UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

LEANDRO DE JESUS FRUTUOSO

Reconhecimento de emoções através de Redes Neurais

São Carlos
2021

LEANDRO DE JESUS FRUTUOSO

Reconhecimento de emoções através de Redes Neurais

Monografia apresentada ao Curso de Engenharia Mecânica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Mecânico.

Orientadora: Profa. Dra. Maíra Martins da Silva

VERSÃO ORIGINAL

São Carlos

2021

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

F944r

Frutuoso, Leandro
Reconhecimento de emoções através de Redes Neurais / Leandro Frutuoso; orientadora Maira Martins da Silva. São Carlos, 2021.

Monografia (Graduação em Engenharia Mecânica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2021.

1. Inteligência Artificial. 2. Redes Neurais. 3. Reconhecimento de emoções. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE AVALIAÇÃO

Candidato: Leandro de Jesus Frutuoso

Título: Reconhecimento de emoções através de Redes Neurais

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo Curso de Engenharia Mecânica.

BANCA EXAMINADORA

Profa. Dra. Maíra Martins da Silva
(Orientador)

Nota atribuída: 9,5 (nove, cinco) Maíra M. da Silva
(assinatura)

Prof. Tit. Glauco Augusto de Paula Caurin p/ Maíra M. da Silva
(assinatura)

Nota atribuída: 9,5 (nove, cinco)

Dr. Guilherme Serpa Sestito p/ Maíra M. da Silva
(assinatura)

Nota atribuída: 9,5 (nove, cinco)

Média: 9,5 (nove, cinco)

Resultado: aprovado

Data: 09/07/2021.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☑ NÃO ☐ Visto do orientador Maíra M. da Silva

AGRADECIMENTOS

À minha família. Meus pais, Viviane e Marcelo, meu irmão, Lucas, por todo o suporte e amor dado ao longo de toda minha vida.

À Cynthia, por todos os momentos de ajuda, amor e companheirismo.

Aos meus amigos, por tornarem minha vida mais feliz e sempre estenderem a mão em momentos difíceis.

À profa. Maíra, por me encorajar, orientar e auxiliar neste trabalho.

RESUMO

FRUTUOSO, L. J. **Reconhecimento de emoções através de Redes Neurais.** 2021. 64 f. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2021.

O uso da inteligência artificial é amplamente difundido nas mais diversas aplicações tecnológicas atuais. Numa sociedade onde a interação entre os seres humanos e as máquinas ocorre de maneira cada vez mais profunda, o entendimento de características humanas se torna fundamental para as máquinas. A fim de tornar mais eficaz esta interação, o reconhecimento de emoções pode ser visto como um dos pilares que constitui as informações necessárias para avaliar o estado mental de uma pessoa. O presente trabalho tem por objetivo treinar uma rede neural capaz de identificar emoções humanas, utilizando como base de dados o conjunto de imagens FER-2013. Na revisão bibliográfica são abordados conceitos históricos sobre o surgimento das redes neurais, sua estrutura, seu funcionamento e problemas comuns. Para a obtenção dos dados utilizados neste trabalho foram treinadas 66 redes, com hiperparâmetros que se demonstraram promissores, por evidência empírica ou por resultados de outros autores. Os conceitos apresentados foram então utilizados no embasamento dos resultados e na escolha da melhor rede dentre todas as treinadas, que apresentou 47.17% de acurácia média no conjunto de testes. Apesar de não ser uma acurácia satisfatória para aplicações práticas, ela foi obtida através de ajustes dos hiperparâmetros demonstrando, claramente, a influência deles sobre a performance das redes neurais.

Palavras chave: Inteligência Artificial, Redes Neurais, Reconhecimento de emoções.

ABSTRACT

FRUTUOSO, L. J. **Emotion recognition through the use of Neural Networks.** 2021. 64 f. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2021.

The use of artificial intelligence is widespread in the most diverse technological applications nowadays. In a society where the interaction between human beings and machines occurs in increasingly deep ways, the comprehension of human characteristics becomes key for the machines. In order to make this interaction more efficient, emotion recognition can be seen as one of the cornerstones that compose the necessary information for evaluating the mental state of a person. This work has the objective of training a neural network capable of identifying human emotions, using as the basis the dataset of images FER-2013. In the literature review, concepts of the history about the arise of the neural networks, their structure, how they work and commun problems revolving the subject are approached. To obtain the data used in this work 66 nets were trained with hyperparameters that appeared to be promising, by empirical evidence or by results of other authors. Then, the concepts presented were used in the foundation about the explanation of the results and in the process of choosing the best net among all the trained ones, which showed an average accuracy of 47.17% in the test dataset. Even though it's not a satisfactory accuracy for practical application, it was obtained through the tunning of the hyperparameters showing, clearly, their influence on the performance of the neural nets.

Keywords: Artificial Intelligence, Neural Networks, Emotion Recognition.

LISTA DE ILUSTRAÇÕES

Figura 1 - Taxas de erro dos vencedores do desafio ImageNet entre 2010 e 2017.....6
Figura 2 - Diagrama dos campos de estudo abordados.....9
Figura 3 - Estrutura de um perceptron com 3 entradas.....10
Figura 4 - Estrutura de uma rede neural completamente conectada.....12
Figura 5 - Sigmoid vs Step Function.....13
Figura 6 - Configuração esquemática de feedforward para um neurônio.....14
Figura 7 - Representação da derivada parcial do custo em relação a 19
Figura 8- Feedforward em uma camada dividida em dense layer e activation layer.....19
Figura 9 - Representação do feedforward e backpropagation em uma camada subdividida. 20
Figura 10 - Representação da Dense Layer.....20
Figura 11 - Representação da Activation Layer.....24
Figura 12 - Não convergência por um learning rate com valor alto.....28
Figura 13 - Convergência com learning rate menor.....28
Figura 14 - Representação de uma convergência otimizada.....29
Figura 15 - Exemplos de emoções da base de dados.....35
Figura 16 - Gráfico test accuracy vs epochs 1ª Rede, primeira leva.....41
Figura 17 - Gráfico test accuracy vs epochs 2ª Rede, primeira leva.....42
Figura 18 - Gráfico test accuracy vs epochs 3ª Rede, primeira leva.....42
Figura 19 - Gráfico test accuracy vs epochs 1ª Rede, segunda leva.....44
Figura 20 - Gráfico test accuracy vs epochs 2ª Rede, segunda leva.....45
Figura 21 - Gráfico test accuracy vs epochs 3ª Rede, segunda leva.....45
Figura 22 - Gráfico training cost vs epochs 1ª Rede, primeira leva.....47
Figura 23 - Gráfico training cost vs epochs 2ª Rede, primeira leva.....47
Figura 24 - Gráfico training cost vs epochs 2ª Rede, segunda leva.....48
Figura 25 - Gráfico test accuracy vs epochs 1ª Rede, segunda leva, mini batch size 16.....49
Figura 26 - Gráfico test accuracy vs epochs 1ª Rede, segunda leva, mini batch size 32.....49
Figura 27 - Gráfico test accuracy vs epochs 1ª Rede, segunda leva, mini batch size 64.....50
Figura 28 - Gráfico training accuracy vs epochs 1ª Rede, segunda leva.....51
Figura 29 - Gráfico training accuracy vs epochs 2ª Rede, segunda leva.....51

LISTA DE TABELAS

Tabela 1 - Quantidade de imagens por emoção no conjunto de Treino.....34
Tabela 2 - Quantidade de imagens por emoção no conjunto de Teste.....35
Tabela 3 - Emoções e seus respectivos valores de encoding.....36
Tabela 4 - Hiperparâmetros testados.....37
Tabela 5 - Hiperparâmetros refinados.....38
Tabela 6 - Hiperparâmetros primeira rede de treino.....39
Tabela 7 - Hiperparâmetros das 30 redes treinadas na primeira leva.....40
Tabela 8 - Hiperparâmetros 3 redes com maior acurácia média da primeira leva.....40
Tabela 9 - Média e desvio padrão das 3 redes com maior acurácia média da primeira leva.
.....42
Tabela 10 - Hiperparâmetros das 3 redes com maior acurácia média na segunda leva.....43
Tabela 11 - Média e desvio padrão das 3 redes com maior acurácia média da segunda leva.
.....43

LISTA DE ABREVIATURAS E SIGLAS

IA	*Inteligência Artificial*
ML	*Machine Learning*
NN	*Neural Network*
EQM	*Erro quadrático médio.*
CNN	*Convolutional Neural Network*

SUMÁRIO

1 INTRODUÇÃO **5**
1.1 Apresentação do tema **5**
1.2 Objetivos 7
2 REVISÃO BIBLIOGRÁFICA **8**
2.1 Emoções 8
2.2 Imagens 8
2.3 Inteligência Artificial, Machine Learning, Neural Networks e Deep Learning 9
2.4 Tipos de aprendizado 10
2.5 Estrutura das Redes Neurais 10
2.6 Funcionamento das Redes Neurais 12
2.6.1 Passagem de informação: forward propagation ou feedforward 12
2.6.2 Função de custo, perda ou objetiva 14
2.6.3 Gradient descent. 15
2.6.4 Convenções adotadas 17
2.6.5 Backpropagation 19
2.7 Hiperparâmetros, otimizações e boas práticas 27
2.7.1 Learning rate 28
2.7.2 Epochs 30
2.7.3 Stochastic gradient descent e mini batch size 30
2.7.4 Valores iniciais dos parâmetros de aprendizado 30
2.7.5 Número de neurônios e camadas 31
2.8 Problemas comuns das redes neurais 31
2.8.1 Saturação do neurônio 31
2.8.2 Overfitting e underfitting 33
2.8.3 Vanishing Gradient 33
2.9 Categorical Encoding 33
3 METODOLOGIA **35**
3.1 Base de dados 35
3.2 Linguagem e Ambiente de execução utilizados 36
3.3 Código utilizado 36
3.4 Tratamento dos dados 36
3.4.1 Encoding dos dados 36
3.4.2 Balanceamento dos dados 37
3.5 Treinamento das redes 37
3.5.1 Parâmetros iniciais 37
3.5.2 Teste dos parâmetros 38
3.5.3 Refinamento dos parâmetros 38
3.5.4 Avaliação de performance 39
4 RESULTADOS **40**

4.1 Redes iniciais 40
4.2 Busca por parâmetros 40
4.2.1 Primeira leva de treinamento 40
4.2.1 Segunda leva de treinamento 44
4.3 Análise dos hiperparâmetros com melhor performance 46
4.3.1 Learning rate 46
4.3.2 Mini batch size 49
4.3.3 Epochs e overfitting 51
5 CONCLUSÃO 53
6 REFERÊNCIAS 54

1 INTRODUÇÃO

1.1 Apresentação do tema

A visão é uma ferramenta evolutiva adquirida, ao longo de milhões de anos, pela maioria das espécies de animais que habitam o planeta. Dentre os sistemas sensoriais, a visão é, sem dúvida, um dos sentidos mais importantes para que o animal possa sobreviver no meio em que habita. Isso ocorre porque suas chances de sobrevivência estão diretamente atreladas à quantidade de informações que ele consegue absorver e interpretar sobre o ambiente (JONES, 2014).

Apesar da sobrevivência humana frente às ameaças naturais do ambiente ser um desafio passado, a informação obtida pelos sistemas sensoriais ainda possui extrema relevância e não apenas para nós. As máquinas necessitam extrair e interpretar informações sobre o ambiente de maneiras cada vez mais sofisticadas para realizarem tarefas cada vez mais complexas. Para tal, o campo de estudo da visão computacional é essencial.

Visão computacional pode ser definida como um campo científico que extrai informações de imagens digitais (MOINDROT, 2018). Os recentes estudos nesse campo possibilitaram a construção de diversas aplicações como, por exemplo, sistemas de reconhecimento facial, análise médica de imagens, carros autônomos etc.

Essas aplicações só se tornaram factíveis através do uso da *Inteligência Artificial*. Que, de acordo com o próprio criador do termo, é definida como: “A ciência e engenharia de produzir máquinas inteligentes, especialmente, programas inteligentes. Ela está relacionada com a tarefa similar de usar computadores para entender a inteligência humana, porém a Inteligência Artificial não precisa se confinar a métodos que são biologicamente observáveis.” (MCCARTHY, 2004).

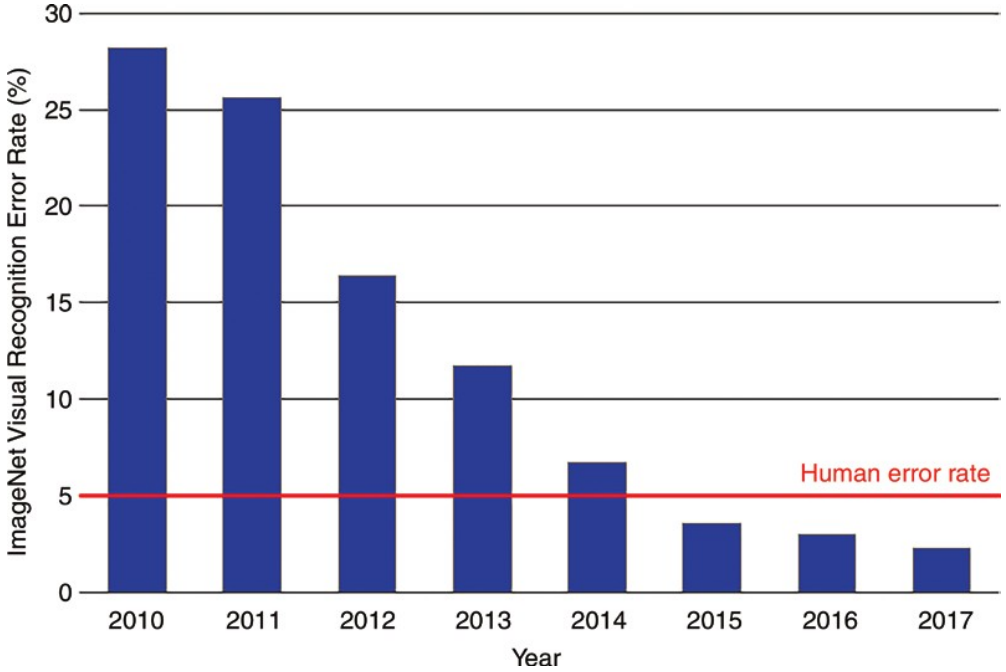
Os campos da Inteligência Artificial e Visão Computacional existem desde os anos 1950 e 1960, respectivamente (MOINDROT, 2018 e ANYOHA, 2017). A Inteligência Artificial avançou de forma significativa entre 1957 e 1974, juntamente com os avanços tecnológicos na capacidade de processamento e de memória dos computadores. Após esse período, os avanços não atingiram as expectativas até 1997, quando a inteligência artificial *Deep Blue* da IBM derrotou o campeão mundial de xadrez *Gary Kasparov* (ANYOHA, 2017).

A técnica mais popular da atualidade utilizada no reconhecimento de imagens, *deep learning*, surgiu na década de 1980, quando o termo foi introduzido por *Rina Dechter* (DECHTER, 1986). Porém, uma barreira limitou o avanço desse campo de estudo, a falta de imagens disponíveis e devidamente classificadas para se trabalhar com as redes neurais.

Foi apenas entre 2009 e 2010 que ocorreu o marco divisor de águas no campo da visão computacional. Com o avanço e popularização de aparelhos como smartphones e câmeras digitais, o projeto *ImageNet* (FEI-FEI et al, 2009) atacou o problema da falta de dados para o campo de estudo. Classificando e disponibilizando, em 2010, mais de 10 milhões de imagens contendo mais de 10 mil classes diferentes de objetos.

O projeto consiste não apenas na classificação destes dados, mas também em um desafio onde os participantes criam algoritmos com o objetivo de classificar automaticamente as imagens. Um fato relevante ocorreu em 2012 com o artigo publicado por *Alex Krizhevsky* que abordou o desafio *ImageNet* utilizando redes neurais convolucionais (KRIZHEVSKY et al, 2012), que se mostrou uma abordagem muito mais promissora comparada às anteriores e se tornou a mais utilizada até o presente momento. Nota-se um grande avanço na área em um curto espaço de tempo. Em 2010 a taxa de erro na classificação das imagens do algoritmo vencedor do desafio era de 28% e em 2017 o erro foi menor que 3%, conforme mostra a Figura 1.

Figura 1 - Taxas de erro dos vencedores do desafio ImageNet entre 2010 e 2017.



Fonte: Langlotz (2018)

Desde então houve uma popularização na utilização de técnicas de inteligência artificial, não só no meio acadêmico, como também no meio industrial, financeiro, tecnológico e até mesmo no âmbito pessoal com bibliotecas de uso livre, como *Tensorflow* e *Keras* produzidas pelo *Google LCC.*

As pessoas estão se tornando cada vez mais dependentes destas aplicações, gerando questões sobre como será um futuro não tão distante. Segundo Zuffo (2020), a relação do ser humano com as máquinas pode se tornar até mesmo simbiótica, que de certa forma já é realidade em algumas aplicações médicas como o marcapasso, por exemplo.

