.1, 0.23111951, 0.5 e 1.0. Na Figura 10 podemos ver as medidas de acurácia e custo, no treino e no teste, e a norma dos pesos ao longo das épocas para diferentes valores de t.

Com relação a métrica da acurácia, observamos na Tabela 11 que o desvio padrão da acurácia, tanto no conjunto de treino quanto no conjunto de teste, foi muito pequeno indicando que não houve grande variação entre as execuções independentes para cada valor de t. Ainda nessa tabela podemos observar que o valor de t encontrado pelo algoritmo Evolução Diferencial teve a maior média na acurácia no conjunto de teste e a segunda maior no conjunto de treino. O mesmo pode ser observado em relação ao valor máximo da acurácia obtido na Tabela 12. Todos os valores de t tiveram uma boa acurácia, no conjunto de treino e de teste, com valores acima de 0.9. Nas Figuras 10 (a) e (b) podemos ver que valores de t mais altos aumentam a acurácia mais rapidamente nas primeiras épocas do treinamento, porém, ao final das 100 épocas, os valores 0.1 e 0.23111951 obtiveram maiores valores de acurácia. No *box plot*, na Figura 10 (c), é possível observar melhor esse resultado para os valores de t iguais a 0.1 e 0.23111951 os quais tiveram valores de acurácia muito parecidos, nos conjuntos de treino e de teste, e maiores que o dos demais valores de t, destacando também o valor de $t = 0.01$ que obteve a menor acurácia e uma maior dispersão dos dados.

Em relação a norma dos pesos da rede, podemos observar na Tabela 15 que os valores de t iguais a 0.01 e 0.1 tiveram desvio padrão alto na primeira época e, na época 100, esse desvio padrão diminui pela metade para t igual a 0.01. Os demais valores de t diminuíram consideravelmente o valor do desvio padrão entre a época 1 e 100. Em relação aos valores médios não houve uma alteração muito grande comparando a época 1 e 100 em nenhum valor de t. O valor mínimo e máximo tiveram uma mudança significativa em quase todos os valores t saindo de um mínimo muito pequeno, e de um máximo muito alto e indo para valores próximos da média obtida na época 100. Isso também pode ser observado na Figura 10 (d) onde é mostrado que, no inicio, em todos os valores de t, as normas dos pesos da rede, estão próximas e, ao longo do treinamento, elas atingem

assíntotas horizontais e se estabilizam nesse valor. Nessa figura, destaca-se o valor de t igual a 0.01 que teve um desvio padrão considerável ao longo das épocas de treinamento.

Com relação a função custo, podemos ver nas Figuras 10 (e) e (f) um comportamento semelhante ao que ocorreu na acurácia. Os valores maiores de t diminuíram o custo mais rapidamente. Os valores de t igual a 0.1 e 0.23111951 obtiveram os menores valores para a função custo. Nas Tabelas 13 e 14 observa-se que todos os valores de t tiveram um valor da função custo baixo, algo já esperado pelo que foi visto com a acurácia, sendo o valor de t igual a 0.23111951 o que obteve o menor custo médio no teste e o segundo menor no conjunto de treino. Observa-se também, um desvio padrão muito baixo em todos os casos e um tempo médio muito próximo entre os diferentes valores de t.

O tempo total para rodar os 150 testes, 30 sementes para cada um dos 5 valores de t, foi de 131.723 segundos, ou aproximadamente $36,5$ horas.

Tabela 11 – Médias da Acurácia e do tempo, em segundos, com o desvio padrão para diferentes valores de t.

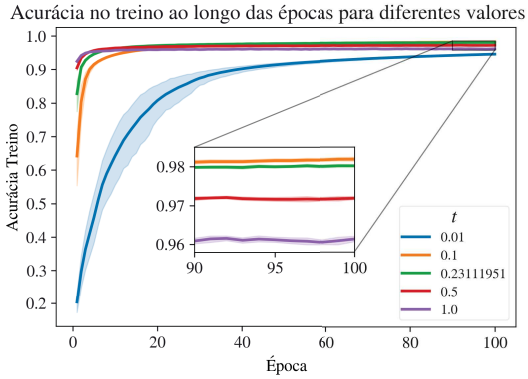
t	Acurácia Treino (± std)	Acurácia Teste (± std)	Tempo (± std)
0.010	0.945 (± 0.005)	0.930 (± 0.005)	1064.191 (± 290.034)
0.100	0.982 (± 0.001)	0.956 (± 0.002)	955.705 (± 155.352)
0.231	0.980 (± 0.001)	0.957 (± 0.002)	820.880 (± 303.790)
0.500	0.972 (± 0.001)	0.954 (± 0.002)	833.399 (± 74.778)
1.000	0.961 (± 0.002)	0.947 (± 0.003)	716.599 (± 3.809)

Tabela 12 – Valores máximos da Acurácia e do tempo em segundos para diferentes valores de t.

