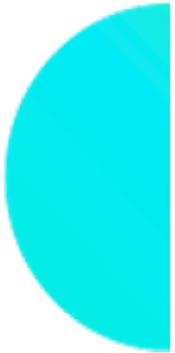


FUTURO DO TRABALHO, TRABALHO DO FUTURO

NEURAL NETWORK

Apostila do Aluno



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÕES



Adson Nogueira Alves - Analista Capacitação Técnica

Autor

Larissa Jessica Alves - Analista de suporte Pedagógico

Revisão da apostila

FIT Instituto de Tecnologia

Sorocaba, novembro de 2021



Autor



Adson Nogueira Alves é técnico de Informática (2007), Graduação em Engenharia de Controle e Automação (2016), Mestrado em Engenharia Elétrica (2021), atualmente é doutorando em Ciências da Computação pela Universidade Estadual de Campinas - UNICAMP.

Áreas de estudo e atuação: Sistemas Embarcados, Robótica, Inteligência Artificial e Aprendizado de Máquinas, Sistemas distribuídos, Automação, Controle e Educação.

Mais informações:

Linkedin: [Linkedin](https://linkedin.com/in/adson-alves) ou acesse:
[http://linkedin.com/in/
adson-alves](https://linkedin.com/in/adson-alves).

Orcid: [Orcid](https://orcid.org/0000-0002-0201-740X) ou acesse:
[https://orcid.org/
0000-0002-0201-740X](https://orcid.org/0000-0002-0201-740X).

Lattes: [Lattes](https://lattes.cnpq.br/1742965582100820) ou acesse:
[https://lattes.cnpq.br/
1742965582100820](https://lattes.cnpq.br/1742965582100820).

Github: [Github](https://github.com/AdsonNAlves) ou acesse:
<https://github.com/AdsonNAlves>.

Apresentação

Diversas inspirações que geraram desenvolvimento tecnológico surgiram de observações do nosso ambiente e da própria natureza, como é o caso do velcro inspirado na suíça pelo engenheiro Georges de Mestral. O engenheiro notou que pequenas sementes de uma planta, conhecida como bardana, ficavam presas nos pelos dos cachorros e em suas roupas, o sonar inspirado na observação dos golfinhos, o formato de turbinas inspirado nas nadadeiras das baleias, entre outras. Segundo essa lógica, foi natural se inspirar no cérebro humano para o desenvolvimento de modelos conhecidos hoje como **Redes Neurais**.

Nesse contexto, essa apostila abordará as redes **Perceptron**, **Multilayer Perceptron (MLP)** e **Convolutional Neural Network (CNN)**.

Algumas áreas de atuação:

Um profissional com habilidades em redes neurais atinge vasta possibilidade de aplicação dentro de uma abordagem de Inteligência Artificial (IA) e Aprendizado de Máquina (ML), entre elas, podemos atuar com:

- Robótica Inteligente,
- Data Science,
- Big Data,
- Estratégias de negócios e investimentos,
- Sistemas Autônomos,
- Rastreamento e Identificação de objetos
- e diversas outras ...

O objetivo desse curso é apresentar algumas redes frequentemente utilizadas no contexto de aprendizado de máquina, inspirações e conceitos que levaram ao desenvolvimento das arquiteturas que serão apresentadas.

Bons estudos!

Sumário

1	Introdução	5
1.1	História	5
1.2	Biológico para Matemático	6
2	Neural Networks	11
2.1	Conceitos Iniciais	11
2.1.1	Propagação	12
2.1.2	Retro-propagação	13
2.1.3	Função de custo	15
2.1.4	Gradiente Descendente	16
2.2	Perceptron	19
2.3	Multi-layer Perceptron	20
2.4	Convolutional Neural Network (CNN)	22
3	Exemplos de Código	26
3.1	Exemplo Perceptron	26
3.2	Exemplo Multilayer-Perceptron	26
3.3	Exemplo Rede Neural Convolucional	28
4	Exercícios Avaliativo	30

1 Introdução

Nessa seção apresentaremos uma breve introdução do histórico da pesquisa de Redes Neurais, a inspiração biológica deste modelo e sua definição matemática.

1.1 História

Em 1943, o neurofisiologista Warren McCulloch e o matemático Walter Pitts publicaram um artigo sobre suposto funcionamento dos neurônios. Para isso, eles modelaram uma Rede Neural simples usando circuitos elétricos. Em 1949, Donald Hebb abordou o fato de que as vias neurais são fortalecidas a cada vez que são usadas, assim se dois nervos disparam ao mesmo tempo, a conexão entre eles é reforçada. Em 1950, foi simulado uma Rede Neural hipotética, passo dado por Nathaniel Rochester, da IBM, porém sem sucesso. Em 1959, Bernard Widrow e Marcian Hoff de Stanford desenvolveram modelos chamados **ADALINE** e **MADALINE**, que vêm de **Multiple ADaptive LINEar Elements**. O **ADALINE** foi desenvolvido para reconhecer padrões binários de forma que, se estivesse lendo bits de streaming de uma linha telefônica, pudesse prever o próximo bit. **MADALINE** foi a primeira Rede Neural aplicada a um problema do mundo real, usando um filtro adaptativo que elimina ecos nas linhas telefônicas.

Em 1962, Widrow Hoff desenvolveram um procedimento de aprendizagem que examina o valor antes que seja aplicado um peso de ajuste (ou seja, 0 ou 1) de acordo com a regra: **Mudança de peso = (valor da linha de pré-peso) * (erro / (número de entradas))**, baseado na ideia de atribuir um peso ao erro para distribuí-lo pela rede.

Tempo depois, um artigo foi escrito sugerindo que não poderia haver uma extensão da Rede Neural de camada única para uma Rede Neural de várias camadas. Assim, a primeira rede multicamadas foi desenvolvida em 1975, uma rede não supervisionada.

Em 1982, John Hopfield apresentou uma abordagem utilizando linhas bidirecionais, visto que anteriormente, as conexões entre os neurônios eram apenas uma via. Naquele mesmo ano, foi proposto uma "rede híbrida" com várias camadas.