Num horizonte próximo, possivelmente, será dado um passo além. Tecnologias poderão viabilizar a comunicação direta com as máquinas através da atividade cerebral, como demonstrado na apresentação da *Neuralink* onde um macaco jogou o videogame *Pong* utilizando um chip implantado diretamente no seu cérebro (WAKEFIELD, 2021).

Tendo uma proximidade tão grande com as máquinas, é importante ter em mente a quantidade de informação que elas conseguirão obter sobre os seres humanos, incluindo suas emoções.

1.2 Objetivos

O objetivo deste trabalho é implementar um algoritmo de rede neural com aprendizado profundo, para o reconhecimento de emoções humanas através de expressões faciais obtidas por imagens classificadas em 5 emoções básicas: Neutralidade, tristeza, felicidade, medo e raiva.

Esse objetivo será abordado através das seguintes etapas:

1. Implementação e treinamento de uma rede neural utilizando a base de dados de imagens FER-2013 (KAGGLE, 2020).
2. Teste de acurácia através de experimentação com diferentes hiperparâmetros.

2 REVISÃO BIBLIOGRÁFICA

2.1 Emoções

Segundo Izard (2010), a comunidade científica ainda não possui um consenso para a definição de emoção. Porém, uma descrição consiste em: “circuitos neurais dos sistemas de resposta e um estado de sensações que motivam e organizam cognição e ação”.

De fato, refletir sobre a possível definição de emoção é uma tarefa com complexidade intrínseca. Pensando apenas que as pessoas são muitas vezes incertas sobre suas próprias emoções, poderiam elas classificar com clareza as emoções dos outros? E, mesmo assim, muitas vezes elas reconhecem facilmente quando outras pessoas estão tristes, felizes, assustadas etc. Como seria possível então tentar ensinar uma máquina a reconhecer emoções humanas?

Ekman et al. (1971), definiu seis emoções básicas: felicidade, surpresa, nojo, medo, raiva e tristeza. E propôs um método para reconhecê-las através das expressões faciais, o Facial Affect Scoring Technique (FAST), onde ele propôs que as expressões faciais correspondentes às emoções teriam componentes ou “itens” em comum.

A emoção de surpresa, por exemplo, teria componentes de expressões faciais como sobrancelhas levantadas, olhos abertos, boca aberta etc. Em uma abordagem similar, a rede neural será treinada com o intuito de reconhecer as emoções baseando-se nas expressões faciais das imagens presentes na base de dados FER-2013 (KAGGLE, 2020).

2.2 Imagens

Para os computadores, imagens são matrizes constituídas por Pixels que possuem valores de 0 a 255, representando sua intensidade. Uma imagem colorida do tipo RGB, por exemplo, é um versor de rank 3. Pois, a imagem é representada por uma matriz onde seus elementos possuem um valor de 0 a 255 para cada um de seus canais de cores básicas, vermelho, verde e azul (EARNSHAW, 2017).

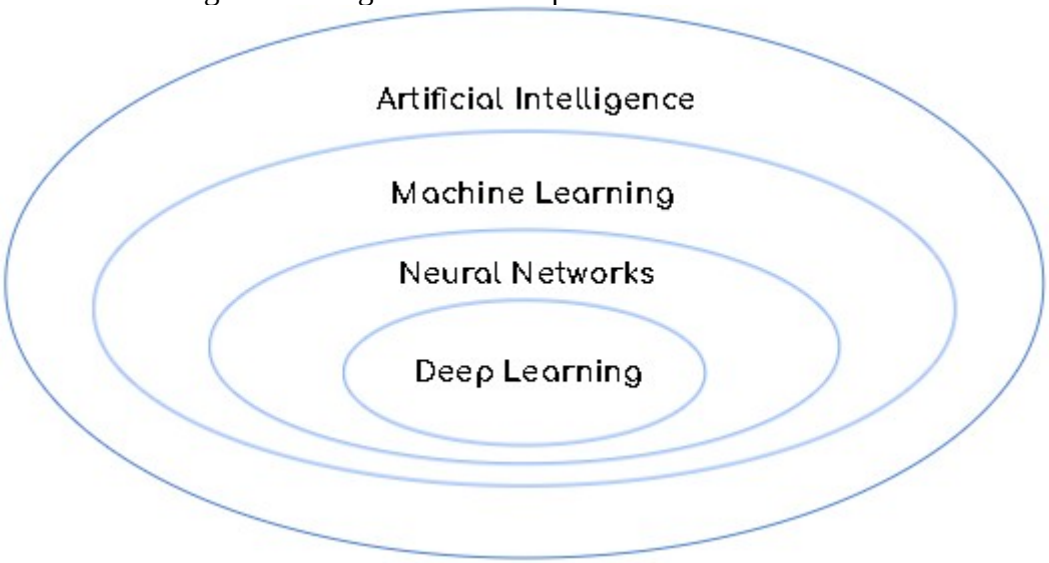
Imagens em escala de cinza possuem apenas um canal. Na análise de imagens cujas cores não são um fator relevante, o uso da escala de cinza possui uma vantagem com relação ao custo computacional, já que não é necessário processar valores de intensidade de 3 canais.

2.3 Inteligência Artificial, Machine Learning, Neural Networks e Deep Learning

“Na sua forma mais simples, IA (Inteligência Artificial) é um campo que combina ciência da computação e robustos conjuntos de dados, possibilitando a solução de problemas.” (IBM Cloud Education, 2020). Partindo dessa definição e da definição de McCarthy (2004), o campo da IA engloba o campo de *Machine Learning*, sendo este o estudo que fornece às máquinas a capacidade de aprender sem serem explicitamente programadas para isso (SAMUEL, 1959).

Dentro do campo de ML (*Machine Learning)*, existem diversas técnicas, como *Naive Bayes, K-Means, Support Vector Machines, Neural Networks*, entre outras. As Redes Neurais, ou *Neural Networks*, são portanto um subcampo de estudo de ML. E, por sua vez, aprendizado profundo, ou *deep learning*, é um subcampo de estudo de Redes Neurais. Essas relações são representadas no diagrama da Figura 2.

Figura 2 - Diagrama dos campos de estudo abordados.



Fonte: Autoria própria.

O paradigma comum a todas as técnicas de *Machine Learning* que tornam esses campos de estudo tão interessantes é a inversão no fluxo dos algoritmos. Os algoritmos comuns recebem um determinado input e são programados de maneira específica para produzirem um output, já os algoritmos de *Machine Learning* tem especificados tanto seus inputs quanto seus outputs e são programados de forma que possam automaticamente produzir uma maneira de fornecer esses outputs. Com a esperança de que, futuramente, essa maneira responda bem em um contexto generalizado.

2.4 Tipos de aprendizado

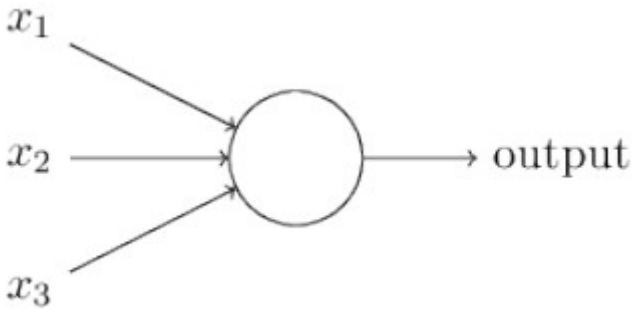
Segundo Delua (2021), existem duas abordagens básicas utilizadas no campo de *Machine Learning*, o aprendizado supervisionado e o não supervisionado. O aprendizado supervisionado é feito através de inputs e outputs já conhecidos, o não supervisionado se dá pelo uso apenas de inputs e não fornece outputs conhecidos.

Para problemas de classificação e regressão são mais comuns os usos de aprendizagem supervisionada, trabalhos com análise de sentimentos são geralmente abordados como problemas de classificação. Problemas de agrupamento ou *clustering* utilizam aprendizado não supervisionado, onde o principal objetivo é observar relações e padrões entre dados disponíveis, como trabalhos de detecção de anomalias.

2.5 Estrutura das Redes Neurais

A unidade básica de uma Rede Neural conhecida como nó, neurônio, ou perceptron surgiu a partir de uma estrutura desenvolvida por Frank Rosenblatt em 1957 (NIELSEN, 2015). Seu objetivo, na época, era explicar como a memória era armazenada em sistemas biológicos, sendo o perceptron a unidade básica do seu “brain model”, que consistia em unidades conectadas formando uma rede. Onde cada uma delas, ao receber um sinal de entrada, responde gerando um sinal de saída que pode ser transmitido, através de conexões, para um grupo seleto de unidades receptoras (ROSENBLATT, 1961).

Figura 3 - Estrutura de um perceptron com 3 entradas.



Fonte: Nielsen (2015)

O perceptron possui uma estrutura como a representada na Figura 3, este em específico recebe 3 entradas. A saída gerada possui valores binários, 1 ou 0, a maneira que Rosenblatt utilizou para computar essa saída foi através de pesos ou *weights*, números reais que atribuem a importância das conexões. Generalizando para mais do que 3 entradas, o valor de entrada no neurônio é dado pela soma do produto das saídas dos perceptrons da camada anterior e seus respectivos pesos, se o valor obtido ultrapassar um limite estabelecido a saída do perceptron é 1, sinalizando que ocorreu uma ativação, caso contrário, sua saída possui o valor 0, indicando uma inativação.

Sendo x_j e w_j as saídas dos perceptrons da camada anterior e os pesos das conexões, respectivamente. Temos que a saída de um perceptron é dada por:

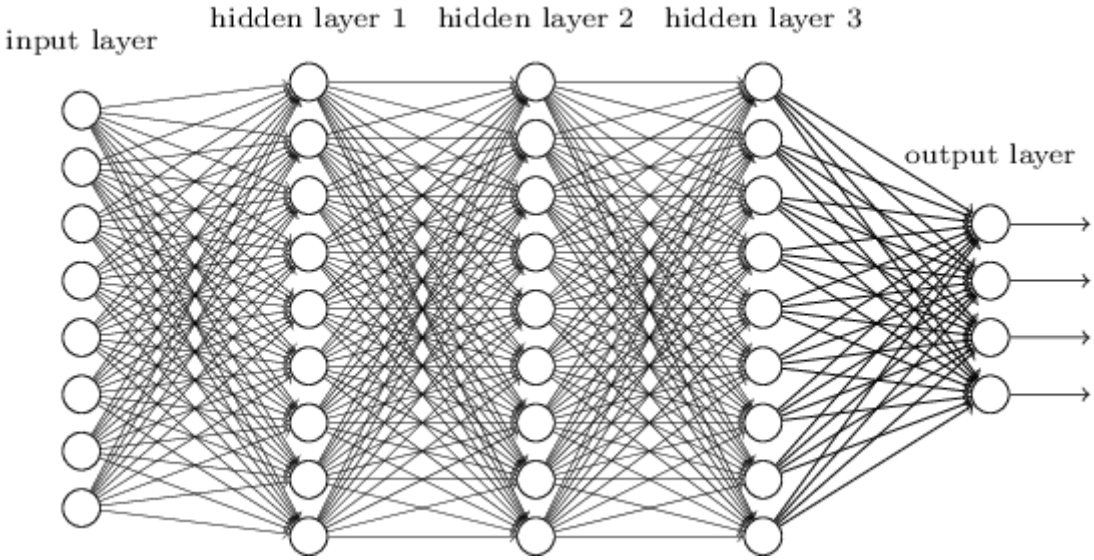
$$\text{output} = \begin{cases} 0 & \text{if} \quad \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if} \quad \sum_j w_j x_j > \text{threshold} \end{cases} \qquad (1)$$

A diferença crucial entre o perceptron, ou neurônio, das redes utilizadas atualmente está no modo que é computada sua saída. Diferente dos perceptrons de Rosenblatt, que não possuem níveis intermediários de ativação, os neurônios atuais possuem um valor de saída sendo um número real entre 0 e 1. Essa característica proporciona uma propriedade fundamental de que pequenas alterações nos pesos causam pequenas alterações nas saídas dos neurônios. E, como será visto mais adiante, essa é a propriedade que torna possível o aprendizado das redes.

Estruturalmente, as redes neurais consistem de camadas de neurônios, que nada mais são do que números reais representando suas “ativações”, interligados por pesos, que também são

números reais, cuja função é atribuir relevância entre as conexões. Uma Rede Neural possui no mínimo 3 destas camadas, uma camada de entrada, uma camada intermediária "escondida" chamada de *hidden layer* e uma camada de saída. As Redes Neurais Profundas ou *Deep Neural Networks* são redes com mais de uma *hidden layer*. Essa estrutura pode ser observada na Figura 4.

Figura 4 - Estrutura de uma rede neural completamente conectada.



Fonte: Nielsen (2015)

As Redes Neurais também podem ser completamente conectadas, onde a saída de cada neurônio é uma das entradas de cada um dos neurônios da camada seguinte, ou podem ser parcialmente conectadas, onde a afirmação anterior não é satisfeita. Neste trabalho serão abordadas apenas as redes completamente conectadas ou *fully connected networks*.

2.6 Funcionamento das Redes Neurais

2.6.1 Passagem de informação: *forward propagation* ou *feedforward*

A propagação de informação nas redes tratadas neste trabalho se dá de maneira sequencial de uma camada para a outra e é chamada de *forward propagation* ou *feedforward*, essa ideia se mantém semelhante a técnica utilizada por Rosenblatt (1961) e se dá por duas etapas.

- Entrada de informação no neurônio pela soma das saídas dos neurônios anteriores multiplicadas pelos seus respectivos pesos, somado também um *bias*.

- Aplicação de uma função de ativação que resulta na saída do neurônio e será utilizada para obter a entrada nos neurônios da camada subsequente.

Com exceção da camada de entrada que recebe os dados a serem analisados, todos os neurônios dependem das saídas dos neurônios anteriores, essa relação se dá pela equação 2.

$$input = b + \sum_k x_k w_k \tag{2}$$

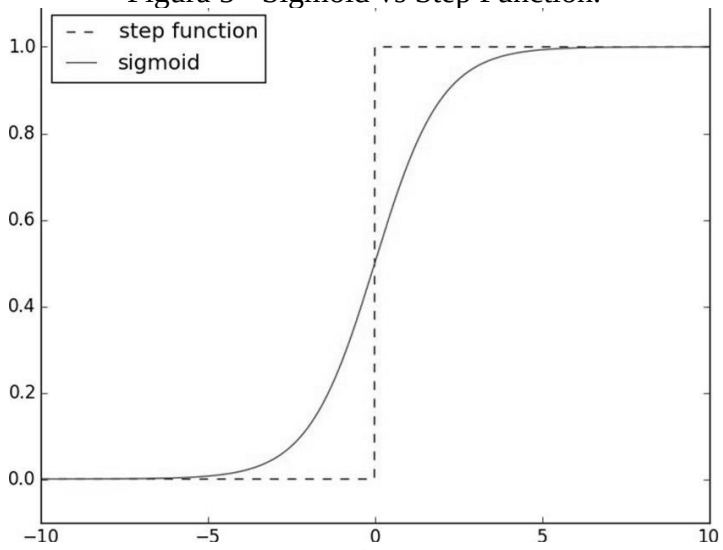
Pode ser visto que a equação 2 é praticamente análoga a equação de Rosenblatt (1961), tendo como diferença apenas o fator do *bias*, sendo este mais um parâmetro ajustável para a rede. O *bias* pode ser interpretado como um viés de ativação, quando o valor do *bias* é bastante positivo há uma maior facilidade da ativação do seu neurônio, quando é negativo há uma maior dificuldade desta ativação (NIELSEN, 2015).

Computada a entrada do neurônio, sua saída será dada aplicando uma função de ativação. Diferentemente da saída proposta para o perceptron através de uma função degrau, as funções de ativação são um grupo de funções que fornecem uma saída com valores contínuos que possuem propriedades semelhantes a ela.

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{3}$$

A função sigmóide (Equação 3) é uma das mais comuns, quanto maior seu valor de entrada mais ela se aproxima do valor unitário, e quanto menor o valor de entrada, seu resultado se aproxima a zero, como pode ser observado na Figura 5.

Figura 5 - Sigmoid vs Step Function.

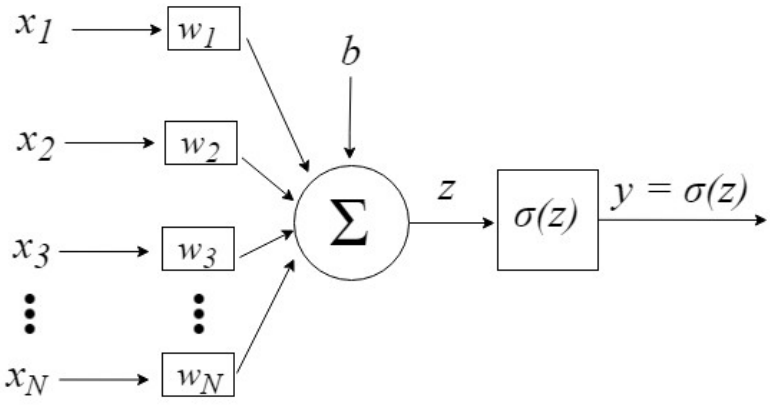


Fonte: Matthew (2020).