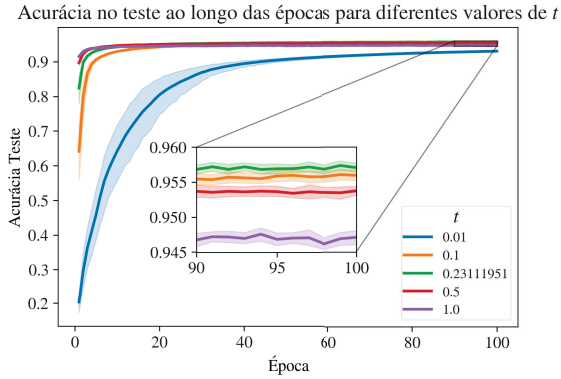
t	Acurácia Treino Máx	Acurácia Teste Máx	Tempo Máx
0.010	0.952	0.939	2413.164
0.100	0.984	0.959	1340.738
0.231	0.983	0.960	2015.913
0.500	0.974	0.957	1012.071
1.000	0.966	0.952	728.240

Tabela 13 – Médias da função Custo e do tempo, em segundos, com o desvio padrão para diferentes valores de t.

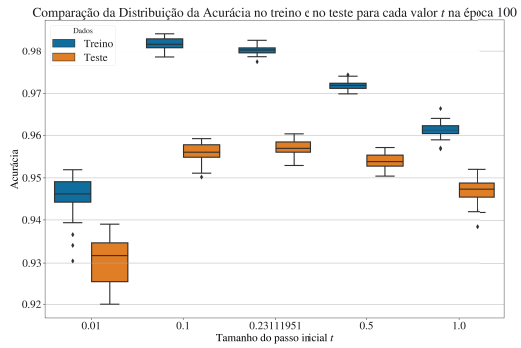
t	Custo Treino (± std)	Custo Teste (± std)	Tempo (± std)
0.010	0.208 (± 0.020)	0.250 (± 0.018)	1064.191 (± 290.034)
0.100	0.088 (± 0.005)	0.156 (± 0.006)	955.705 (± 155.352)
0.231	0.092 (± 0.003)	0.154 (± 0.005)	820.880 (± 303.790)
0.500	0.116 (± 0.003)	0.167 (± 0.005)	833.399 (± 74.778)
1.000	0.145 (± 0.006)	0.188 (± 0.007)	716.599 (± 3.809)