Em 1982, houve um esforço para *Quinta Geração* em Redes Neurais, o que envolveria a Inteligência Artificial. Sendo que, a *primeira geração* utilizava interruptores e fios, a *segunda geração* transistores, a *terceira tecnologia* de estado sólido como circuitos integrados e linguagens de programação de alto nível e a *quarta geração* geradores de código.

Em 1986, dentro do cenário de Redes Neurais de múltiplas camadas, surge o que agora são chamadas de redes de Propagação Reversa, visto distribuem erros por toda a rede que, diferente das redes híbridas que utilizavam apenas duas camadas, essas redes de Propagação Reversa usam várias, porém o que resulta em um aprendizado lento, necessitando de milhares de iterações para aprender.

Atualmente, as Redes Neurais são usadas em várias aplicações, como sistemas autôno-

Fit-F.24.1.01-04 Rev. B

mos, predição de risco, robótica, comércio, entre outros. O grande gargalo sempre será poder de processamento de **hardware**, porém com todos os novos avanços dentro da computação, logo deixará de ser. A Figura 1, ilustra um pouco essa evolução.

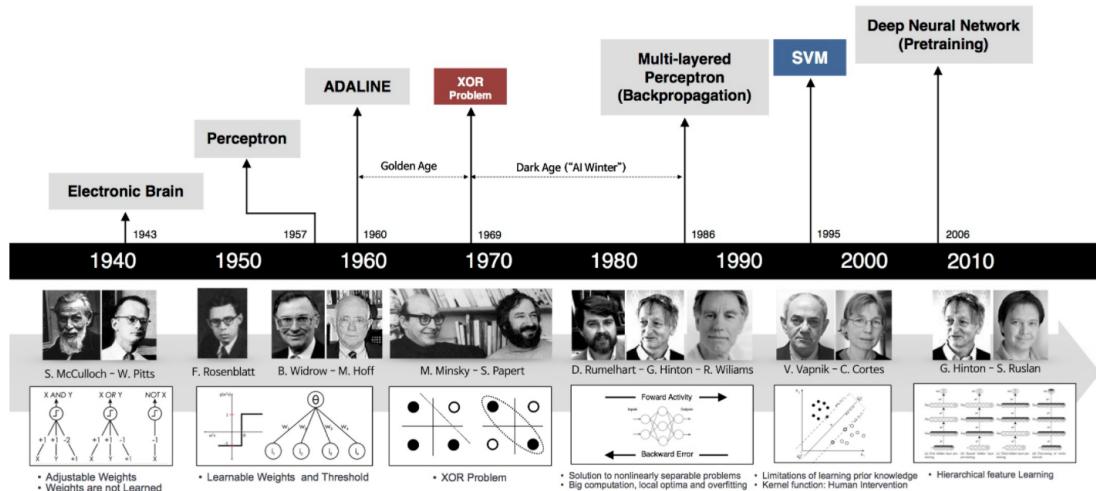


Figura 1: Evolução da pesquisa - Fonte: [8]

1.2 Biológico para Matemático

Uma Rede Neural Artificial (RNA) busca imitar o comportamento do cérebro humano. Uma RNA atinge centenas ou milhares de unidades de processamento; o que não representa tanto, quando comparamos com o cérebro de um mamífero que pode conter muitos bilhões de neurônios, a Figura 2 ilustra esse neurônio. O sistema nervoso é formado por um conjunto extremamente complexo de células, os neurônios, que são fundamentais na determinação do funcionamento e comportamento do corpo humano e do raciocínio, sendo formados pelos dendritos, que são um conjunto de terminais de entrada, pelo corpo central, e pelos axônios que são longos terminais de saída. Não é nosso objetivo explicar o funcionamento biológico do neurônio, mas sim trazer para o nosso cenário da Inteligência Artificial. Os neurônios quando interligados, tem como objetivo transmitir uma informação ao próximo gerando um determinado comportamento biológico.

Em Redes Neurais Artificiais buscamos imitar esse comportamento aplicando uma analogia a esse modelo, de forma que as entradas da RNA (similar aos dendritos) quando estimuladas, transmitam uma determinada informação (similar aos axônios), ponderando a informação para determinar um estado de saída que melhor atenda o propósito do estímulo. Como exemplo, quando se é estimulado o estado de dor, esperamos um comportamento de fulga ou outro mais adequado.