O aspecto fundamental da função sigmóide como já comentado antes é a propriedade de assumir valores contínuos entre zero e um. Isso permitirá o uso do algoritmo de *backward propagation* ou *backpropagation*, que será abordado posteriormente.

Um resumo esquemático do algoritmo de *feedforward* pode ser visto na Figura 6, onde, após todo o processo descrito, a saída do neurônio será utilizada na entrada dos neurônios da camada seguinte, por se tratar de uma rede completamente conectada.

Figura 6 - Configuração esquemática de feedforward para um neurônio.



Fonte: Calebe (2019)

2.6.2 Função de custo, perda ou objetiva

Em problemas de aprendizado supervisionado é possível calcular a diferença entre o output fornecido pela rede e o seu valor esperado. Essa diferença é calculada através de uma

função vista na literatura como *Loss Function, Cost Function* ou *Objective Function* (NIELSEN, 2015).

Uma delas é o erro quadrático médio (EQM), a *quadratic cost function* é dada pela equação 4.

$$C(w,b)=\frac{1}{2n}\sum_{x}\left(y(x)-a\right)^2 \tag{4}$$

Sendo w e b todos os pesos e biases da rede, respectivamente, n é o número de amostras de treino, x é o input de uma única amostra, $y(x)$ é o output esperado em relação ao input x e a é o output fornecido pela rede. Note que a notação utilizada difere da figura 6, agora o output do neurônio é dado pela variável a e y representa o output real que deveria ser observado.

Outra função de custo também muito utilizada é a *cross-entropy cost*, dada pela equação 5.

$$C(w,b)=\frac{-1}{n}\sum_{x}\left[yln(a)+(1-y)\ln(1-a)\right] \tag{5}$$

Como $y(x)$ é um valor já conhecido para cada input, o único valor que pode ser alterado é a saída a fornecida pela rede, que é função dos pesos e biases. Então, o objetivo é modificar estes dois parâmetros, chamados de parâmetros de aprendizado, de forma a minimizar a função de custo. Para isso, é utilizado o método da descida do gradiente ou *gradient descent*.

2.6.3 *Gradient descent.*

“O que queremos dizer quando falamos que uma rede está aprendendo é apenas que ela está minimizando uma função de custo.” (SANDERSON, 2017).

Gradient descent é um método iterativo utilizado para minimizar funções de múltiplas variáveis. Dado um ponto de ínicio da função, a cada iteração é calculado o seu gradiente neste ponto, em seguida, o gradiente é multiplicado por um escalar negativo (chamado em ML de *learning rate)* e então somado ao ponto inicial resultando dessa maneira em um novo ponto, onde o processo será repetido. A equação 6 demonstra de maneira mais geral o método.

$$v_{n+1} = v_n - \eta \nabla f(v_n) \tag{6}$$

Sendo $f(v)$ uma função de múltiplas variáveis, $v = v_1, v_2, \ldots$.

Dois pontos importantes a se notar neste método, são:

- O gradiente dá a direção de maior variação na superfície f, um escalar positivo resultaria em uma tendência de maximizar a função.
- Não há garantia de convergência para o mínimo global, apenas mínimos locais.

O segundo ponto é o mais importante, pois ele está relacionado com o hiperparâmetro *learning rate* na equação, dado por η. Enquanto não há o que possa ser feito com relação à garantia de encontrar apenas mínimos locais, a escolha do *learning rate* afeta consideravelmente os resultados da rede. Uma vez que um η grande pode tirar a convergência de um mínimo local e levá-la a outro, torná-lo o menor possível é importante para evitar que isso aconteça. Mas, ao torná-lo pequeno, serão necessárias mais iterações para a convergência e, consequentemente, um maior custo computacional. Este e outros pontos de otimização e boas práticas serão abordados posteriormente.

Então, aplicando *gradient descent* na função de custo, seus componentes serão dados pelas equações 8 e 9.

$$\nabla C = \left(\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}\right) \tag{7}$$

$$w_{n+1} = w_n - \eta \frac{\partial C}{\partial w_n} \tag{8}$$

$$b_{m+1} = b_m - \eta \frac{\partial C}{\partial b_m} \tag{9}$$

Aplicar este método se mostrou uma tarefa extremamente desafiadora. Pois, é necessário encontrar a derivada parcial da função de custo em relação a cada um dos pesos e cada um dos biases. Sendo que as redes neurais de hoje chegam a ter facilmente milhões desses parâmetros, acaba sendo inviável a abordagem de um cálculo analítico de cada uma dessas derivadas individualmente e também muito custoso um processo numérico de aproximação delas.

O algoritmo que solucionou este problema e que, segundo Nielsen (2015), é responsável pelo avanço das redes neurais visto nos dias de hoje, só ganhou relevância após a publicação de um paper em 1986 (RUMELHART, 1986), tal algoritmo é chamado de *backpropagation*.

2.6.4 Convenções adotadas

Antes de abordar o tópico *backpropagation* serão definidas algumas convenções adotadas por Nielsen (2015).

- O peso que conecta dois neurônios terá notação w^l_{jk} onde jé o número do neurônio da camada l e k é o número do neurônio da camada anterior $(l-1)$.
- O bias de um neurônio é denotado por b^l_j, sendo l a camada em que ele se encontra e j o seu número nesta camada.
- A saída de um neurônio será denotada de forma similar ao bias, onde j representa o número do neurônio na camada l, sendo representada então por a^l_j . A saída do neurônio também pode ser denominada como sua ativação.

Agora é possível generalizar a ativação de um neurônio pela equação 10.

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right) \tag{10}$$

Sendo σ a função de ativação, a expressão $\sum_k w^l_{jk} a^{l-1}_k + b^l_j$ pode ser denominada pela variável z^l_j chamada de *weighted input*, tornando a equação 10 mais compacta.

$$a^l_j = \sigma\left(z^l_j\right) \tag{11}$$

Nota-se que são equações repletas de índices, isso por estarem sendo considerados neurônios individuais. Para que seja possível escrevê-las de forma matricial, serão definidas duas representações adotadas e o produto de Hadamard.

- A aplicação de uma função em um vetor não é definida na matemática, mas convenciona-se neste trabalho que uma função aplicada a um vetor nada mais é que aplicar a função em cada um dos seus elementos.
- De maneira similar, a derivada de um vetor será dada pela derivada de cada um dos seus elementos.
- O produto de Hadamard é menos usual, porém é definido na matemática como o produto de elemento por elemento de uma matriz e é denotado pelo símbolo $s \odot j$ sendo s e j matrizes de mesmas dimensões (NIELSEN, 2015; MILLION, 2007).

De maneira mais elegante a equação 10 pode ser reescrita com a notação de vetores e matrizes pela equação 12, considerando as convenções acima.

$$A^l = \sigma\left(W^l A^{l-1} + B^l\right) \tag{12}$$

Sendo, A^l o vetor de ativações de j neurônios da camada l, W^la matriz de dimensões $j \times k$ com todos os pesos que conectam a camada l com sua antecessora $l-1$, A^{l-1} o vetor de ativações de k neurônios da camada $l-1$e B^lo vetor de bias dos j neurônios da camada l.

De maneira análoga à equação 12, podemos reescrever a equação 11, onde $Z^l = W^l A + B^l$ representa o vetor de *weghted inputs* da camada l.

$$A^l = \sigma\left(Z^l\right) \tag{13}$$

2.6.5 *Backpropagation*

Como visto anteriormente, a função degrau, utilizada por Rosenblatt (1961), não possuía uma característica fundamental para a aplicação do método *gradient descent*, ela não é contínua e, portanto, não é diferenciável.

O algoritmo de *backpropagation* é uma maneira inteligente de computar as derivadas parciais da função de custo em relação aos parâmetros de aprendizado de uma rede (seus *weights* e *biases*).

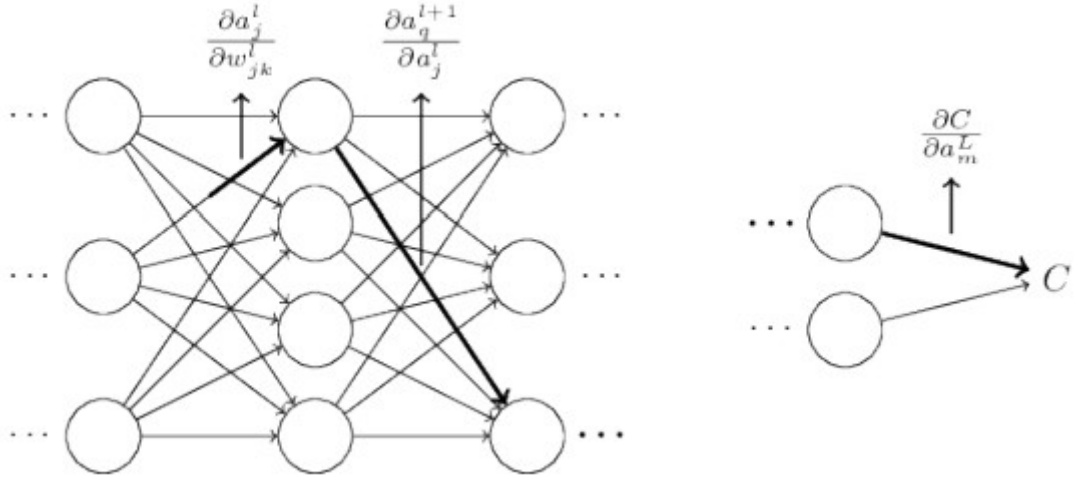
Observa-se que a ativação de uma camada é dependente de todas as camadas anteriores, com exceção da camada de input dos dados. Sendo A^l o vetor de ativações da última camada de uma rede com input X, a aplicação sucessiva da equação 12 descreve a dependência entre as camadas e é representada pela equação 14.

$$A^l = \sigma\left(W^l \sigma\left(W^{l-1} \sigma\left(\ldots W^2 \sigma\left(W^1 X + B^1\right) + B^2\right) + B^{l-1}\right) + B^l\right) \quad (14)$$

Dessa forma, nota-se que a saída de cada camada é uma função composta e, portanto, para encontrar a derivada parcial da função de custo em relação a um peso específico é necessário aplicar a regra da cadeia, como demonstrado na equação 15 e ilustrado na figura 7.

$$\frac{\partial C}{\partial w^l_{jk}} = \sum_{mnp\ldots q} \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \quad (15)$$

Figura 7 - Representação da derivada parcial do custo em relação a w^l_{jk}.



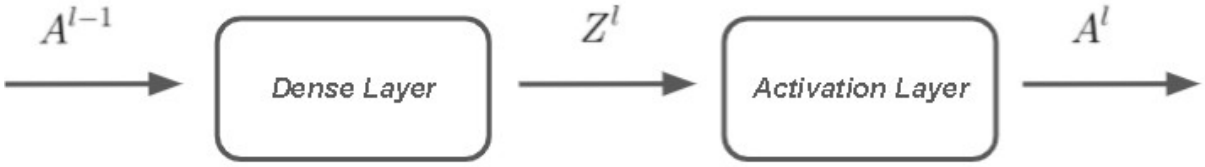
Fonte: Nielsen (2015).

Não é necessário esforço para demonstrar o motivo das redes neurais dependerem de um algoritmo que simplifique este processo. Baseado na mesma abordagem da passagem de informação do *feedforward*, a ideia do *backpropagation* consiste em propagar o valor das derivadas necessárias e, posteriormente, a descida do gradiente partindo da última camada para a primeira, computando o seu efeito em cada uma delas.

A adaptação de uma abordagem proposta por Aflak (2018), similar a de Nielsen (2015), permite visualizar seu funcionamento com mais clareza. Subdividindo uma camada

qualquer l em duas, uma denominada de *dense layer* e outra de *activation layer* o fluxo de informação no *feedforward* pode ser representado na figura 8.

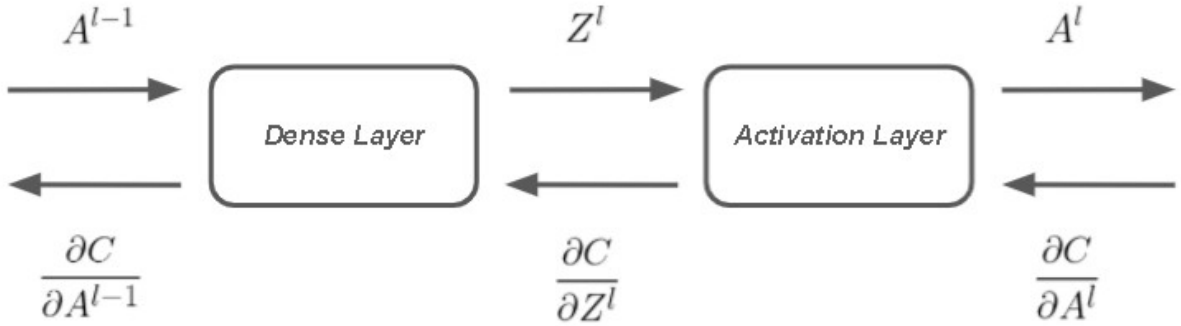
Figura 8- Feedforward em uma camada l dividida em dense layer e activation layer.



Fonte: Adaptado de Aflak (2018).

Sendo a *dense layer* a camada cuja entrada é o vetor de ativação de k neurônios de A^{l-1} que se ligam a j neurônios da camada A^l , sua saída é Z^l, o vetor de *weighted inputs*. E *activation layer* a camada que apenas aplica a função de ativação em todos os elementos do vetor Z^l resultando na saída A^l.

Figura 9 - Representação do feedforward e backpropagation em uma camada l subdividida.



Fonte: Adaptado de Aflak (2018).

De maneira análoga, o *backpropagation* é representado no fluxo da figura 9, pelas setas localizadas na sua parte inferior e, a partir da análise de ambas as camadas, serão apresentadas as 4 equações necessárias para sua aplicação.

- *Dense Layer*

Figura 10 - Representação da Dense Layer

Fonte: Adaptado de Aflak (2018).

Foram retirados os índices da camada para fins de simplificação na notação, uma vez que a camada é arbitrária. O vetor Z é dado pela equação 16.

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_j \end{bmatrix} = \begin{bmatrix} w_{11}a_1 + w_{12}a_2 + \cdots + w_{1k}a_k + b_1 \\ w_{21}a_1 + w_{22}a_2 + \cdots + w_{2k}a_k + b_2 \\ \vdots \\ w_{j1}a_1 + w_{j2}a_2 + \cdots + w_{jk}a_k + b_j \end{bmatrix} \tag{16}$$

A equação 16 pode ser reescrita de forma matricial, como mostra a equação 17.

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_j \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} \\ w_{21} & w_{22} & \cdots & w_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j1} & w_{j2} & \cdots & w_{jk} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_j \end{bmatrix} \tag{17}$$

Dado um peso qualquer w_{cd}, a derivada parcial do custo C em relação a w_{cd} é dada pela equação 18.

$$\frac{\partial C}{\partial w_{cd}} = \frac{\partial C}{\partial z_1}\frac{\partial z_1}{\partial w_{cd}} + \frac{\partial C}{\partial z_2}\frac{\partial z_2}{\partial w_{cd}} + \cdots + \frac{\partial C}{\partial z_c}\frac{\partial z_c}{\partial w_{cd}} + \cdots + \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial w_{cd}} \tag{18}$$

O único z cuja derivada parcial da equação 18 não é zero é o z_c correspondente ao peso w_{cd}, pois esse é o único elemento de Z que depende de w_{cd} e o valor desta derivada é a ativação correspondente a_d. Então, simplificando, a derivada parcial do custo em relação a qualquer peso é dada pela equação 19.

$$\frac{\partial C}{\partial w_{cd}} = \frac{\partial C}{\partial z_c}\frac{\partial z_c}{\partial w_{cd}} = \frac{\partial C}{\partial z_c}a_d \tag{19}$$

Representando todas as derivadas parciais do custo em relação a todos os pesos em uma matriz pela equação 20, e reescrevendo-a em forma de um produto matricial, chega-se à primeira das quatro equações fundamentais para o método, a equação 21.

$$\frac{\partial C}{\partial W} = \begin{bmatrix} \frac{\partial C}{\partial w_{11}} & \frac{\partial C}{\partial w_{12}} & \cdots & \frac{\partial C}{\partial w_{1k}} \\ \frac{\partial C}{\partial w_{21}} & \frac{\partial C}{\partial w_{22}} & \cdots & \frac{\partial C}{\partial w_{2k}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial w_{j1}} & \frac{\partial C}{\partial w_{j2}} & \cdots & \frac{\partial C}{\partial w_{jk}} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial z_1}a_1 & \frac{\partial C}{\partial z_1}a_2 & \cdots & \frac{\partial C}{\partial z_1}a_k \\ \frac{\partial C}{\partial z_2}a_1 & \frac{\partial C}{\partial z_2}a_2 & \cdots & \frac{\partial C}{\partial z_2}a_k \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial z_j}a_1 & \frac{\partial C}{\partial z_j}a_2 & \cdots & \frac{\partial C}{\partial z_j}a_k \end{bmatrix} \tag{20}$$

$$\frac{\partial C}{\partial W} = \begin{bmatrix} \frac{\partial C}{\partial z_1} \\ \frac{\partial C}{\partial z_2} \\ \vdots \\ \frac{\partial C}{\partial z_j} \end{bmatrix} \begin{bmatrix} a_1 & a_2 & \cdots & a_k \end{bmatrix} = \frac{\partial C}{\partial Z}A^T \tag{21}$$