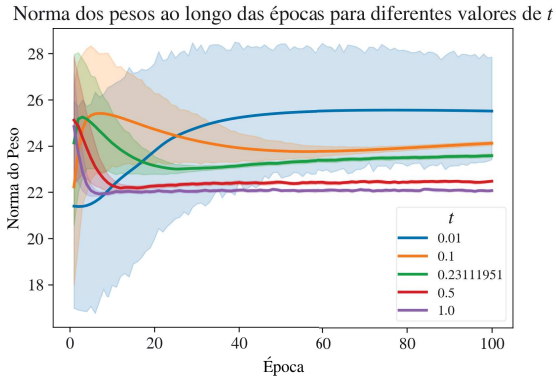
(a) Acurácia no Treino



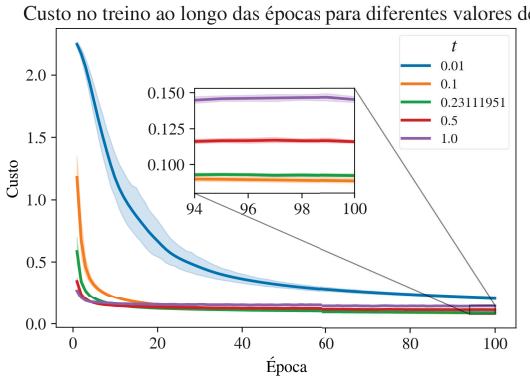
(b) Acurácia no Teste



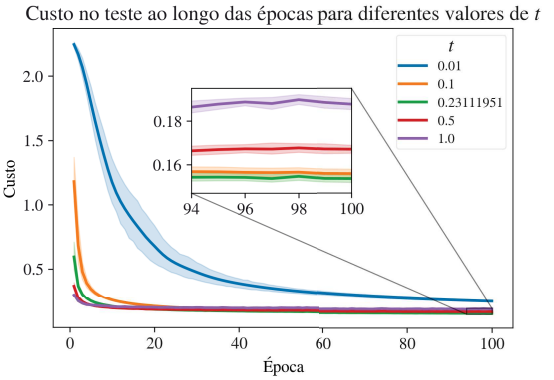
(c) Acurácia na época 100



(d) Norma dos pesos da rede



(e) Custo no Treino



(f) Custo no Teste

Figura 10 – Acurácia, custo e norma do peso para diferentes valores de raio.

Tabela 14 – Valores mínimos da função Custo e do tempo, em segundos, para diferentes valores de t.

t	Custo Treino Mín	Custo Teste Mín	Tempo Mín
0.010	0.185	0.225	779.797
0.100	0.081	0.141	768.045
0.231	0.086	0.142	720.413
0.500	0.109	0.154	718.776
1.000	0.131	0.174	711.920

Tabela 15 – Norma do Peso inicial e final para diferentes valores de t.

t	Norma do peso na época 1				Norma do peso na época 100			
	Mínimo	Máximo	Média	Std	Mínimo	Máximo	Média	Std
0.010	2.709	42.732	21.398	12.948	18.573	38.613	25.507	6.543
0.100	2.665	42.465	22.233	12.502	23.554	24.607	24.113	0.258
0.231	6.774	41.954	24.132	10.487	23.056	23.995	23.576	0.220
0.500	14.822	40.589	25.122	8.194	22.220	22.845	22.469	0.165
1.000	18.137	37.930	24.844	6.323	21.660	22.449	22.061	0.172

7.2.3 Variação do Número de Iterações

Para a variação do número de iterações foram escolhidos 5 valores para serem avaliados, sendo eles: 1, 2, 5, 7 e 10. Na Figura 11 são apresentados gráficos da acurácia e da função custo, no conjunto de treino e de teste, e a norma dos pesos da rede ao longo das épocas considerando esses diferentes números de iterações no treinamento.

Tabela 16 – Médias da Acurácia e do tempo, em segundos, com o desvio padrão para diferentes números de Iterações.

Iterações	Acurácia Treino (± std)	Acurácia Teste (± std)	Tempo (± std)
1	0.964 (± 0.001)	0.949 (± 0.002)	672.535(± 172.209)
2	0.980 (± 0.001)	0.957 (± 0.002)	820.880(± 303.790)
5	0.985 (± 0.002)	0.957 (± 0.002)	2026.218(± 1090.844)
7	0.985 (± 0.002)	0.956 (± 0.002)	2120.654(± 696.625)
10	0.983 (± 0.002)	0.955 (± 0.002)	2641.864(± 10.525)

Tabela 17 – Valores máximos da Acurácia e do tempo em segundos para diferentes números de Iterações.

Iterações	Acurácia Treino Máximo	Acurácia Teste Máximo	Tempo Máximo
1	0.966	0.953	1062.489
2	0.983	0.960	2015.913
5	0.988	0.961	4732.276
7	0.988	0.961	5433.911
10	0.987	0.960	2660.893