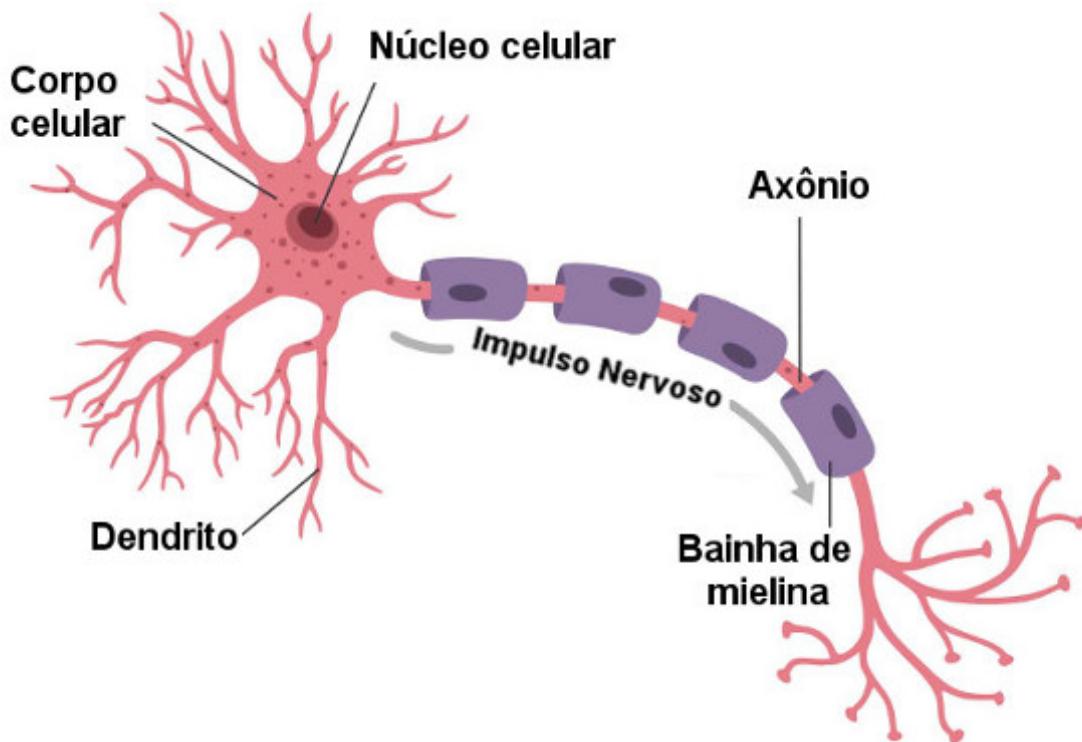


Figura 2: Principais partes do neurônio - Fonte: [10]

Um modelo matemático da representação do modelo, pode ser vista na Figura 3, onde as entradas do neurônio são representadas pelo valor $[x_1 \dots x_n]$, na sequência as entradas são ponderadas com pesos $[\omega_1 \dots \omega_n]$, aplicando ao final um somatório. É apresentado um *bias*, junto ao somatório, que pode ser entendido como um comportamento proposital para convergência do modelo, na sequência é aplicada um função de transferência, responsável por transmitir esses dados ao próximo neurônio ou gerar a informação de saída (y).

Dessa forma, a junção de vários desses neurônios é o que chamamos de Rede Neural.

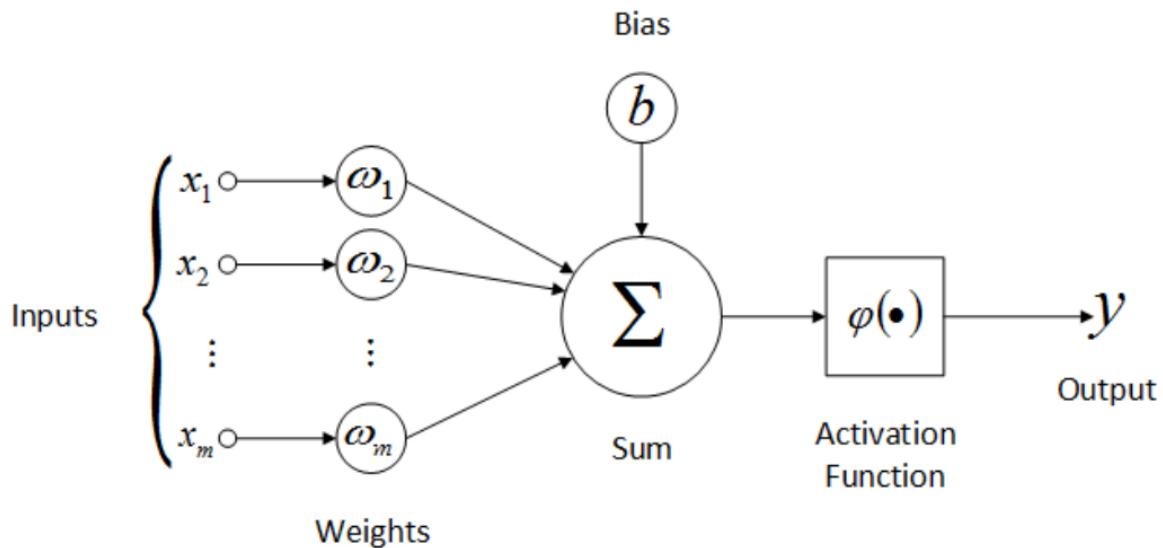


Figura 3: Representação do neurônio - Fonte: [6]

Em 1957 surge o **Perceptron** proposto por Frank Rosenblatt, com a proposta de aplicar o aprendizado através de uma soma ponderada das entradas que ao final é aplicado uma função de ativação afim de determinar se ativa ou não ativa o neurônio. A partir dos resultados desta arquitetura surgiram diversas outras, diferenciando essencialmente em número de neurônios e camadas. Todos os termos ficarão mais claros a seguir, além disso, essas arquiteturas diferem em tipo de aplicação e objetivo da rede. A Figura 4 apresenta algumas de várias arquiteturas consolidadas dentro da abordagem de Redes Neurais Artificiais (RNA).