Partindo da equação 17, de maneira análoga à derivada parcial do custo em relação aos pesos, é possível obter a derivada parcial do custo em relação aos biases, representada pela equação 22.

$$\frac{\partial C}{\partial B} = \begin{bmatrix} \frac{\partial C}{\partial b_1} \\ \frac{\partial C}{\partial b_2} \\ \vdots \\ \frac{\partial C}{\partial b_j} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial z_1}\frac{\partial z_1}{\partial b_1} + \frac{\partial C}{\partial z_2}\frac{\partial z_2}{\partial b_1} + \cdots + + \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial b_1} \\ \frac{\partial C}{\partial z_1}\frac{\partial z_1}{\partial b_2} + \frac{\partial C}{\partial z_2}\frac{\partial z_2}{\partial b_2} + \cdots + + \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial b_2} \\ \vdots \\ \frac{\partial C}{\partial z_1}\frac{\partial z_1}{\partial b_j} + \frac{\partial C}{\partial z_2}\frac{\partial z_2}{\partial b_j} + \cdots + + \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial b_j} \end{bmatrix} \quad (22)$$

Assim como os pesos, a única derivada parcial que não é zero é a de z_c correspondente ao bias b_c, e essa por sua vez é igual a 1, já que não há nenhum fator que o multiplica. Então, obtém-se a equação 23, a segunda equação necessária.

$$\frac{\partial C}{\partial B} = \begin{bmatrix} \frac{\partial C}{\partial b_1} \\ \frac{\partial C}{\partial b_2} \\ \vdots \\ \frac{\partial C}{\partial b_j} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial z_1} \\ \frac{\partial C}{\partial z_2} \\ \vdots \\ \frac{\partial C}{\partial z_j} \end{bmatrix} = \frac{\partial C}{\partial Z} \quad (23)$$

Foram obtidas as matrizes com as derivadas parciais necessárias para atualizar os parâmetros de aprendizado no método da descida do gradiente. A *dense layer* ainda fornecerá a terceira equação necessária para o *backpropagation*, a derivada do custo em relação às entradas, representada pela equação 24.

$$\frac{\partial C}{\partial A} = \begin{bmatrix} \frac{\partial C}{\partial a_1} \\ \frac{\partial C}{\partial a_2} \\ \vdots \\ \frac{\partial C}{\partial a_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial z_1}\frac{\partial z_1}{\partial a_1} + \frac{\partial C}{\partial z_2}\frac{\partial z_2}{\partial a_1} + \cdots + \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial a_1} \\ \frac{\partial C}{\partial z_1}\frac{\partial z_1}{\partial a_2} + \frac{\partial C}{\partial z_2}\frac{\partial z_2}{\partial a_2} + \cdots + \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial a_2} \\ \vdots \\ \frac{\partial C}{\partial z_1}\frac{\partial z_1}{\partial a_k} + \frac{\partial C}{\partial z_2}\frac{\partial z_2}{\partial a_k} + \cdots + \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial a_k} \end{bmatrix} \quad (24)$$

Trabalhando com a derivada parcial do custo em relação uma entrada arbitrária a_c, pela equação 25.

$$\frac{\partial C}{\partial a_c} = \frac{\partial C}{\partial z_1}\frac{\partial z_1}{\partial a_c} + \frac{\partial C}{\partial z_2}\frac{\partial z_2}{\partial a_c} + \cdots + \frac{\partial C}{\partial z_d}\frac{\partial z_d}{\partial a_c} + \cdots + \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial a_c} \tag{25}$$

Todos os termos da equação 25 são diferentes de zero, pois o fator a_c está presente em todos os elementos do vetor Z, e a derivada parcial de qualquer z_d em relação a a_c é o peso w_{dc} que conecta o neurônio c da camada anterior ao neurônio d da camada atual, resultando na equação 26.

$$\frac{\partial C}{\partial A} = \begin{bmatrix} \frac{\partial C}{\partial a_1} \\ \frac{\partial C}{\partial a_2} \\ \vdots \\ \frac{\partial C}{\partial a_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial z_1}w_{11} + \frac{\partial C}{\partial z_2}w_{21} + \cdots + \frac{\partial C}{\partial z_j}w_{j1} \\ \frac{\partial C}{\partial z_1}w_{12} + \frac{\partial C}{\partial z_2}w_{22} + \cdots + \frac{\partial C}{\partial z_j}w_{j2} \\ \vdots \\ \frac{\partial C}{\partial z_1}w_{1k} + \frac{\partial C}{\partial z_2}w_{2k} + \cdots + \frac{\partial C}{\partial z_j}w_{jk} \end{bmatrix} \tag{26}$$

O vetor $\frac{\partial C}{\partial A}$ pode ser escrito como o produto entre a transposta dos pesos e o vetor de derivadas parciais do custo em relação aos *weighted inputs*, obtendo a equação 27, a terceira das quatro necessárias.

$$\frac{\partial C}{\partial A} = \begin{bmatrix} \frac{\partial C}{\partial a_1} \\ \frac{\partial C}{\partial a_2} \\ \vdots \\ \frac{\partial C}{\partial a_k} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{j1} \\ w_{12} & w_{22} & \cdots & w_{j2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1k} & w_{2k} & \cdots & w_{jk} \end{bmatrix} \begin{bmatrix} \frac{\partial C}{\partial z_1} \\ \frac{\partial C}{\partial z_2} \\ \vdots \\ \frac{\partial C}{\partial z_j} \end{bmatrix} = W^T \frac{\partial C}{\partial Z} \tag{27}$$

A quarta e última equação necessária para o *backpropagation* será obtida pela análise da *activation layer*.

- *Activation Layer*

Figura 11 - Representação da Activation Layer.

Fonte: Adaptado de Aflak (2018).

Na camada de ativação, não há parâmetros de aprendizado para atualizar, ela apenas aplica a função de ativação em todos os *weighted inputs* que ela recebe da *dense layer* resultando na saída que será utilizada como entrada pela próxima *dense layer*. Então resta apenas uma relação necessária a ser encontrada, a derivada parcial dos *weighted inputs* em relação ao custo na camada de ativação.

Agora Z é o vetor de entradas da camada de ativação, a derivada parcial de seus elementos em função do custo pode ser representada pela equação 28.

$$\frac{\partial C}{\partial Z} = \begin{bmatrix} \frac{\partial C}{\partial z_1} \\ \frac{\partial C}{\partial z_2} \\ \vdots \\ \frac{\partial C}{\partial z_j} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial a_1}\frac{\partial a_1}{\partial z_1} \\ \frac{\partial C}{\partial a_2}\frac{\partial a_2}{\partial z_2} \\ \vdots \\ \frac{\partial C}{\partial a_j}\frac{\partial a_j}{\partial z_j} \end{bmatrix} \tag{28}$$

Esse vetor pode ser representado pelo produto de Hadamard e as ativações estão relacionadas aos *weighted inputs* pela equação 11, por $a=\sigma(z)$. Resultando na equação 29, a quarta e última equação necessária.

$$\frac{\partial C}{\partial Z} = \begin{bmatrix} \frac{\partial C}{\partial a_1} \\ \frac{\partial C}{\partial a_2} \\ \vdots \\ \frac{\partial C}{\partial a_j} \end{bmatrix} \odot \begin{bmatrix} \frac{\partial a_1}{\partial z_1} \\ \frac{\partial a_1}{\partial z_2} \\ \vdots \\ \frac{\partial a_1}{\partial z_j} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial a_1} \\ \frac{\partial C}{\partial a_2} \\ \vdots \\ \frac{\partial C}{\partial a_j} \end{bmatrix} \odot \begin{bmatrix} \sigma'(z_1) \\ \sigma'(z_2) \\ \vdots \\ \sigma'(z_j) \end{bmatrix} = \frac{\partial C}{\partial A} \odot \sigma'(Z) \quad (29)$$

Em resumo, todas as equações utilizadas na *dense layer* dependem apenas das derivadas parciais representadas pelo vetor $\frac{\partial C}{\partial Z}$, obtidos os seus valores é possível implementar computacionalmente o cálculo da derivada parcial do custo em relação a todos os pesos e biases de maneira simples, através das igualdades $\frac{\partial C}{\partial W}=\frac{\partial C}{\partial Z}A^T$ e $\frac{\partial C}{\partial B}=\frac{\partial C}{\partial Z}$, das equações 21 e 23, respectivamente. Essas duas equações, são as únicas necessárias para atualizar os pesos e biases na aplicação da descida do gradiente.

As outras duas equações apresentadas são as que dão continuidade ao ciclo. Uma vez obtida a derivada parcial do custo em relação aos *weighted inputs* $\frac{\partial C}{\partial Z}$ da *dense layer* poderá ser obtida a derivada parcial do custo em relação às ativações de entrada da *dense layer*, $\frac{\partial C}{\partial A}=W^T\frac{\partial C}{\partial Z}$ (Equação 27), ela será equivalente a derivada parcial do custo em relação às ativações de saída da *activation layer* da camada anterior. Obtida a derivada parcial do custo em relação às ativações da saída, $\frac{\partial C}{\partial A}$, da *activation layer* é possível obter $\frac{\partial C}{\partial Z}=\frac{\partial C}{\partial A}\odot\sigma'(Z)$ (Equação 29), que nada mais é do que a derivada parcial do custo em relação aos *weighted inputs* para a *dense layer*, seguindo para as demais camadas de maneira sucessiva.

Então, da mesma maneira que estabelecidos os pesos e biases iniciais, basta fornecer

o input para que ocorra a etapa de propagação de informação da primeira para última camada, ou *feedforward*, basta fornecer na última camada o valor da derivada da função de ativação em relação aos *weighted inputs* do vetor Z , $\sigma'(Z)$, e a derivada da função de custo em relação às ativações $\frac{\partial C}{\partial A}$, para que ocorra a etapa de *backpropagation*.

Fornecidos esses valores da última camada, essas 4 equações são aplicadas de forma sucessiva até a primeira camada. Em seguida é realizado método da descida do gradiente, em todos os elementos da matriz de pesos e da matriz de biases, camada por camada, como representado pelas equações 30 e 31, respectivamente, que nada mais são do que a representação matricial das equações 8 e 9.

$$W_{n+1} = W_n - \eta \frac{\partial C}{\partial W_n} \tag{30}$$

$$B_{m+1} = B_m - \eta \frac{\partial C}{\partial B_m} \tag{31}$$

Ao realizar esse processo a função de custo tenderá à um mínimo local, e os parâmetros de aprendizado serão ajustados para obter um maior número de outputs A^l que coincidam com o output esperado $Y(X)$, para todas as entradas X fornecidas na base de dados de treino da rede. O número de vezes que isso deve ser repetido (*epochs*), o valor do *learning rate*, os valores iniciais dos parâmetros de aprendizado serão abordados a seguir.

2.7 Hiperparâmetros, otimizações e boas práticas

O intuito de treinar uma rede neural é que ela obtenha sozinha seus parâmetros de aprendizado que permitam fornecer os outputs desejados. Porém, os parâmetros como *learning rate*, número de *epochs*, *mini batch size*, valores iniciais dos parâmetros de aprendizado, número de neurônios ou número de camadas, são parâmetros estabelecidos manualmente na implementação da rede neural e são denominados como hiperparâmetros. Eles serão abordados um a um com suas possíveis otimizações e boas práticas.

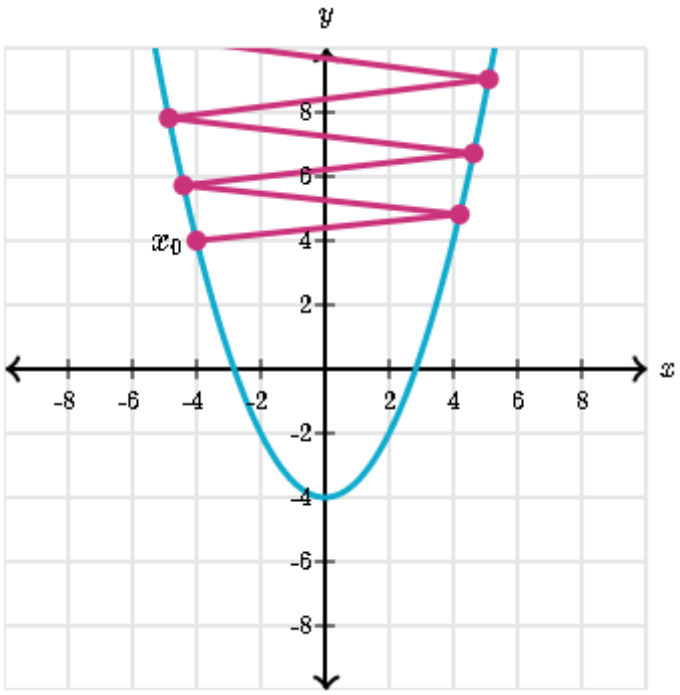
2.7.1 *Learning rate*

O *learning rate* representado nas equações anteriores por η é o escalar que altera o tamanho da variação dos parâmetros de aprendizado para cada iteração do *gradient descent*.

Como pode ser visto nas equações 30 e 31, cada peso e cada bias será subtraído pela derivada parcial do custo em relação a ele multiplicada por η, o que torna tentador atribuir valores altos a esse escalar, dessa maneira aumentando a velocidade com que se chega no mínimo local.

Porém, ao fazer isso, provavelmente não haverá convergência para um mínimo local. A função de custo pode acabar sendo levada para outros mínimos locais sem conseguir permanecer em algum deles e, mesmo que consiga, ela possivelmente não apresentará convergência dentro deste mínimo local. Ela ficará oscilando ao se aproximar do mínimo, como pode ser representado visualmente na figura 12.

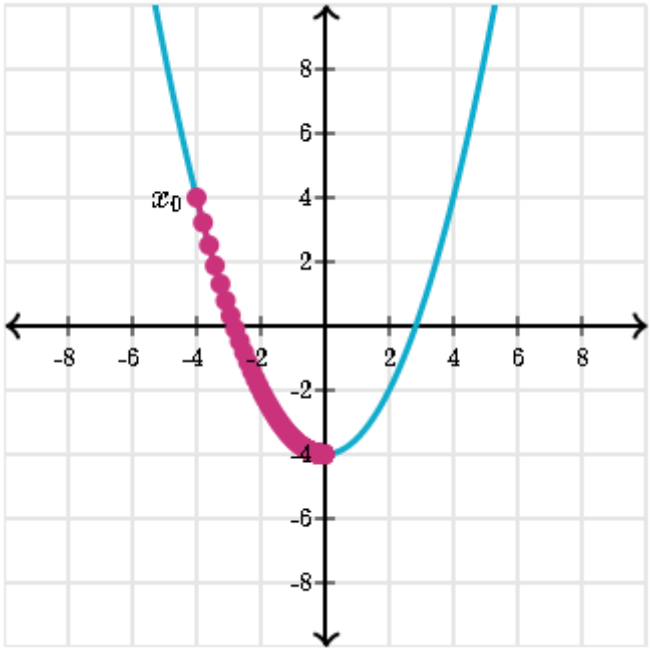
Figura 12 - Não convergência por um learning rate com valor alto



Fonte: https://bit.ly/3i5zAN8

Para que isso seja evitado pode-se adotar um valor pequeno para este parâmetro, porém a convergência se dará de maneira muito mais lenta, principalmente se a função a ser minimizada possuir milhões de variáveis, como pode ser representado na figura 13.

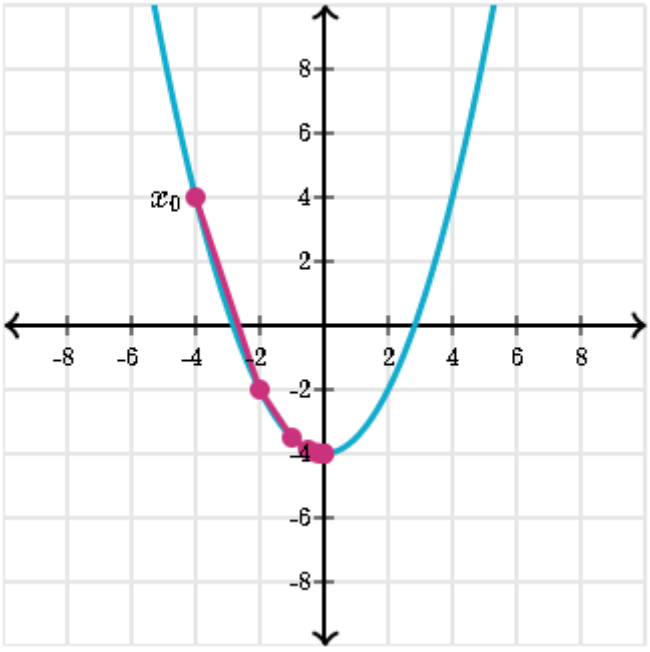
Figura 13 - Convergência com learning rate menor.