Para a medida da acurácia, podemos ver nas Figuras 11 (a) e (b) que números de iteração mais altos chegam mais rápido em sua assíntota horizontal mas não necessariamente chegam em valores mais altos. Isso pode ser visto nas Tabelas 16 e 17 que mostram que o valor de 5 iterações foi o que se saiu melhor na média e no máximo da acurácia tanto no conjunto de treino quanto no conjunto de teste. Porém, analisando os resultados, vemos que as acurácias obtidas, tanto médias quanto máximas, não estão muito longe uma das outras, e nesse caso poderíamos utilizar o tempo gasto como um desempate para escolher o melhor hiperparâmetro. Com isso, vemos que o número de iterações igual a 2 tem

resultados muito bons de acurácia, destacando a acurácia média no teste que empata como a melhor acurácia no teste, e que possui o segundo menor tempo médio de treinamento e segundo menor tempo máximo. O desvio padrão de todos os casos considerados é muito pequeno indicando pouca variabilidade entre as execuções independentes. Na Figura 11 (c), é possível observar a distribuição da acurácia ao final do treinamento para cada número de iterações. Todos tiveram desempenhos parecidos de acurácia final no conjunto de treino e de teste e apresentam um baixo desvio padrão. Destaca-se ai, o modelo com número de iterações igual a 1 que obteve o pior resultado, se comparado aos outros, mas basta olhar a escala para perceber que apesar disso ele obteve bons valores de acurácia.

Tabela 18 – Médias da função Custo e do tempo, em segundos, com o desvio padrão para diferentes números de Iterações.

Iterações	Custo Treino (± std)	Custo Teste (± std)	Tempo (± std)
1	0.164 (± 0.003)	0.201 (± 0.005)	672.535 (± 172.209)
2	0.092 (± 0.003)	0.154 (± 0.005)	820.880 (± 303.790)
5	0.059 (± 0.005)	0.146 (± 0.005)	2026.218 (± 1090.844)
7	0.057 (± 0.005)	0.149 (± 0.007)	2120.654 (± 696.625)
10	0.057 (± 0.005)	0.154 (± 0.007)	2641.864 (± 10.525)

Tabela 19 – Valores mínimos da função de Custo e do tempo, em segundos, para diferentes números de Iterações.

Iterações	Custo Treino	Custo Teste	Tempo
1	0.158	0.190	480.946
2	0.086	0.142	720.413
5	0.050	0.137	1448.147
7	0.048	0.134	1897.821
10	0.049	0.143	2618.845

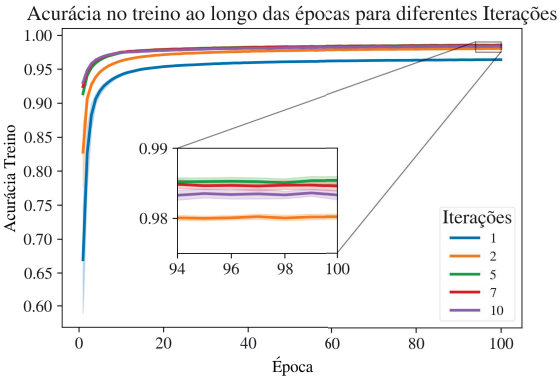
Tabela 20 – Norma do Peso inicial e final para diferentes números de Iterações.

Iterações	Norma do peso na Época 1				Norma do peso na Época 100			
	Mínimo	Máximo	Média	Std	Mínimo	Máximo	Média	Std
1	2.593	41.549	21.701	12.019	19.089	19.507	19.281	0.115
2	6.774	41.954	24.132	10.487	23.056	23.995	23.576	0.220
5	17.023	43.430	27.941	8.533	27.068	28.787	27.813	0.401
7	19.544	44.122	29.314	8.047	27.955	29.821	28.665	0.387
10	21.797	44.922	30.767	7.612	28.606	30.565	29.452	0.418

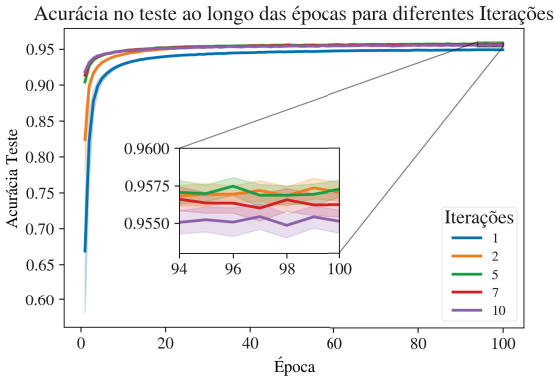
Na Figura 11 (d), observamos que nas primeiras 20 épocas, todos os modelos apresentam variância na norma dos pesos da rede. Além disso, é possível ver que quanto maior o número de iterações maior é a assíntota horizontal alcançada em todos os casos. Na tabela 20 observa-se que na época 100, a norma dos pesos aumenta em média, em

valor máximo e em valor mínimo quanto maior for o número de iterações. Vemos também na tabela que, o desvio padrão que, na época 1, é alto, ao final do treinamento, na época 100, é baixo.