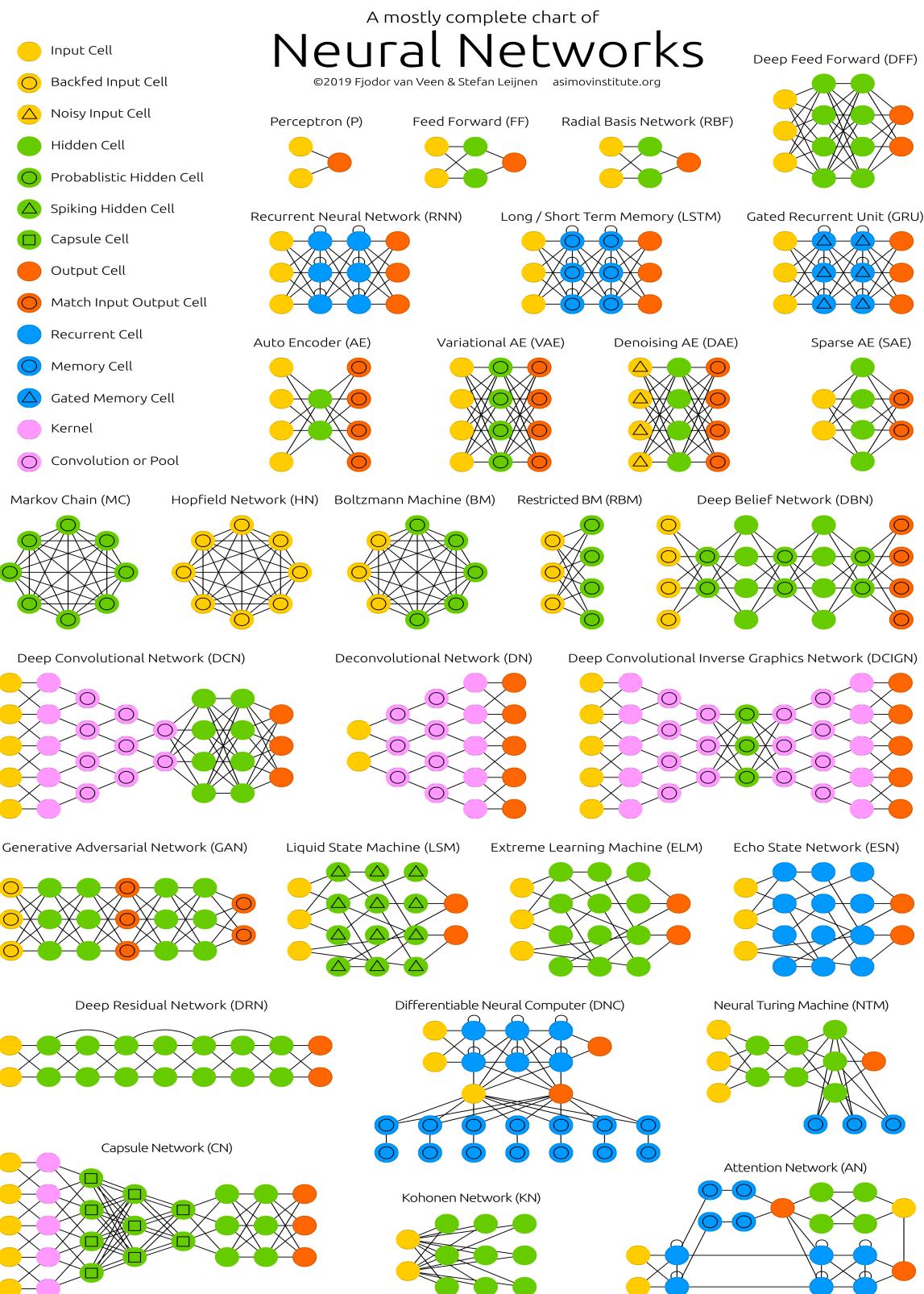


Figura 4: Algumas Arquiteturas de RNA - Fonte: [5]

Nas próximas seções exploraremos algumas arquiteturas de Redes Neurais importantes para o correto aprendizado da técnica, como o **Perceptron** e **Mulilayer-Perceptron**. Na sequência outra tradicional chamada **Convolutional Neural Network (CNN)**.

2 Neural Networks

Nesta seção será apresentado a base teórica das técnicas propostas, bem como as *arquiteturas* das redes.

2.1 Conceitos Iniciais

Nesta subseção falaremos de alguns conceitos importantes dentro do contexto de Rede Neural (RN), use como referência e consulte sempre que necessário. Apresentaremos alguns conceitos e nomenclaturas como: **Estrutura da Rede Neural, Propagação, Retropropagação, Gradiente, Função de custo, Otimizadores** entre outras. Estrutura da Rede Neural

A Figura 5 apresenta um exemplo de RN simples, nela temos 3 camadas: a primeira chamamos de camada de entrada (com 4 quatro neurônios - {A,B,C,D}), a última é chamada de camada de saída (com 1 neurônio) e todas as camadas que estiverem entre a camada de entrada e a camada de saída é chamada de camada oculta, no nosso exemplo, temos apenas 1 (uma) camada oculta com 5 neurônios. Observe que cada entrada $x_1 \dots x_n$ representa uma característica / atributo do problema que está sendo analisado. Note que todos os neurônios de uma camada estão conectados aos neurônios da anterior isso chamamos de camada densa, comumente usadas em RN's. Estas conexões são feitas pelos *links* ou setas apresentadas na Figura 5.

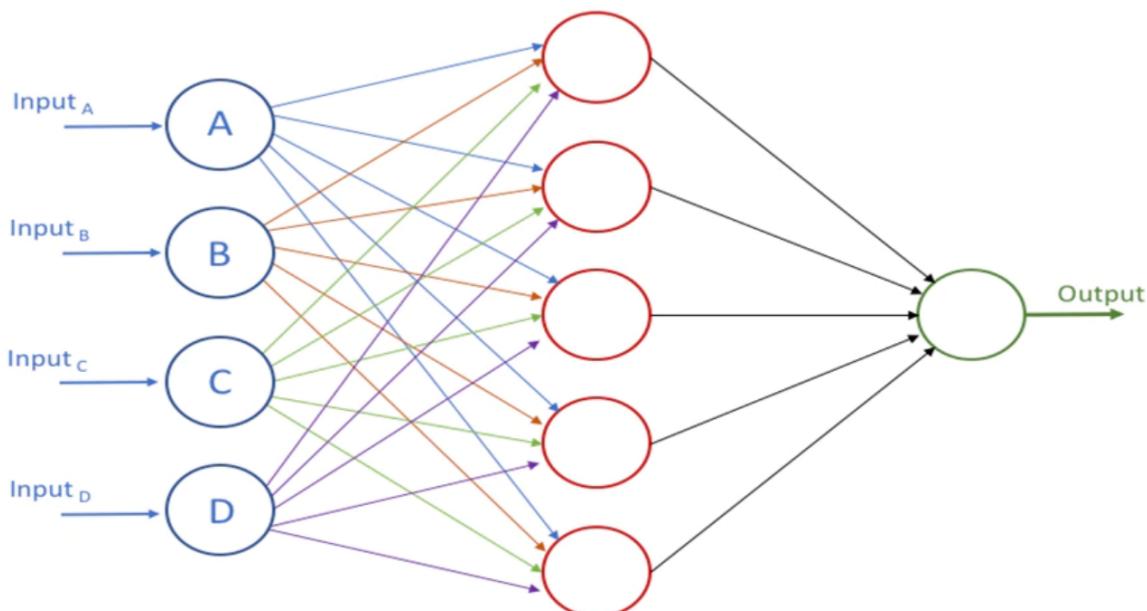


Figura 5: Rede Neural Simples - Fonte: [11]

Apesar de não aparecer na Figura 5, cada *link* tem um atributo que pode ser chamado de *peso*, no qual esse conjunto forma uma matriz de pesos controlados, que devem ser

Fit-F.24.1.01-04 Rev. B



aprendidos pela Rede Neural. Pode aparecer neste modelo um *bias*, que como dito anteriormente, é estímulo proposital que faz com que o modelo converja para o *estado esperado*. Ele é representado como um novo neurônio, existente para todas as camadas ocultas e a camada de saída, porém não precisa necessariamente conter o mesmo *atributo*, podendo variar entre as camadas.