Fonte: https://bit.ly/3i5zAN8

Segundo Yan-Tak (2018), uma abordagem comum é adotar $\eta = 0.01/n$, sendo n o número de amostras de treino utilizadas, como um valor inicial e ir realizando ajustes finos através de testes. O objetivo é obter um valor que cause uma convergência rápida e para um intervalo de valores razoável, que varia de aplicação para aplicação, uma convergência ideal pode ser representada pela figura 14.

Figura 14 - Representação de uma convergência otimizada.



Fonte: https://bit.ly/3i5zAN8

2.7.2 *Epochs*

Epochs, ou épocas de treino, é apenas o número predeterminado de vezes que o ciclo *feedforward* e *backpropagation* será repetido para os inputs de treino fornecidos. O número de épocas necessárias varia bastante e esta quantidade está atrelada a um dos maiores problemas das redes neurais, o *overfitting*, que será abordado posteriormente.

2.7.3 *Stochastic gradient descent* e *mini batch size*

Quando se aplicam os métodos de *backpropagation* e *gradient descent*, o mais usual é determinar o *learning rate* como um escalar arbitrário α, com valor inicial de 0.01, dividido pelo número n de amostras da base de dados de treino que será utilizado pela rede para atualizar os parâmetros de aprendizado, como pode ser observado pela equação 32.

$$\eta = \frac{\alpha}{n} \tag{32}$$

Idealmente, a cada *epoch*, o *gradient descent* é aplicado em todas as amostras do conjunto de treino. Porém, isso pode tornar o treino da rede substancialmente mais lento, para contornar este problema é utilizado um método denominado *stochastic gradient descent*. Ele consiste em organizar o conjunto de dados de treino aleatoriamente e, em seguida, dividi-lo em *mini batches* com um número arbitrário de amostras com tamanho denominado de *mini batch size*.

Então, é aplicado o método *gradient descent* sobre essa amostra consideravelmente menor, o que traz ganhos de tempo e custo computacional. Segundo Bottou (2012), a convergência deste método é quase sempre garantida em condições moderadas e seu valor médio é aproximadamente igual ao valor médio obtido utilizando-se todas as amostras disponíveis.

Segundo Bengio (2012), são usualmente utilizados valores de *mini batch size* sendo potências de 2 partindo de 32 até 512. No paper de Keskar et al (2017), os pesquisadores observam indícios de que *mini batches* maiores degradam a qualidade do modelo em questão de capacidade de generalização.

2.7.4 Valores iniciais dos parâmetros de aprendizado

Segundo Nielsen (2015), uma abordagem comum é inicializar uma rede com pesos aleatórios e biases com valor zero. Uma vez que a derivada parcial do custo em relação aos

biases é dependente apenas da derivada parcial do custo em relação aos *weighted inputs*, zerar seus valores iniciais ainda permitirá sua atualização normalmente. Isso já se torna inviável para os pesos, se seu valor inicial for zero, todas as ativações serão zero e a rede não terá a capacidade de aprender.

Porém, há maneiras melhores de iniciar os pesos do que apenas atribuir valores aleatórios sem análise prévia. Um bom método para inicializá-los é atribuir a eles valores de uma distribuição normal com média 0 e desvio padrão $1/\sqrt{n}$, por conta da saturação do neurônio, que será abordada posteriormente. Mas, a ideia é que pesos mais concentrados em um intervalo de valores auxiliam no aprendizado da rede.

2.7.5 Número de neurônios e camadas

A quantidade de parâmetros de aprendizado é definida pelo número de camadas e neurônios em cada uma delas. Quanto mais parâmetros de aprendizado uma rede possuir, mais ela conseguirá “se moldar” aos dados do conjunto de treino. Mas, não necessariamente, isso é benéfico por conta de um fenômeno denominado como *overfitting* que, essencialmente, prejudica a capacidade da rede de generalização.

Segundo Brownlee (2018), não há uma heurística para o número de neurônios que devem existir em cada camada ou o número de camadas que devem constituir a rede, pesquisar por papers sobre problemas similares pode ser um bom ponto de partida para atribuir o número inicial de neurônios e camadas na arquitetura utilizada.

As redes deste trabalho terão no máximo duas camadas, por conta de um problema denominado *vanishing gradient* que está relacionado às funções de ativação e será abordado posteriormente.

2.8 Problemas comuns das redes neurais

2.8.1 Saturação do neurônio

Ao iniciar o treino de uma rede neural atribuindo aos pesos valores aleatórios, muito provavelmente os neurônios da camada de saída da rede fornecerão valores distantes dos valores esperados, o que é esperado. Porém, caso algum neurônio de saída tenha valores muito próximos a 1 ou a 0, sendo sua função de ativação a sigmóide e sua função de custo o EQM, ocorrerá um fenômeno denominado de saturação do neurônio.

Esse fenômeno ocorre por conta da derivação da sigmóide na etapa de *backpropagation*, dada pela equação 33, onde pode-se observar que se z for grande o suficiente para que $\sigma(z)$ se aproxime de 1, o valor da derivada se aproximará de zero. E caso z seja pequeno o suficiente para que $\sigma(z)$ se aproxime de zero, o valor da derivada sofrerá o mesmo efeito.

$$\sigma'(z) = \sigma(z)(1-\sigma(z)) \tag{33}$$

Com a função de custo EQM, a derivada parcial do custo em relação a um peso é dependente da derivada da sigmóide, como pode ser visto pela equação 34. Dessa forma, se $\sigma'(z)$ for próximo de zero, a taxa de aprendizado dos pesos atrelados a este neurônio será próxima de zero tornando-os irrelevantes para o aprendizado da rede. Isso causa uma lentidão no aprendizado ou *learning slowdown*.

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x \tag{34}$$

Para contornar essa limitação, a função de custo mais utilizada, dado que a função de ativação é a sigmóide, é a *cross-entropy cost* (equação 5). Pois, sua derivada em relação aos pesos, equação 35, não depende da derivada da sigmóide.

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z)-y) \tag{35}$$

Ao escolher essa função de custo, o problema é resolvido para a saturação dos neurônios da camada de saída, porém, esse mesmo problema pode ocorrer nas *hidden layers*. Se o *weighted input* z de qualquer neurônio for um valor que torne a derivada da sigmóide próxima de zero os pesos atrelados a esse neurônio sofreram o mesmo *learning slowdown*. Infelizmente para esse problema nas *hidden layers* a alteração da função de custo para *cross-entropy cost* não causa nenhum efeito, já que inevitavelmente será utilizada a derivada da sigmóide na etapa de *backpropagation*.

Uma maneira de tentar suprimir esse problema é na inicialização dos pesos e como descrito anteriormente, um bom método para fazer isso é iniciá-los como valores de uma distribuição normal com média 0 e desvio padrão $1/\sqrt{n}$. Se os pesos iniciais tomarem valores suficientemente maiores que 1 ou menores que -1, haverá um fator multiplicativo nas suas conexões, que pode causar um *weighted input* resultante grande o suficiente (tanto positivo quanto negativo) para saturar o neurônio. Tornar os pesos iniciais mais concentrados em

torno da média 0 auxilia diminuindo a quantidade de neurônios em que isso ocorre (NIELSEN, 2015).

2.8.2 *Overfitting* e *underfitting*

Quando a rede neural continua a melhorar sua acurácia no conjunto de dados de treino, mas não apresenta melhorias de acurácia no conjunto de dados de teste, ela está adaptando seus parâmetros para obter especificamente os outputs esperados do seu conjunto de treino, o que causa um grande problema. Se a rede se especializar em encontrar os outputs esperados, não desempenhará bem de forma generalizada em dados desconhecidos, isso é denominado *overfitting*.

Por isso ele é o fenômeno dominante na escolha do número de *epochs* ou épocas de treino, assim que não houver mais uma melhoria na acurácia dos dados de teste é recomendável parar o treinamento da rede. Pois, dali em diante a rede apenas aprenderá peculiaridades do seu conjunto de dados de treino, este método é conhecido como *early stopping*.

Underfitting é exatamente o contrário, parar o treino de uma rede sem que ela alcance o máximo de acurácia no conjunto de testes prejudicará seu desempenho em novos dados, uma vez que ela ainda poderia aprender mais sem perder sua generalização (NIELSEN, 2015).

2.8.3 *Vanishing Gradient*

Em uma das etapas de execução do *backpropagation* ocorre uma multiplicação pela derivada da sigmóide para cada uma das camadas da rede. A função sigmóide possui valores entre 0 e 1 e sua derivada também, como pode ser visto na equação 33. Este fato causará um efeito de diminuição no gradiente a cada camada presente na rede, de maneira exponencial.

Sendo o gradiente cada vez menor, a atualização dos parâmetros de aprendizado se tornará cada vez mais lenta. Portanto, é necessário tomar um grande cuidado com esse problema, principalmente em redes neurais profundas com muitas camadas (NIELSEN, 2015).

2.9 *Categorical Encoding*

Como a rede neural utilizada neste trabalho utiliza aprendizado supervisionado (inputs e outputs são fornecidos) e o problema abordado é um problema de classificação das emoções, uma técnica denominada *encoding* precisa ser aplicada para viabilizar o treinamento da rede.

Encode significa “modificar informação de forma que possa ser processada por um computador” (ENCODE, 2021). *Categorical Encoding* é apenas a representação de dados categóricos por números ou vetores que serão utilizados pela rede no processo de treinamento.

Uma variável categórica é uma variável cujos valores são representados por classificações. A variável “cor”, por exemplo, pode assumir valores classificados como “azul”, “verde” e “vermelho”. Essa variável pode ser representada pelo método de *One Hot Encoding* como um vetor binário de 3 posições, onde cada um deles representa uma cor e poderia ser visto como: Azul = $[1,0,0]$, Verde = $[0,1,0]$, Vermelho = $[0,0,1]$ (BROWNLEE, 2019).

Este método é particularmente útil em redes neurais, já que a camada de saída nada mais é do que um vetor com valores entre 0 e 1, onde a posição de maior valor indicará a sua categoria. Dessa forma, se a saída de uma rede treinada para detectar cores em um pixel assumir o valor $[0.89,0.13,0.06]$, ela indica que o pixel possui a cor azul, caso seja o valor correto, significa que o output esperado é, de fato, o vetor $[1,0,0]$.

3 METODOLOGIA

3.1 Base de dados

Para o treinamento das redes neurais foi utilizada a base de dados FER-2013 (KAGGLE, 2020). Ela consiste em mais de 30 mil imagens em escala de cinza de 48x48 pixels divididas em um conjunto de treino e um conjunto de teste, classificadas em 7 emoções: Raiva, nojo, medo, felicidade, tristeza, surpresa e neutralidade. Distribuídas de acordo com as tabelas 1 e 2.

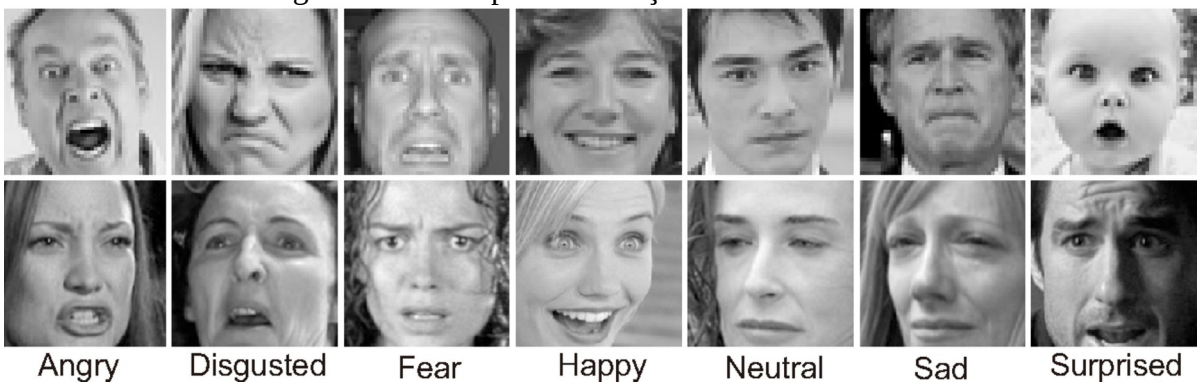
Tabela 1 - Quantidade de imagens por emoção no conjunto de Treino.

Emoção	Quantidade de imagens
Raiva	3995
Nojo	436
Medo	4097
Felicidade	7215
Neutralidade	4965
Tristeza	4830
Surpresa	3171

Tabela 2 - Quantidade de imagens por emoção no conjunto de Teste.

Emoção	Quantidade de imagens
Raiva	958
Nojo	111
Medo	1024
Felicidade	1774
Neutralidade	1233
Tristeza	1247
Surpresa	831

Figura 15 - Exemplos de emoções da base de dados.



Fonte: (KAGGLE, 2020).

3.2 Linguagem e Ambiente de execução utilizados

As primeiras redes foram treinadas utilizando Python 3.7.1 no *Google Collaboratory*, que é um ambiente de execução em nuvem fornecido pelo *Google Research*. Nele, é possível executar códigos em Python tendo acesso a uma máquina com 12GB de memória RAM e 100GB de disco que executa código por até 12h ininterruptas, de forma gratuita.

Após determinados parâmetros com melhor performance dentre as redes treinadas, foram treinadas 66 redes em um notebook dell vostro 5470 com 8GB de RAM, com hiperparâmetros próximos a estes pré-determinados para fins de comparação e validação de uma rede ótima entre elas.

3.3 Código utilizado

O código utilizado foi adaptado do código de Dobrzanski (2016), que é uma versão atualizada para Python 3 do já disponível no livro e GitHub de Nielsen (2015) feito em Python 2.7.

Foram utilizadas também bibliotecas como, *numpy*, *matplotlib* e *pandas* para auxiliar no tratamento e análise dos dados.

3.4 Tratamento dos dados

3.4.1 Encoding dos dados

Como cada imagem do dataset possui dimensões de 48x48 pixels, seus valores foram todos normalizados, dividindo-os por 255 (máximo de intensidade possível de um pixel). Em seguida elas foram transformadas em vetores de uma coluna e 48x48 = 2304 posições, sendo eles as entradas na camada de input da rede.

Foi realizado um *categorical encoding* na classificação das emoções, a cada uma das 5 emoções foi atribuído um número de 0 a 4 , sendo representados pela tabela 3.

Tabela 3 - Emoções e seus respectivos valores de encoding.

Emoção	Valor do Enconding	Vetor de One Hot Encoding
Raiva	0	[1, 0, 0, 0, 0]
Medo	1	[0, 1, 0, 0, 0]
Felicidade	2	[0, 0, 1, 0, 0]
Neutralidade	3	[0, 0, 0, 1, 0]
Tristeza	4	[0, 0, 0, 0, 1]

Então, para cada emoção é atribuído um vetor de 5 posições cujo valor do encoding representa a posição que possui valor 1. Dessa maneira será possível comparar diretamente a camada de saída que é definida com 5 neurônios.

3.4.2 Balanceamento dos dados

Quanto maior a quantidade de dados para treinar uma rede, melhor. O conjunto de dados utilizado possui aproximadamente um mínimo de 4 mil imagens por emoção, exceto as emoções de nojo e surpresa que possuem 436 e 3171 imagens, respectivamente.

Por conta do desbalanceamento dos dados em relação às emoções de nojo e surpresa, para evitar um viés nos dados de treinamento da rede, elas não foram utilizadas. Restando apenas 5 emoções para a análise.

3.5 Treinamento das redes

3.5.1 Parâmetros iniciais

Nas primeiras redes treinadas alguns parâmetros utilizados foram escolhidos com base em heurísticas gerais, como *learning rate* inicial de 0.01, e outros com base em parâmetros ótimos para o problema de reconhecimento de números escritos à mão, MNIST abordado por Nielsen (2015), como *mini batch size* de valor 10 e 30 neurônios por camada.

3.5.2 Teste dos parâmetros

Após observar a acurácia de diversos parâmetros foram utilizadas redes com apenas duas *hidden layers*, uma vez que o problema de *vanishing gradient* torna muito mais lento o processo de treinamento de uma rede com múltiplas hidden layers se a função de ativação possui sua imagem contida entre 0 e 1.

Definido o número de *hidden layers*, foram testados números de neurônios próximos a 5000 como apresentados por Nordén (2019), e empiricamente *learning rates* próximas a 0.1 e número de *epochs* superiores a 100, demonstraram uma melhor performance e evidenciaram pontos que serão discutidos nos resultados.

A partir disso determinou-se intervalos de valores de *epochs*, *learning rate* e número de neurônios por camada para cada uma das redes a serem comparadas como pode ser visto na tabela 4.

Tabela 4 - Hiperparâmetros testados.

Hiperparâmetro	Intervalo de Valores
neurônios por camada	50; 100; 150
learning rate	0.1; 0.2; 0.3; 0.4; 0.5
epochs	100; 200; 300
mini batch size	32

3.5.3 Refinamento dos parâmetros

Comparando os resultados das 45 redes treinadas com os parâmetros de teste, houve motivação para treinar mais 36 redes com parâmetros refinados. Tal motivação será discutida nos resultados, eles podem ser observados na tabela 5.