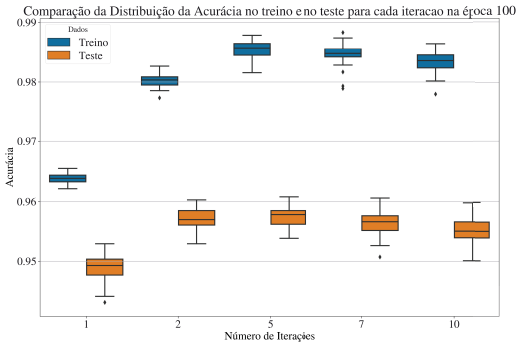
Em relação à função custo, as Figuras 11 (e) e (f) mostram que números de iterações mais altos diminuem mais rapidamente o custo ao longo do treinamento. Nas Tabelas 18 e 19 mostram que todos os casos se saíram bem, com custo médio e mínimo no treino e teste abaixo de 0.21. O modelo com número de iterações igual a 5 obteve o menor custo médio igual a 0.146 no conjunto de teste, o segundo menor custo médio igual a 0.059 no conjunto de treino, e ficou entre os 3 menores custos mínimos no conjunto de treino e de



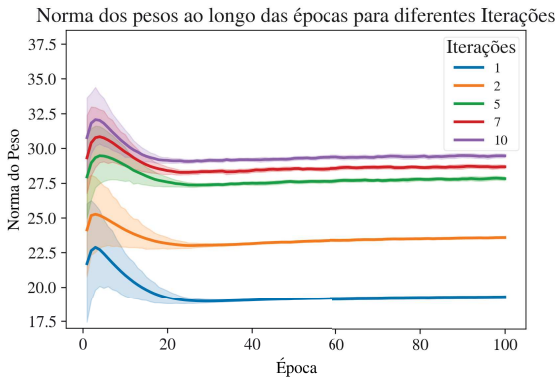
(a) Acurácia no Treino



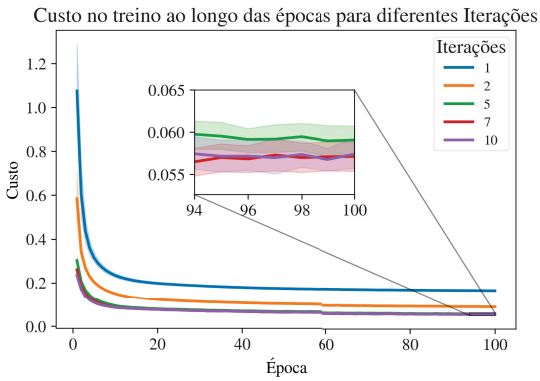
(b) Acurácia no Teste



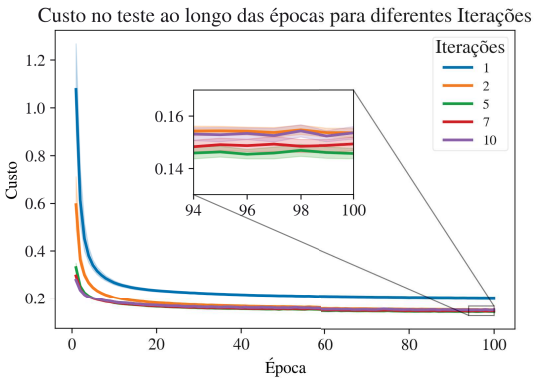
(c) Acurácia na época 100



(d) Norma dos pesos da rede



(e) Custo no Treino



(f) Custo no Teste

Figura 11 – Acurácia, custo e norma do peso para diferentes valores de raio.

teste. Todos os resultados tiveram um desvio padrão abaixo de 0.01 indicando uma baixa variação entre as execuções independentes. Apesar de alguns modelos terem obtido um custo menor em alguns casos, analisando de uma forma geral, todos se saíram muito bem com um valor da função custo muito baixo e novamente, poderíamos desempatar uma suposta escolha do melhor modelo olhando para o tempo médio gasto. Com isso, o modelo com número de iterações igual a 2 é o que tem um melhor equilíbrio entre custo médio e mínimo e tempo médio gasto.