2.1.1 Propagação

Em Propagação, no contexto de RN, estamos buscando representar a propagação de toda a informação de entrada da rede até o final, ou seja, a saída da Rede Neural. Para isso observe a Figura 6, a camada de entrada é representada por x_1, x_2, x_3 , a camada oculta por $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$ e a camada de saída por $a_1^{(3)}$, logo a saída da rede será $h_w(x)$.

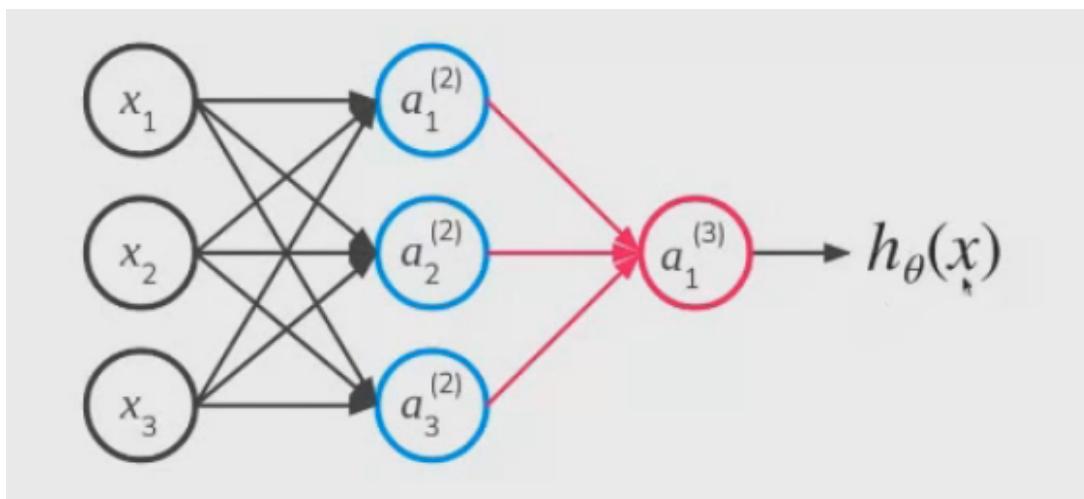


Figura 6: Propagação - Fonte: [12]

Considerando que a matriz do *peso* da rede é representado pelo índice $w_{i,j}$, chegamos nas seguintes equações para a camada oculta:

$$a_1^{(2)} = g(w_{10}^{(1)}x_0 + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(w_{20}^{(1)}x_0 + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(w_{30}^{(1)}x_0 + w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3)$$

Com isso a camada de saída é dada pela Equação 1. Dessa forma, a propagação da RN também pode ser definida pela mesma equação, considere θ como w .

$$h_w(x) = a_1^{(3)} = g(w_{10}^{(2)}a_0^{(2)} + w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)}) \quad (1)$$

2.1.2 Retro-propagação

A Retro-Propagação é baseado no erro encontrado em cada neurônio da última camada. Assim é aplicado a retro-propagação do erro até a camada de entrada da rede por meio da derivada desses erros (Regra da cadeia).

Considere a Rede Neural proposta por [1] apresentada na Figura 7, todas as camadas possuem 2 neurônios sendo a camada de entrada representada por i_i , camada oculta h_i e camada de saída o_i . Todos os parâmetros de pesos e o valor esperado de saída y_i são informados.

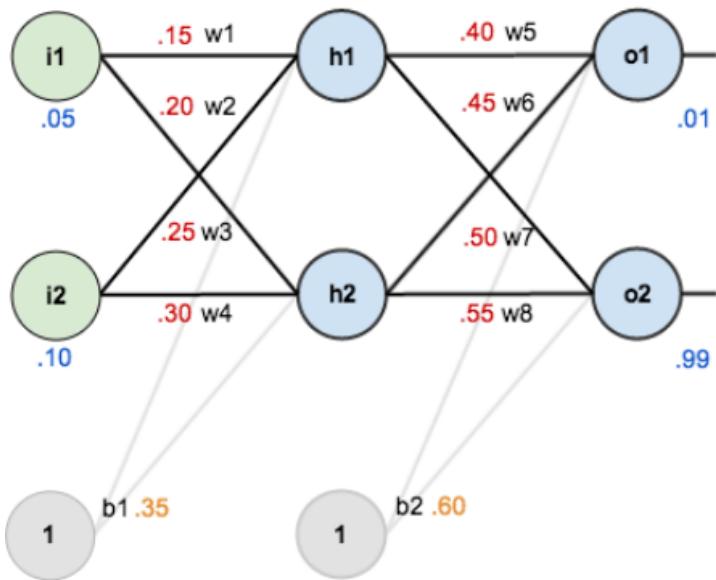


Figura 7: Rede Neural (BackPropagation) - Fonte: [1]

Para exemplificar a Retro-Propagação, iremos realizar a propagação das entradas conforme mostrado nas equações a seguir:

Entrada do neurônio h_1

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

Saída do neurônio h_1 e h_2 , através da função logística de transferência.

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

Entrada do neurônio o_1

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

Saída do neurônio o_1 e o_2 , através da função logística de transferência.