Tabela 5 - Hiperparâmetros refinados

Hiperparâmetro	Intervalo de Valores
neurônios por camada	100; 150
learning rate	0.05; 0.1; 0.15
epochs	100; 200
mini batch size	16; 32; 64

3.5.4 Avaliação de performance

A partir dos dados obtidos para cada uma dessas redes, foram escolhidas as 3 melhores em questão de performance, adotando como medida a melhor acurácia média exibida no conjunto de imagens de teste e com o menor desvio padrão, a partir da *epoch* 50.

O número de *epochs* a partir do qual foi medida a performance foi determinado pelo método de *early stopping* utilizado para evitar o problema de *overfitting* observando os gráficos de acurácia da rede sobre o conjunto de teste.

4 RESULTADOS

Nesta seção serão discutidos os resultados obtidos ao longo do treinamento das redes com diversos hiperparâmetros, assim como destacados os pontos que levaram à escolha dos mesmos e das redes com melhores performances.

4.1 Redes iniciais

Os hiperparâmetros utilizados no treinamento da primeira rede foram os apresentados pela tabela 6.

Tabela 6 - Hiperparâmetros primeira rede de treino.

Nodes / hidden layer	30
Mini batch size	10
Epochs	30
Learning Rate	3

Esta foi a única rede treinada com apenas uma *hidden layer*, a acurácia média obtida sobre o conjunto de teste desta rede foi de 22.2%, que se aproxima da acurácia esperada de 20% ao se escolher uma emoção aleatoriamente entre as 5 classificadas.

Isso está relacionado ao fato de que os hiperparâmetros escolhidos foram os mesmos que Nielsen (2015) utiliza em seu livro para obter uma acurácia superior a 95% no reconhecimento de dígitos escritos a mãos, um problema completamente diferente. Seus resultados de nada serviriam para identificar emoções em imagens.

4.2 Busca por parâmetros

As 66 redes treinadas foram divididas em duas levas de parâmetros, onde na primeira delas foram treinadas 30 redes e na segunda 36. As motivações e resultados são demonstrados a seguir.

4.2.1 Primeira leva de treinamento

Foram treinadas mais de 20 redes no *google colaboratory* a fim de delimitar um range de parâmetros promissores a serem utilizados na primeira leva da procura de uma rede com melhor acurácia.

Partindo dos trabalhos de Bendio (2012) e Keskar et al (2017), como mencionado na seção 2.7.3, o *mini batch size* escolhido para iniciar a busca foi de 32 amostras por *mini*

batch. O número total de neurônios de cada uma das redes foi definido na mesma ordem de grandeza dos 5000 tidos como melhor quantidade no paper apresentado por Nordén (2019) comentado na seção 3.5.2. A partir do treinamento das redes no *google colaboratory* foi observado um melhor desempenho utilizando *learning rates* em torno de 0.1.

O período de uso limitado do *google laboratory* impossibilita uma melhor avaliação do número de *epochs*, pois quanto maior este número maior o tempo de treinamento necessário. Houve então a tentativa de treinar redes com todos os parâmetros listados na tabela 4 em um *dell vostro 5470*. Porém, após aproximadamente 158h de execução no treinamento das redes, notou-se uma pior performance naquelas com *learning rates* superiores a 0.2 e o código foi interrompido obtendo os resultados de treino de apenas 30 redes como primeira leva de treinamento. Excluindo, dessa forma, as redes com 150 neurônios por camada, como pode ser visto na tabela 7.

Tabela 7 - Hiperparâmetros das 30 redes treinadas na primeira leva.

Hiperparâmetro	Intervalo de Valores
neurônios por camada	50; 100
learning rate	0.1; 0.2; 0.3; 0.4; 0.5
epochs	100; 200; 300
mini batch size	32

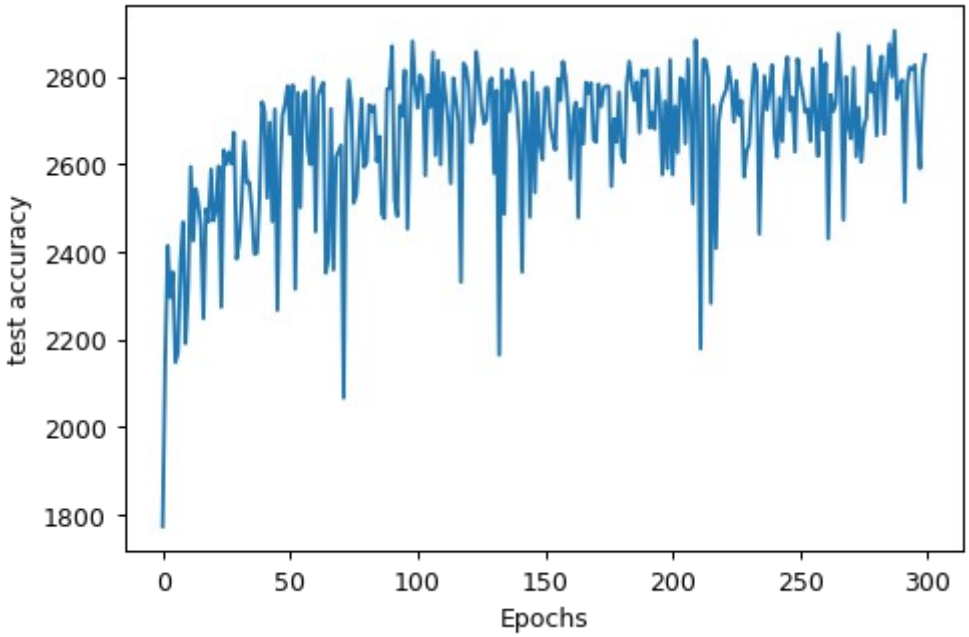
Dentre as 30 redes treinadas foram escolhidas as 3 com maior acurácia média, que resultantes dos hiperparâmetros apresentados na tabela 8.

Tabela 8 - Hiperparâmetros 3 redes com maior acurácia média da primeira leva.

Hiperparâmetro	1ª Rede	2ª Rede	3ª Rede
neurônios por camada	100	100	100
learning rate	0.1	0.2	0.2
epochs	300	300	200
mini batch size	32	32	32

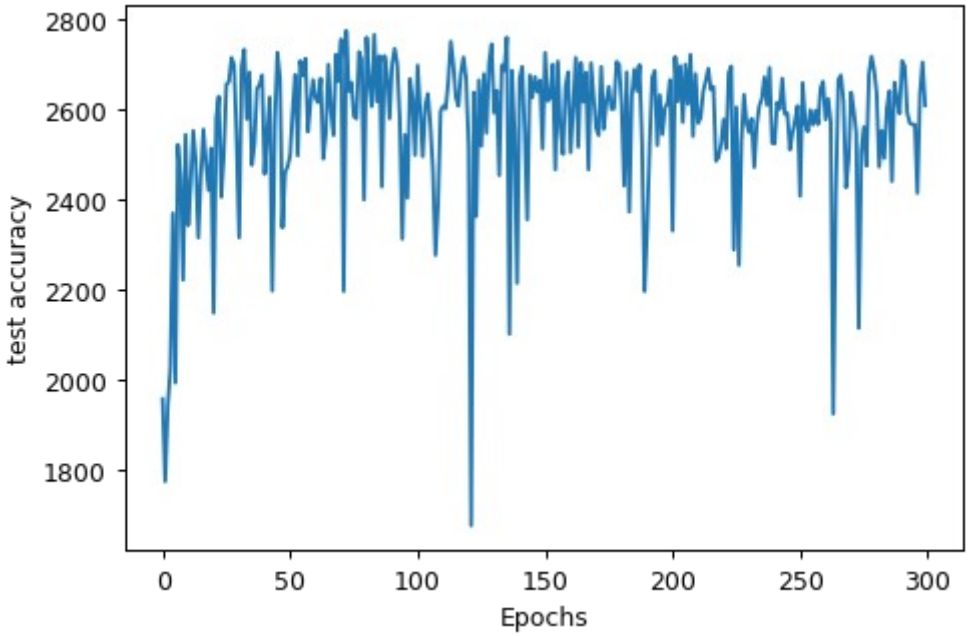
Foi observado também uma tendência de estabilização da acurácia no conjunto de testes a partir da *epoch* 50, como pode ser visto nos gráficos das figuras 16, 17 e 18.

Figura 16 - Gráfico test accuracy vs epochs 1ª Rede, primeira leva.



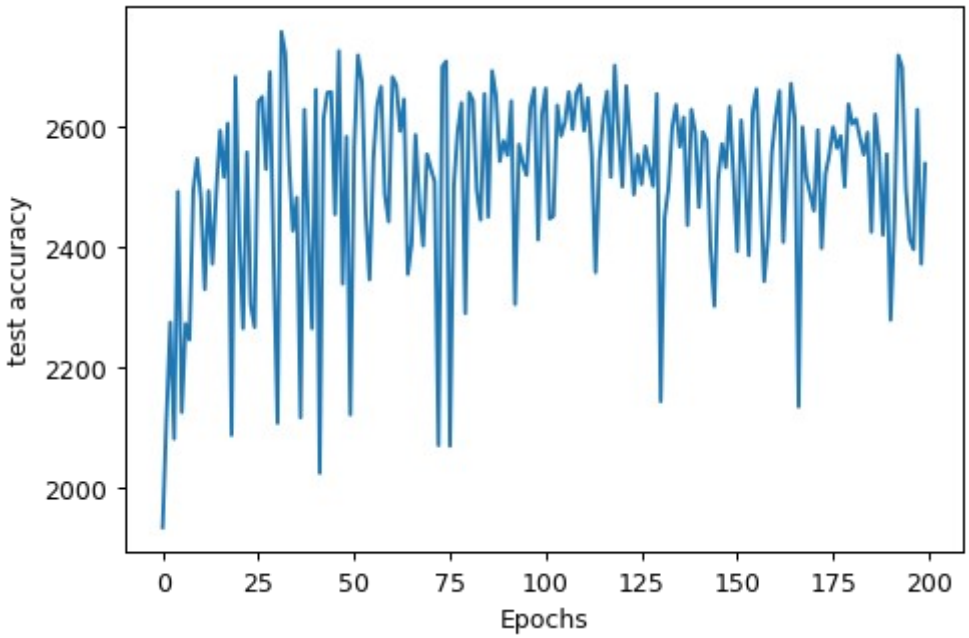
Fonte: Elaborado pelo autor.

Figura 17 - Gráfico test accuracy vs epochs 2ª Rede, primeira leva.



Fonte: Elaborado pelo autor.

Figura 18 - Gráfico test accuracy vs epochs 3ª Rede, primeira leva.



Fonte: Elaborado pelo autor.

Suas acurácias médias e respectivos desvios padrão são apresentados na tabela 9.

Tabela 9 - Média e desvio padrão das 3 redes com maior acurácia média da primeira leva.

Rede	Acurácia Máxima	Acurácia Média	Desvio Padrão
1ª	2904	2688.313	138.604
2ª	2776	2585.52	135.221
3ª	2538	2529.514	140.107

Das 5834 imagens do conjunto de teste, a 1ª rede obteve acurácia média e acurácia máxima na classificação das emoções de aproximadamente 46% e 49.7%, respectivamente, que é muito superior em comparação aos 22% da primeira rede.

Dois fatores interessantes também podem ser observados nos gráficos acima, a influência do *learning rate* no tempo de convergência e também na dispersão dos dados. Ambos serão comentados posteriormente.

4.2.1 Segunda leva de treinamento

Baseado nos resultados obtidos pelo treino das primeiras 30 redes, foram treinadas mais 36 redes com parâmetros próximos aos das redes que apresentaram melhores performances na primeira leva de treinamento, exatamente como descrito na tabela 5 da seção 3.5.3.

Após, aproximadamente, 151h de execução foram obtidas as 3 melhores redes dentre a última leva de treinamento, essas redes possuem os hiperparâmetros listados na tabela 10.

Tabela 10 - Hiperparâmetros das 3 redes com maior acurácia média na segunda leva.

Hiperparâmetro	1ª Rede	2ª Rede	3ª Rede
neurônios por camada	100	150	150
learning rate	0.05	0.05	0.05
epochs	100	200	100
mini batch size	16	16	16

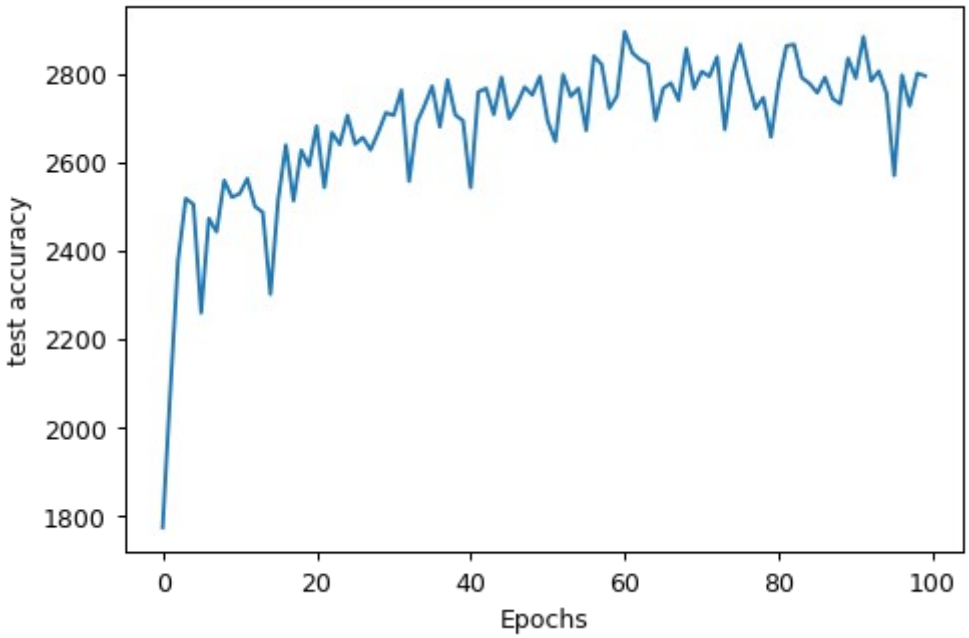
Os resultados obtidos por elas são consideravelmente melhores e podem ser observados na tabela 11.

Tabela 11 - Média e desvio padrão das 3 redes com maior acurácia média da segunda leva.

Rede	Acurácia Máxima	Acurácia Média	Desvio Padrão
1ª	2894	2752.306	72.889
2ª	2931	2750.422	99.623
3ª	2863	2728.187	90.680

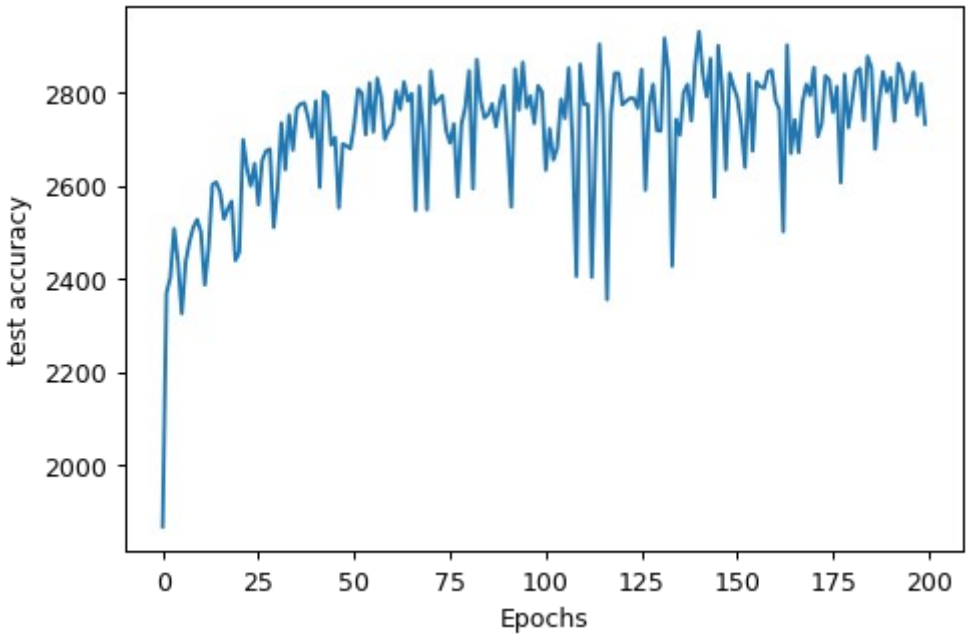
A estabilização da acurácia parece se iniciar em torno do mesmo número de *epochs* da primeira leva, 50. E pode ser observado nas figuras 19, 20 e 21.

Figura 19 - Gráfico test accuracy vs epochs 1ª Rede, segunda leva.