O tempo total para rodar os 150 testes, 30 sementes para cada um dos 5 número de iterações, foi de $248.464,503$ segundos, ou aproximadamente 69 horas.

8 CONCLUSÃO

Nesse trabalho, foi feita uma modificação do algoritmo FDIPA com a finalidade de realização do treinamento de redes neurais artificiais. Foram estudadas técnicas utilizadas em treinamento de redes neurais com especial atenção para o *backpropagation* que é uma abordagem bastante empregada para o cálculo do gradiente da função custo. Para a realização desse trabalho, implementamos um algoritmo para definição de redes neurais, o algoritmo de treinamento SGD, o algoritmo *backpropagation* e o algoritmo FDIPA, incluindo as adaptações que se fizeram necessárias. Após todas as implementações, realizamos testes com diferentes cenários, conforme ilustrado no Capítulo 7.

Com a obtenção dos valores ótimos de hiperparâmetros para o SGD e para o FDIPA observamos que o FDIPA teve uma acurácia maior que o SGD no conjunto de treino e de teste. O FDIPA obteve 0.98 de acurácia média no conjunto de treino e 0.957 no conjunto de teste enquanto que o SGD teve 0.926 de acurácia média no conjunto de treino e 0.889 no conjunto de teste. É importante ressaltar que o SGD obteve resultados melhores com o valor da taxa de aprendizagem $\eta = 1$. Com esse valor de η, o SGD obteve uma acurácia média no conjunto de treino de 0.981, valor próximo ao obtido pelo FDIPA, e de 0.907 no conjunto de teste, abaixo do obtido pelo FDIPA. Em termos de comparação, nos nossos testes o FDIPA obteve resultados melhores na métrica de acurácia tanto no conjunto de treino quanto no conjunto de teste se comparado ao SGD por nós implementado. Apesar disso, cabe ressaltar que o tempo médio do SGD foi próximo de 196 segundos, um pouco mais do que 3 minutos, enquanto que o FDIPA teve um tempo médio próximo de 900 segundos, ou 15 minutos. Isso nos mostra que o FDIPA ainda está muito lento para esses problemas.

Os resultados obtidos pelo FDIPA em si, foram satisfatórios uma vez que obteve-se uma acurácia alta para o problema considerado. Além disso, os resultados foram obtidos utilizando no algoritmo a matriz identidade como aproximação para a matriz hessiana B. Com isso, vemos como muito positivo os resultados uma vez que foi utilizada a mais simples matriz como aproximação da matriz hessiana. Espera-se que os resultados sejam ainda melhores com uma escolha mais adequada para B. Restrições do tipo caixas ($x \in [a_1, b_1] \times ... \times [a_n, b_n]$) ou anelares ($r_0 \leq \frac{\|x\|^2}{2} \leq r_1$) também podem ser consideradas.

Para trabalhos futuros, pretende-se testar melhores escolhas para a matriz B, por exemplo utilizando a técnica BFGS para a sua atualização. Por se tratar de problemas de redes neurais, a dimensão da matriz B pode ficar extremamente grande e, portanto, trazer problemas tanto em relação ao armazenamento quanto à complexidade computacional dos cálculos. Futuramente, para realização dos cálculos, podem ser utilizados as Unidades de Processamentos de Tensor [36] (TPU, sigla em inglês para Tensor Processing Unit) que tratam-se de aceleradores de hardware especializados em realizar operações de matrizes e

tensores de grandes dimensões. Essa abordagem pode aumentar a acurácia e velocidade de treinamento visto que as implementações feitas nesse trabalho utilizaram CPU e as TPU's são cerca de 20 vezes mais rápidas que as GPU's [36].

REFERÊNCIAS

1 MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, Manole, v. 1, n. 1, p. 32, 2003.

2 DSA, E. *Deep Learning Book*. Disponível em: <https://www.deeplearningbook.com.br/>. Acesso em: 11 de novembro de 2020.

3 CURRY, H. B. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, v. 2, n. 3, p. 258–261, 1944.

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

5 HERSKOVITS, J. A feasible directions interior point technique for nonlinear optimization1. 1998.

6 GRIGORESCU, S.; TRASNEA, B.; COCIAS, T.; MACESANU, G. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, Wiley Online Library, v. 37, n. 3, p. 362–386, 2020.