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$

Calculamos o erro de saída de o_1 e o_2

$$E_{o_1} = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o_2} = 0.023560026$$

$$E_{total} = 0.274811083 + 0.023560026 = 0.298371109$$

Assim para saber o quanto w_5 afetou o erro total, aplicamos a regra da cadeia, dada por:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

A Figura 8 ilustra o fluxo da Retro-Propagação.

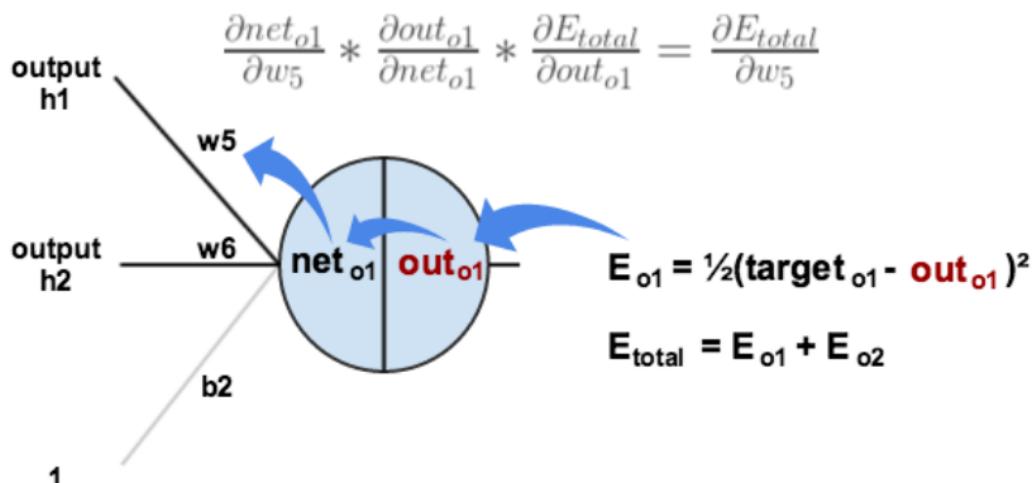


Figura 8: Fluxo BackPropagation - Fonte: [1]

Assim:

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * (-1) + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

Seguindo temos que:

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507 * (1 - 0.75136507) = 0.186815602$$

Para o último termo temos:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{1-1} + 0 + 0 = out_{h1} = 0.593269992$$

Finalmente temos:

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

O ajuste de w_5 é feito a partir da equação 2.

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648 \quad (2)$$

Repetindo o processo para todos os outros encontraremos:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

De forma similar é retro propagado para $w_1^+, w_2^+, w_3^+, w_4^+$ resultando em um:

$$w_1^+ = 0.149780716$$

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Assim, repetindo o ciclo de Propagação e Retro-Propagação os ajustes são feitos em todos os parâmetros de peso, resultando após algumas iterações a um $h_w(x) \approx y_i$.

2.1.3 Função de custo

A função de custo esta relacionada ao erro encontrado entre o resultado que a rede está predizendo e qual é o valor real daquele dado. Uma abordagem tradicional seria a abordagem do erro quadrático médio, erro absoluto médio ou erro absoluto médio ponderado. A função de custo sempre dependerá do problema que estiver sendo abordado, logo, para exemplificar a função ilustremos um problema de regressão, onde o objetivo seria encontrar uma função linear que melhor representa um conjunto de dado como na Figura 9, onde a função em vermelho representaria essa regressão.

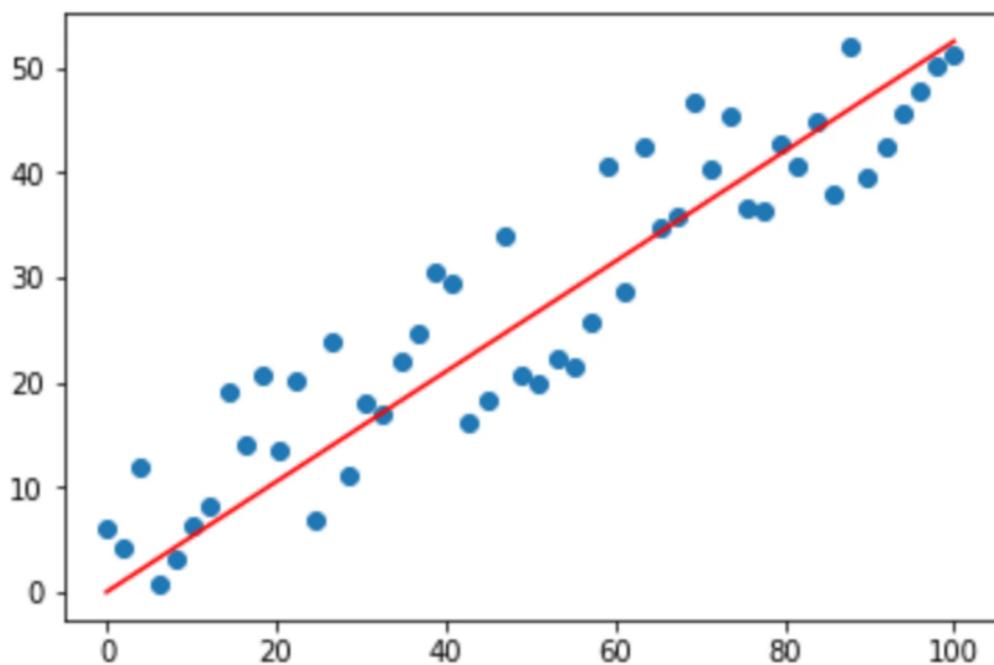


Figura 9: Regressão linear - Fonte: [9]

Sabemos que uma função Linear pode ser descrita como $y = ax + b$ que passando



para o contexto de Redes Neurais chegamos em $h_w(x) = w_1x + w_0$. O objetivo da Função de custo é encontrar os melhores valores para w_0 e w_1 que melhor descrevem a função linear. Assim, aplicando a abordagem do erro quadrático médio chegaremos a nossa função de custo $J(w_0, w_1)$. Vale relembrar que uma função que aborda esta problematização é apresentado pela Equação 3, onde m é o numero de amostras e $y^{(i)}$ o valor referênciâa.