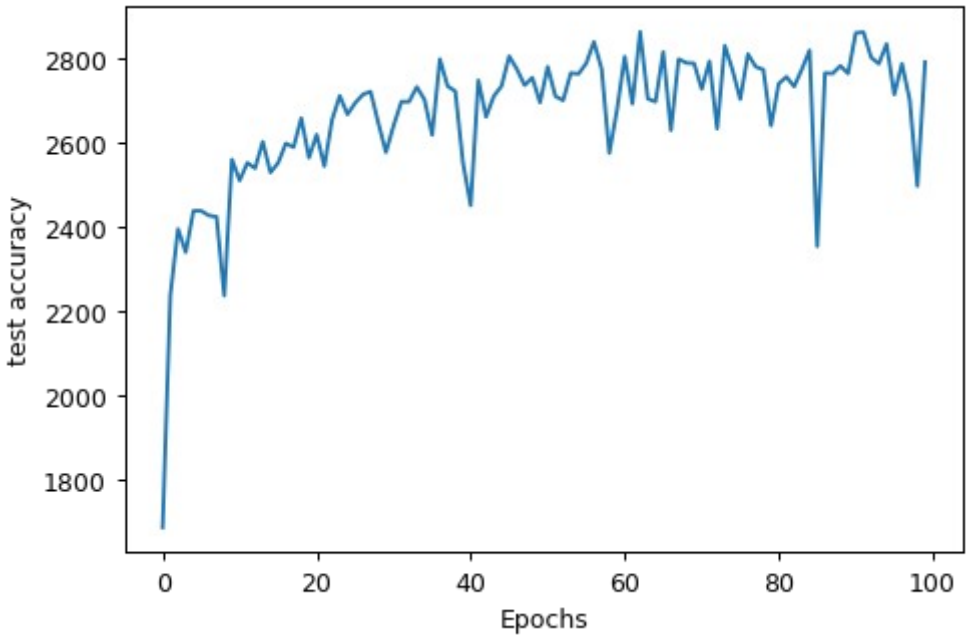
Fonte: Elaborado pelo autor.

Figura 20 - Gráfico test accuracy vs epochs 2ª Rede, segunda leva.



Fonte: Elaborado pelo autor.

Figura 21 - Gráfico test accuracy vs epochs 3ª Rede, segunda leva.



Fonte: Elaborado pelo autor.

É notável a diminuição das oscilações apresentadas pelos gráficos da segunda leva comparados a primeira, essa diminuição de barulho é associada ao *mini batch size*. Outro ponto a considerar é o aumento na acurácia média e a diminuição do desvio padrão, um fato interessante é que o *learning rate* de todas as redes com melhor performance é o mesmo, ambos os pontos serão discutidos posteriormente.

4.3 Análise dos hiperparâmetros com melhor performance

Dentre os resultados exibidos pelas 66 redes treinadas, as evidências de alguns fatores teóricos sobre as redes neurais relacionadas aos hiperparâmetros se destacam e serão analisadas a seguir.

4.3.1 *Learning rate*

Durante a procura por um valor ótimo de *learning rate*, dois pontos principais foram observados:

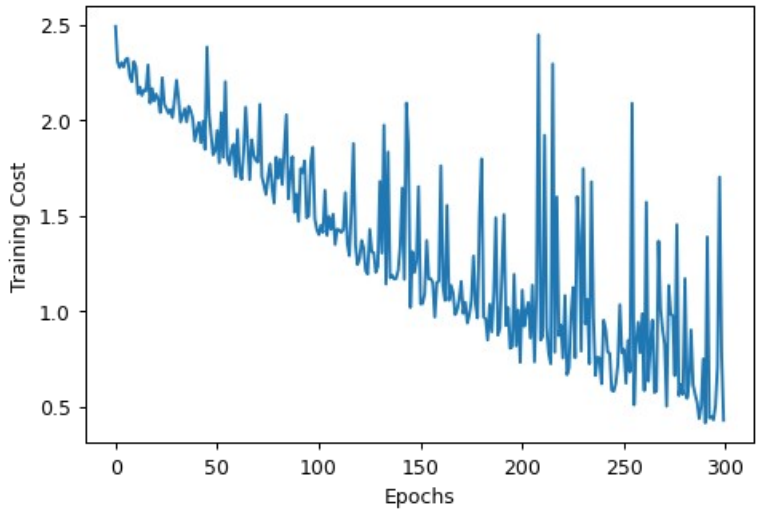
- A velocidade de convergência
- Acurácia para qual convergem os valores.

Sobre a velocidade de convergência, comparando os gráficos das figuras 17 e 18, a acurácia da rede com *learning rate* = 0.1 parece começar sua estabilização na faixa das 50 *epochs* enquanto a rede com *learning rate* = 0.2 apresenta sinal de estabilidade na faixa de 25

epochs o que é esperado já que este é um termo na equação do *gradient descent* que influencia diretamente no quanto o valor do custo se aproxima do mínimo local em cada iteração.

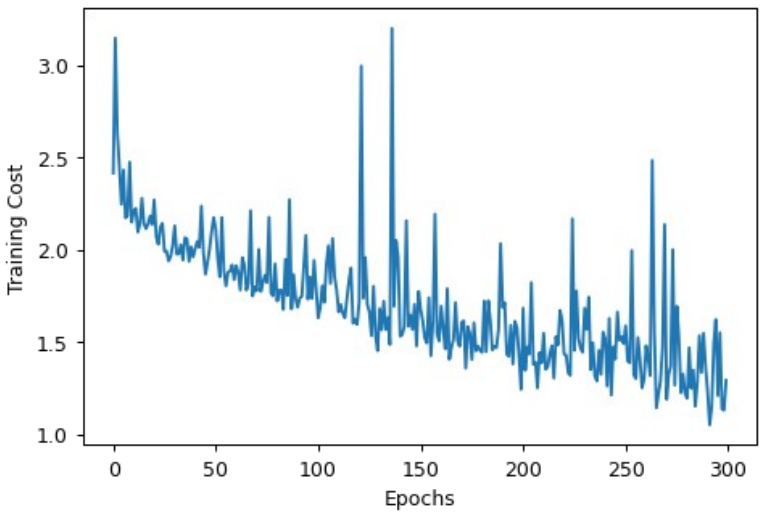
Apesar da velocidade de convergência ser diretamente proporcional ao *learning rate*, a acurácia obtida não possui a mesma relação, uma vez que valores maiores deste hiperparâmetro impossibilitam a função de custo atingir o ponto mínimo de um vale local. Estes dois pontos estão representados nas figuras 22 e 23, que mostram os valores da função de custo da 1ª e 2ª rede da primeira leva, respectivamente.

Figura 22 - Gráfico training cost vs epochs 1ª Rede, primeira leva.



Fonte: Elaborado pelo autor.

Figura 23 - Gráfico training cost vs epochs 2ª Rede, primeira leva

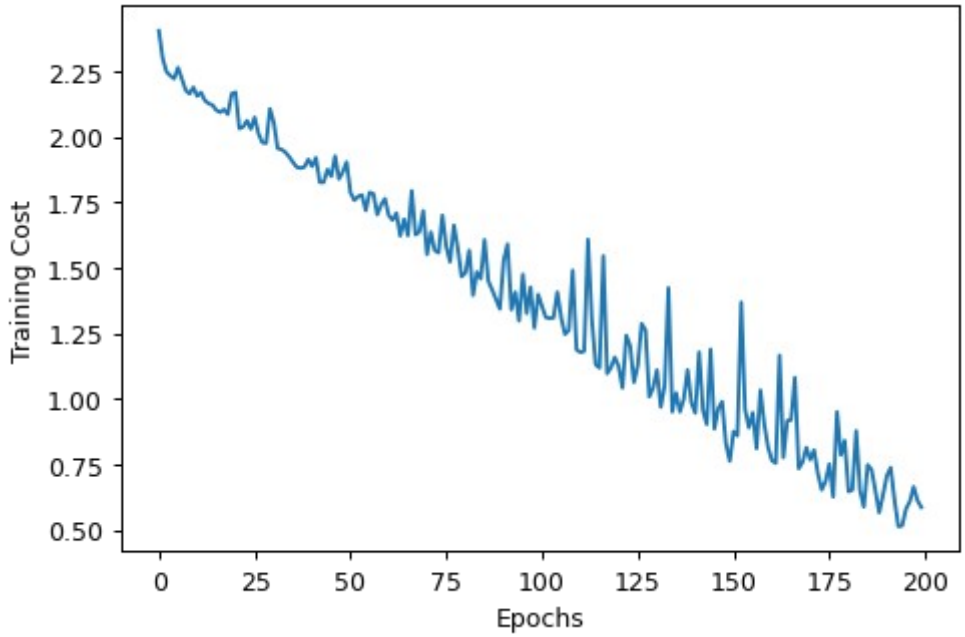


Fonte: Elaborado pelo autor.

Enquanto o gráfico da 2ª rede (figura 23) mostra sinais de um início de estabilização em 300 *epochs*, o gráfico da 1ª rede (figura 22) parece continuar convergindo cada vez mais para zero. Apesar da velocidade de convergência da rede com maior *learning rate* ser maior o valor obtido da função de custo não chega a ficar abaixo de 1.0, em comparação, o *learning rate* menor apresenta valores próximos a 0.5 em 300 *epochs* sem sinal de alterar essa tendência de diminuição.

Para as redes da segunda leva, nota-se algo interessante. Todos os *learning rates* das 3 redes com melhor performance são iguais e possuem valor de 0.05, o menor dos valores de *learning rate* utilizados nos treinamentos. Isso também vai de encontro com a proposta teórica de que quanto menor o seu valor, mais o *gradient descent* conseguirá aproximar o valor do custo ao seu mínimo local. O gráfico da função de custo da 2ª rede da segunda leva (a única dentre as 3 com 200 *epochs*), apresentado na figura 24, permite uma melhor visualização.

Figura 24 - Gráfico training cost vs epochs 2ª Rede, segunda leva.



Fonte: Elaborado pelo autor.

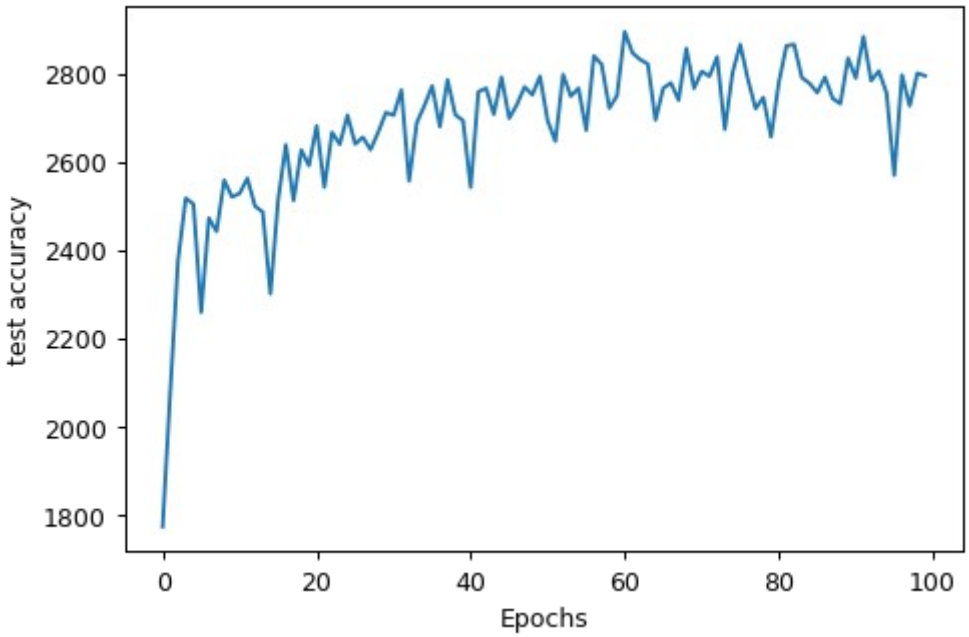
Na figura acima não há sinais de alteração na taxa de variação da curva em direção ao mínimo da função de custo, em 200 *epochs*, o que implica em uma convergência mais lenta. Em contrapartida, os números obtidos na minimização da função de custo são muito mais animadores. É notável também a diminuição das oscilações nos gráficos das redes da segunda leva, isso se deve ao *mini batch size* e será abordado a seguir.

4.3.2 *Mini batch size*

Segundo Keskar et al (2017), um tamanho maior no número de amostras utilizadas em cada *mini batch* causa uma degradação significativa na qualidade do modelo em relação a sua habilidade de generalização.

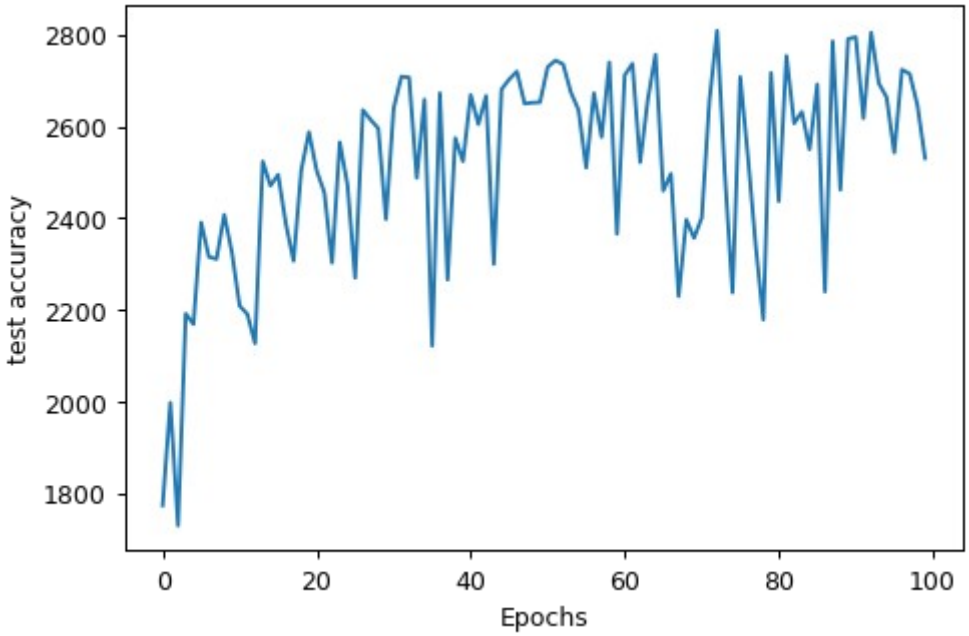
Nos resultados da segunda leva, não foi apenas o menor *learning rate* que demonstrou a melhor performance, mas também o menor *mini batch size* foi o hiperparâmetro utilizado pelas 3 redes com melhor performance. E, ao contrário da primeira leva, foram utilizados 3 valores (16, 32 e 64) a fim de comparar os seus efeitos sobre as redes. As figuras 25, 26 e 27 mostram os gráficos da rede classificada como a melhor dentre as da segunda leva e seus 3 *mini batch sizes* utilizados.

Figura 25 - Gráfico test accuracy vs epochs 1ª Rede, segunda leva, mini batch size 16.



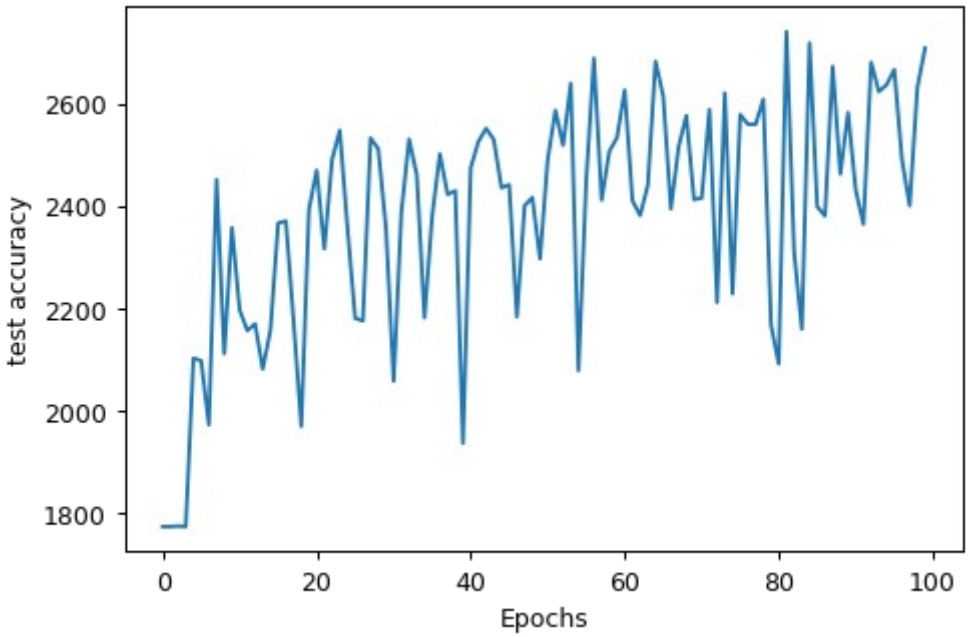
Fonte: Elaborado pelo autor.

Figura 26 - Gráfico test accuracy vs epochs 1ª Rede, segunda leva, mini batch size 32.



Fonte: Elaborado pelo autor.

Figura 27 - Gráfico test accuracy vs epochs 1ª Rede, segunda leva, mini batch size 64.



Fonte: Elaborado pelo autor.