7 ESTEVA, A.; ROBICQUET, A.; RAMSUNDAR, B.; KULESHOV, V.; DEPRISTO, M.; CHOU, K.; CUI, C.; CORRADO, G.; THRUN, S.; DEAN, J. A guide to deep learning in healthcare. *Nature medicine*, Nature Publishing Group, v. 25, n. 1, p. 24–29, 2019.

8 HEATON, J. B.; POLSON, N. G.; WITTE, J. H. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, Wiley Online Library, v. 33, n. 1, p. 3–12, 2017.

9 GUPTA, J. N.; SEXTON, R. S. Comparing backpropagation with a genetic algorithm for neural network training. *Omega*, Elsevier, v. 27, n. 6, p. 679–684, 1999.

10 ÖRKCÜ, H. H.; BAL, H. Comparing performances of backpropagation and genetic algorithms in the data classification. *Expert systems with applications*, Elsevier, v. 38, n. 4, p. 3703–3709, 2011.

11 GUDISE, V. G.; VENAYAGAMOORTHY, G. K. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In: IEEE. *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*. [S.l.], 2003. p. 110–117.

12 RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

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

14 NESTEROV, Y. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In: *Doklady an ussr*. [S.l.: s.n.], 1983. v. 269, p. 543–547.

15 DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, v. 12, n. 7, 2011.

16 ZEILER, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

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

18 HAYKIN, S. *Redes neurais: princípios e prática*. [S.l.]: Bookman Editora, 2007.

19 BENGIO, Y. *Learning deep architectures for AI*. [S.l.]: Now Publishers Inc, 2009.

20 KARLIK, B.; OLGAC, A. V. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, Citeseer, v. 1, n. 4, p. 111–122, 2011.

21 JAYNES, E. T. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, IEEE, v. 70, n. 9, p. 939–952, 1982.

22 JAYNES, E. T. Information theory and statistical mechanics. *Physical review*, APS, v. 106, n. 4, p. 620, 1957.

23 KULLBACK, S. *Information theory and statistics*. [S.l.]: Courier Corporation, 1997.

24 KETKAR, N. Stochastic gradient descent. In: *Deep learning with Python*. [S.l.]: Springer, 2017. p. 113–132.

25 GARDNER, W. A. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal processing*, Elsevier, v. 6, n. 2, p. 113–133, 1984.

26 HALLACK, A. A. Análise iii (análise no ir n). 2008.

27 SHERMAN, J.; MORRISON, W. J. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, JSTOR, v. 21, n. 1, p. 124–127, 1950.

28 HAGER, W. W. Updating the inverse of a matrix. *SIAM review*, SIAM, v. 31, n. 2, p. 221–239, 1989.

29 BARTLETT, M. S. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, JSTOR, v. 22, n. 1, p. 107–111, 1951.

30 KUHN, H. W.; TUCKER, A. W. Nonlinear programming. In: *Traces and emergence of nonlinear programming*. [S.l.]: Springer, 2014. p. 247–258.

31 BROYDEN, C. G. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, Oxford University Press, v. 6, n. 1, p. 76–90, 1970.

32 LV, J.; DENG, S.; WAN, Z. An efficient single-parameter scaling memoryless broyden-fletcher-goldfarb-shanno algorithm for solving large scale unconstrained optimization problems. *IEEE Access*, IEEE, v. 8, p. 85664–85674, 2020.

33 HERSKOVITS, J. A view on nonlinear optimization. In: *Advances in structural optimization*. [S.l.]: Springer, 1995. p. 71–116.

34 LECUN, Y.; CORTES, C. *MNIST handwritten digit database*. Disponível em: <http://yann.lecun.com/exdb/mnist>. Acesso em: 29 de março de 2021.

35 STORN, R.; PRICE, K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, Springer, v. 11, n. 4, p. 341–359, 1997.

36 IBRAHIM, M. *What is a Tensor Processing Unit (TPU) and how does it work?* Disponível em: <https://towardsdatascience.com/what-is-a-tensor-processing-unit-tpu-and-how-does-it-work-dbbe6ecbd8ad>. Acesso em: 11 de novembro de 2021.