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (h_w(x^{(i)}) - y^{(i)})^2 \quad (3)$$

A Equação 3 pode ser representada graficamente pela Figura 10, novamente considere θ como w . Observe que o objetivo do aprendizado é diminuir a função custo, o que graficamente significa chegar no ponto mais baixo da curva, onde $J(w_0, w_1) \approx 0$.

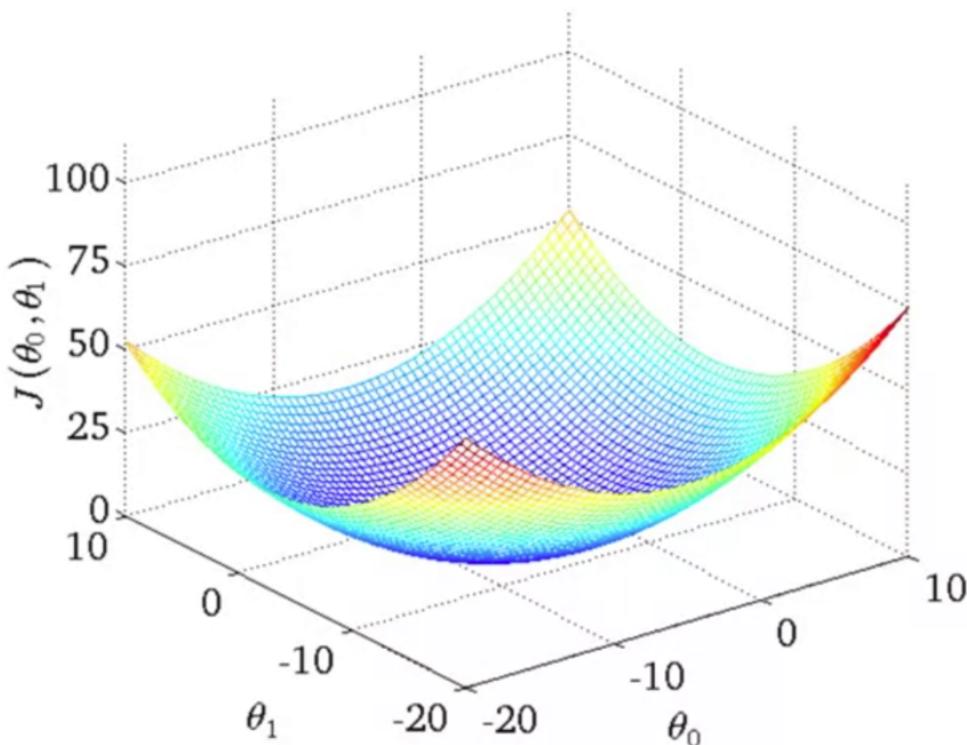


Figura 10: Gráfico da Função Custo Convexa - Fonte: [12]

2.1.4 Gradiente Descendente

O Gradiente Descendente também conhecido como Decida Gradiente é a técnica mais usada quando queremos encontrar nossos parâmetros, o que resulta na minimização da nossa função de custo, aproximando a função do seu mínimo, no caso da regressão linear, mostrada anteriormente, o nosso mínimo global. A Equação 4 apresenta a função de ajuste gradiente e a Figura 11 dois exemplos de iterações da Descida Gradiente até encontrar um mínimo, em

um espaço de vários mínimos locais.

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1) \quad (4)$$

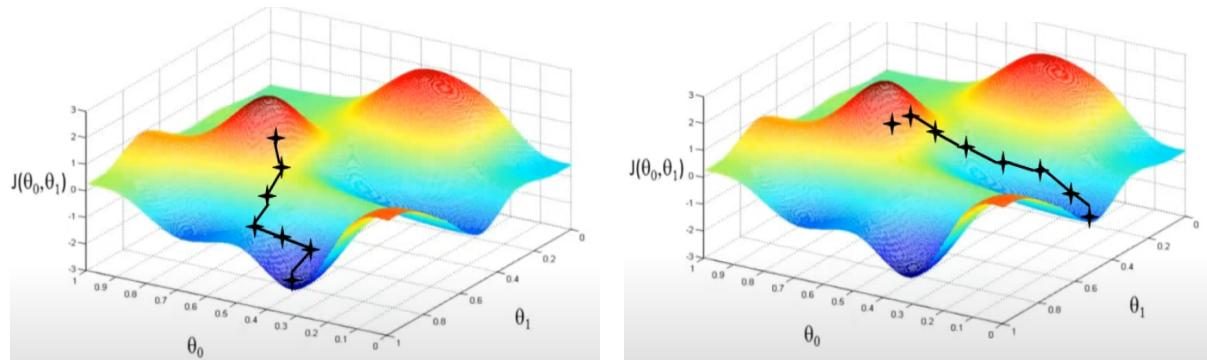


Figura 11: Exemplo Descidas gradiente - Fonte: [12]

Na equação α representa a taxa de aprendizado e $\frac{\partial}{\partial w_j} J(w_0, w_1)$ representa a derivada parcial da nossa função de custo. A taxa de aprendizado define o quanto será ajustado durante a Descida Gradiente. Na Figure 12 é apresentado 3 ajustes para a taxa de aprendizado, quando o valor é baixo o modelo demora muito tempo para convergir, quando aplicamos um valor médio conseguimos a taxa de aprendizado ideal, porém quando aplicamos um valor muito alto o modelo corre o risco de nunca convergir.