Fica evidente, pelos gráficos acima, que há uma piora considerável na acurácia da rede sobre o conjunto de testes quanto maior o *mini batch size* utilizado. Além de oscilações cada

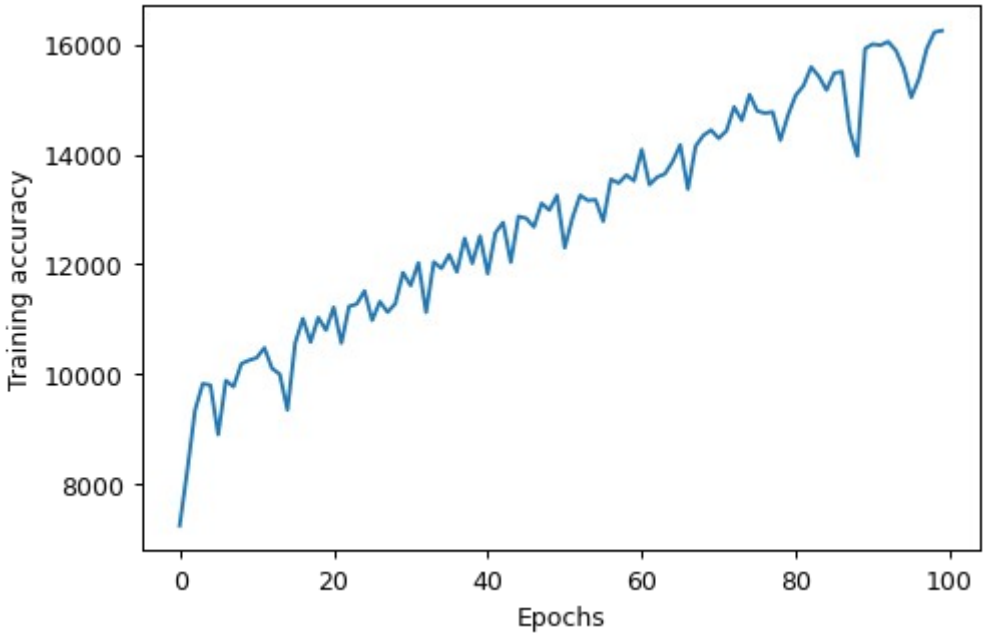
vez maiores, as acurácias obtidas foram cada vez menores onde as máximas praticamente não chegaram a alcançar 2800 emoções classificadas corretamente.

4.3.3 *Epochs* e *overfitting*

Quanto maior o número de *epochs* mais a função de custo se aproximará do seu mínimo local, o que não implica necessariamente em uma melhor performance quando aplicada em dados desconhecidos. O fenômeno *overfitting* é um problema tentador que pode levar a crença de que quanto mais *epochs* melhor a rede performará, se for analisada apenas a acurácia nos dados de treino.

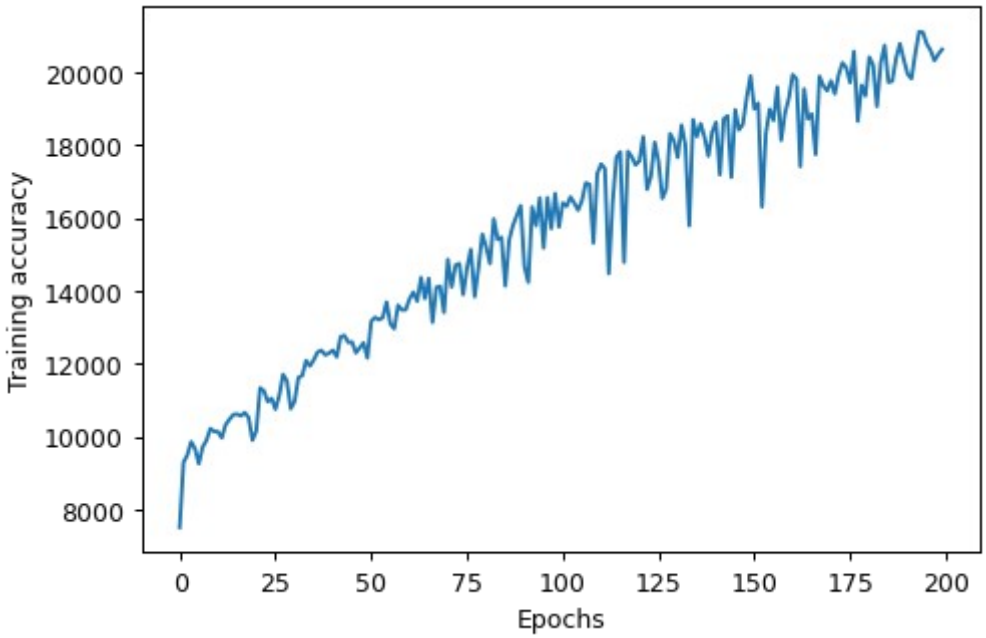
Ao observar a acurácia neste conjunto, os resultados são incrivelmente animadores, as figuras 28 e 29 mostram os resultados obtidos para a 1ª e 2ª redes da segunda leva de treino, respectivamente.

Figura 28 - Gráfico training accuracy vs epochs 1ª Rede, segunda leva.



Fonte: Elaborado pelo autor.

Figura 29 - Gráfico training accuracy vs epochs 2ª Rede, segunda leva.



Fonte: Elaborado pelo autor.

O número de emoções classificadas corretamente no conjunto de treino não mostra tendência alguma em parar de aumentar e, de fato, quanto maior o número de *epochs* maior será o número de imagens classificadas corretamente. Porém, como discutido antes, a acurácia no conjunto de testes apresenta uma estabilização iniciada no máximo por volta das 50 *epochs*.

Como dito na seção 2.8.2, todo o treino realizado a partir do ponto que a acurácia de teste se estabiliza, na realidade, é prejudicial para a performance da rede. O que ela está fazendo aumentando o número de classificações corretas do conjunto de treino é apenas uma “memorização” do mesmo. Dessa forma, o indicado é parar o treinamento quanto antes for detectada a estabilização da acurácia no conjunto de testes.

5 CONCLUSÃO

Considerando todos os fatores apresentados anteriormente, a melhor rede obtida entre as 66 treinadas foi a rede com 100 neurônios por *hidden layer*, 0.05 de *learning rate*, *mini batch size* de 16 e 100 *epochs* de treinamento, coincidentemente classificada como 1ª rede da segunda leva.

Os fatores que levaram a essa conclusão são os que beneficiam uma melhor acurácia em um aspecto generalizado, que é o principal objetivo da área de *machine learning*. Entre estes fatores os mais decisivos foram a acurácia média no conjunto de testes de 47.17% (maior entre todas as testadas) e um baixo número de *epochs* frente às outras redes, o que leva a uma menor chance de *overfitting*.

Apesar de não serem satisfatórios o suficiente para uma aplicação prática no reconhecimento de emoções, os resultados obtidos neste trabalho podem ser comparados aos resultados obtidos por Nordén (2019) de 61.1% na acurácia máxima com o uso de redes neurais não convolucionais. Alem disso estes resultados evidenciam pontos interessantes com relação a influência do conjunto de dados e hiperparâmetros utilizados ao se trabalhar com redes neurais, também mostra uma clara evolução na capacidade do aprendizado das mesmas quando considerados os hiperparâmetros compatíveis com a arquitetura utilizada. Redes neurais convolucionais apresentam resultados muito superiores, como os apresentados por Khaireddin (2021) de 73.28% de acurácia sobre o conjunto de dados de teste da base FER-2013.

A importância de aplicações como a abordada neste trabalho é evidenciada pelos sistemas de reconhecimento de emoções em desenvolvimento na atualidade, com finalidades diversas. Alguns estão sendo implementados em carros autônomos para garantir a segurança dos motoristas (Jain, 2021), outros estão sendo aplicados de maneiras coercivas em cidadãos chineses (Wakefield, 2021 e Standaert, 2021).

Tais implementações levam a discussões cada vez mais complexas sobre o uso correto da inteligência artificial e apenas reforçam a importância da interação crescente entre pessoas e máquinas. Sendo o reconhecimento de emoções apenas um dos usos da inteligência artificial como ferramenta para, esperançosamente, possibilitar um futuro benéfico para a humanidade.

6 REFERÊNCIAS

AFLAK, O. **Neural Network from scratch in Python**: Make your own machine learning library. towards data science, 2018. Disponível em: <https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce65> Acesso em: 27 abril 2021.

ANYOHA, R. **The history of artificial Intelligence**, sitn.hms.harvard.edu, 2017. Disponível em: <https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>. Acesso em: 21 maio 2021.

BENGIO, Y. **Practical Recommendations for Gradient-Based Training of Deep Architectures**. arvix, 2012. Disponível em: <https://arxiv.org/pdf/1206.5533.pdf>. Acesso em: 30 maio 2021.

BOTTOU, L. **Stochastic Gradient Descent Tricks**. Microsoft, 2012. Disponível em: <https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/tricks-2012.pdf>. Acesso em: 29 maio 2021.

BROWNLEE, J. **How to Configure the Number of Layers and Nodes in a Neural Network**. machinelearningmastery, 2018. Disponível em: <https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/>. Acesso em 30 maio 2021.

BROWNLEE, J. **3 Ways to Encode Categorical Variables for Deep Learning.** machinelearningmastery, 2019. Disponível em: <https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/>. Acesso em: 06 junho 2021.

CALEBE, F.; AMARAL, R. **Redes Neurais Multicamadas para Classificação e Regressão**. RPubs, 2019. Disponível em: <https://rpubs.com/CalebeF/534504>. Acesso em 23 maio 2021.

DECHTER, R. **LEARNING WHILE SEARCHING IN CONSTRAINT-SATISFACTION-PROBLEMS**. aaai.org, 1986. Disponível em: <https://www.aaai.org/Papers/AAAI/1986/AAAI86-029.pdf>. Acesso em 21 maio 2021.

DELUA, J. **Supervised vs. Unsupervised Learning: What's the Difference?** IBM Cloud, 2021. Disponível em: <https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning>. Acesso em 25 maio 2021.

DOBRZANSKI, M. D. **DeepLearningPython**. GitHub, 2016. Disponível em: <https://github.com/MichalDanielDobrzanski/DeepLearningPython/blob/master/network2.py>. Acesso em 16 maio 2021.

EARNSHAW, B. **A brief survey of tensors.** Slideshare, 2017. Disponível em: <https://www.slideshare.net/BertonEarnshaw/a-brief-survey-of-tensors>. Acesso em 27 maio 2021.

EKMAN, P.; FRIESEN, W. V.; TOMKINS, S. S. **Facial Affect Scoring Technique**: A First Validity Study. Paulekman, 1971. Disponível em: <https://www.paulekman.com/wp-content/uploads/2013/07/Facial-Affect-Scoring-Technique-A-First-Validity-Study.pdf>. Acesso em: 22 maio 2021.

ENCODE. *In:* DICIO, Oxford Learner's Dictionaries. Oxford University Press, 2021. Disponível em: <https://www.oxfordlearnersdictionaries.com/us/definition/american_english/encode> Acesso em: 06 junho 2021.

FEI-FEI, L.; KAI, L.; SOCHER, R.; DONG, W.; DENG, J. **ImageNet**: A Large-Scale Hierarchical Image Database. arvix, 2009. Disponível em: <https://arxiv.org/abs/1409.0575.pdf>. Acesso em: 21 maio 2021.

Google Colaboratory. Google, https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index#scrollTo=lSrWNr3MuFUS

Gradient descent. Khan Academy. Disponível em: <https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent>. Acesso em: 30 maio 2021.

IZARD, C. E. **The many meanings/aspects of emotion**: Emotion definitions, functions, activation, and regulation. Emotion Review, 2, 363-370, 2010. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.893.7800&rep=rep1&type=pdf>. Acesso em: 22 maio 2021.

JAIN, R. **Affectiva, an emotion-detection tech startup, acquired for $73.5 million**. itmunch, 2021. Disponível em: <https://itmunch.com/affectiva-acquired-for-73-5-million-what-is-emotion-detection/> . Acesso em: 20 junho 2021.

JONES, B. W. **Evolution of Sight in the Animal Kingdom.** Webvision, 2014. Disponível em: <https://webvision.med.utah.edu/2014/07/evolution-of-sight-in-the-animal-kingdom/>. Acesso em: 21 maio 2021.

KAGGLE. **FER-2013**: Learn facial expressions from an image. Kaggle, 2020. Disponível em: <https://www.kaggle.com/msambare/fer2013>. Acesso em: 22 maio 2021.

KESKAR, N. S. et al. **On Large-Batch Training For Deep Learning: Generalization Gap And Sharp Minima**. arxiv, 2017. Disponível em: <https://arxiv.org/pdf/1609.04836.pdf>. Acesso em 30 maio 2021.

KHAIREDDIN, Y.; CHEN, Z. **Facial Emotion Recognition: State of the Art Performance on FER2013**. arxiv, 2021. Disponível em: <https://arxiv.org/pdf/2105.03588v1.pdf>. Acesso em: 20 junho 2021.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. **ImageNet Classification with Deep Convolutional Neural Networks**. Papers, 2012. Disponível em: <https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>. Acesso em: 21 maio 2021.

LANGLOTZ, P. C. et al. **A Roadmap for Foundational Research on Artificial Intelligence in Medical Imaging.** researchgate, 2018. Disponível em: <https://www.researchgate.net/publication/332452649_A_Roadmap_for_Foundational_Research_on_Artificial_Intelligence_in_Medical_Imaging_From_the_2018_NIHRSNAACRThe_Academy_Workshop>. Acesso em: 21 maio 2021.

MATTHEW, J. A.; FLANAGAN, B. **Optimization Techniques to Improve Inference Performance of a Forward Propagating Neural Network on an FPGA**. 2020.

MCCARTHY, J. **What is Artificial Intelligence?**. homes.di.unimi.it, 2004. Disponível em: <https://homes.di.unimi.it/borghese/Teaching/AdvancedIntelligentSystems/Old/IntelligentSystems_2008_2009/Old/IntelligentSystems_2005_2006/Documents/Symbolic/04_McCarthy_whatisai.pdf>. Acesso em 21 maio 2021.

MILLION, E. **The Hadamard Product**. buzzard.ups.edu, 2007. Disponível em: <http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf>. Acesso em: 26 maio 2021.

MOINDROT, O. **CS 131 Lecture 1:** Course introduction. Github, 2018. Disponível em: <https://github.com/StanfordVL/cs131_notes/blob/master/lecture01/lecture01.pdf>. Acesso em: 21 maio 2021.

NIELSEN, M. A. **Neural Networks and Deep Learning**, Determination Press, 2015.

NORDÉN, F.; MARLEVI, F. V. R. **A Comparative Analysis of Machine Learning Algorithms in Binary Facial Expression Recognition**. diva-portal, 2019. Disponível em: <http://www.diva-portal.org/smash/get/diva2:1329976/FULLTEXT01.pdf> . Acesso em: 30 maio 2021.

ROSENBLATT, F. **Principles of Neurodynamics**: Perceptrons and the Theory of Brain Mechanisms. Cornell Aeronautical Laboratory, INC, 1961. Disponível em: <https://apps.dtic.mil/sti/pdfs/AD0256582.pdf>. Acesso em 22 maio 2021.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. **Learning representations by back-propagating errors.** Nature, Vol 323, p 533 - 536, 1986. Disponível em: <http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf>. Acesso em: 25 maio 2021.

SAMUEL, A. L. **Some Studies in Machine Learning Using the Game of Checkers**. IBM Journal, Vol. 3, 1959. Disponível em: <http://www2.stat.duke.edu/~sayan/R_stuff/Datamatters.key/Data/samuel_1959_B-95.pdf> Acesso em 21 maio 2021.

SANDERSON, G. **Gradient descent, how do neural networks learn | Chapter 2, deep learning**. Youtube, 2017. Disponível em: <https://www.youtube.com/watch?v=IHZwWFHWa-w&t=965s>. Acesso em 25 maio 2021.

SEYMOUR, A. P. **The summer vision project**. people.csail.mit.edu, 1966. Disponível em: <http://people.csail.mit.edu/brooks/idocs/AIM-100.pdf>. Acesso em 21 maio 2021.

STANDAERT, M. **Smile for the camera**: the dark side of China's emotion-recognition tech. The Guardian, 2021. Disponível em: <https://www.theguardian.com/global-development/2021/mar/03/china-positive-energy-emotion-surveillance-recognition-tech> . Acesso em: 20 junho 2021.

WAKEFIELD, J. **AI emotion-detection software tested on Uyghurs.** BBC, 2021. Disponível em: <https://www.bbc.com/news/technology-57101248>. Acesso em: 20 junho 2021.

WAKEFIELD, J. **Elon Musk's Neuralink 'shows monkey playing Pong with mind'**. bbc, 2021. Disponível em: <https://www.bbc.com/news/technology-56688812>. Acesso em: 22 maio 2021.

What is Artificial Intelligence (AI)? IBM Cloud Education, 2020. Disponível em: <https://www.ibm.com/cloud/learn/what-is-artificial-intelligence>. Acesso em: 22 maio 2021

YAN-TAK, A. **Lecture 2 - Linear Regression and Gradient Descent | Stanford CS229: Machine Learning (Autumn 2018).** Youtube, 2018. Disponível em: <https://www.youtube.com/watch?v=4b4MUYve_U8&list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU&index=10>. Acesso em: 19 abril 2021.

ZUFFO, M. **Acreditamos em uma relação simbiótica humano-máquina**. jornal.usp, 2020. Disponível em: <https://jornal.usp.br/ciencias/acreditamos-em-um-relacao-simbiotica-humano-maquina/>. Acesso em 22 maio 2021.