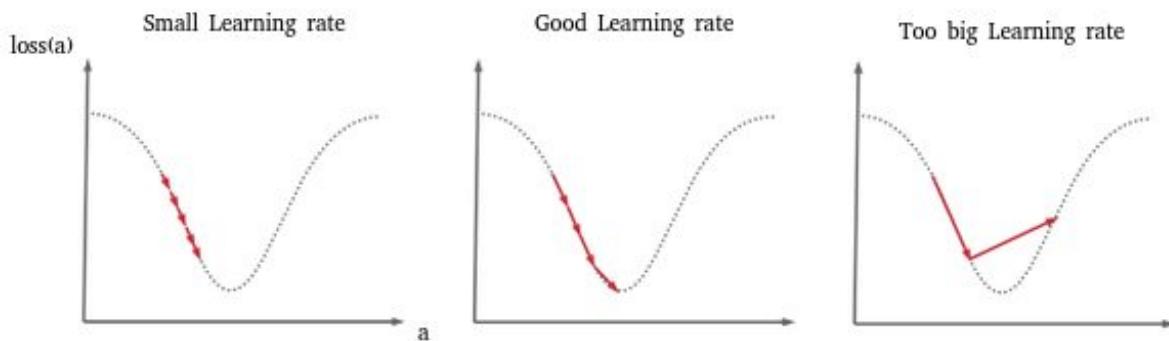


Figura 12: Taxa de Aprendizado - Fonte: [7]

Neste cenário, quando não conseguimos carregar todos os exemplos na memoria de treinamento, podemos aplicar o que chamamos de **gradiente descendente estocástico** ou **gradiente descendente mini-lotes**. A diferença está na forma que atualiza os parâmetros $w_{i,j}$, sendo que um atualiza sempre que chega um novo exemplo e o outro atualiza uma amostra do total de dados, por isso mini-lotes.

Temos também os **Otimizadores**, que por definição são responsáveis por atualizar os

parâmetros de peso para minimizar a função de custo. Assim o **Gradiente Descendente**, **Gradiente Descendente Estocástico** e **Gradiente Descendente Mini-Lotes** são exemplos de otimizadores, porém não são os únicos, temos o **SGD with momentum**, **Nesterov Accelerated Gradient (NAG)**, **Adaptive Gradient (AdaGrad)**, **AdaDelta**, **RMSprop**, **Adam** entre outros. O nosso objetivo não é detalhar esses otimizadores, apenas apresentá-los como possíveis escolhas, cabe ao aluno explorar suas particularidades. O gráfico da Figura 13 apresenta uma comparação de alguns otimizadores, note que o melhor desempenho foi para o **Adam**.

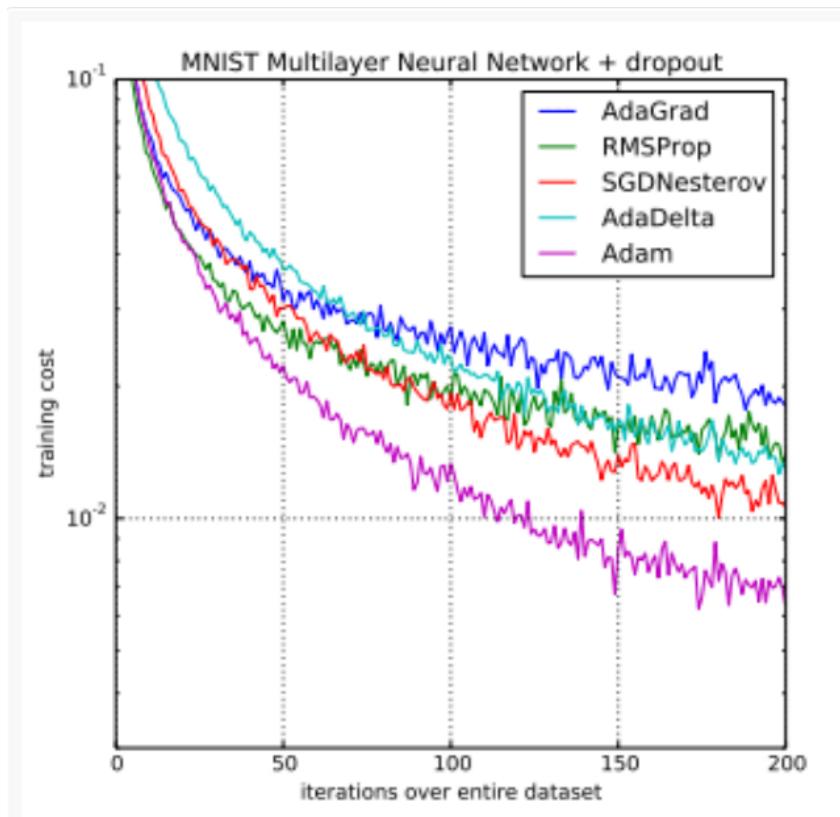


Figura 13: Comparação Otimizadores - Fonte: [1]

Outras definições

- **Overfitting**: modelo super ajustado, gerando perda de generalização quando aplicado a dados desconhecidos. Para evitar é necessário diminuir o tempo de treinamento.
- **Underfitting**: modelo pouco ajustado, gerando pouca contribuição na predição dos dados. Para evitar aumentamos o tempo de treinamento.
- **Aprendizado Supervisionado**: Existe a figura de um "Professor", durante todo processo de treinamento. Assim, sempre que a rede predizer uma saída $h_w(x)$ o algoritmo compara com o resultado esperado y_i , dizendo se está correto e qual foi a taxa de erro encontrada.

- **Aprendizado não Supervisionado:** Não existe a figura do "Professor", ou seja, o algorítimo é responsável por realizar toda a abstração dos dados e realizar uma classificação.
- **Stride:** Na CNN é o tamanho do passo do filtro de kernel utilizado na imagem original.
- **Padding:** Na CNN é quando adicionamos zeros em todo contorno da imagem original. Útil para aplicar diferentes filtros e centralizar no primeiro pixel.
- **Dilatação Convolucional:** Quando movimentamos cada valor do filtro da imagem, horizontalmente ou verticalmente, fazendo o kernel ser expandido para pixels diferentes na imagem.

2.2 Perceptron

Como introduzido na seção 1, o **Perceptron**, foi proposto por Frank Rosenblatt, com a proposta de aplicar o aprendizado por meio de uma soma ponderada das entradas. Assim ao final é aplicado uma função de ativação afim de determinar se ativa ou não ativa o neurônio, similar as sinapses que acontece no nosso cérebro. Na Figura 14, podemos ver o funcionamento com mais detalhes. A saída h_w é dada por uma função de ativação degrau que transmite o somatório de todas as entradas x multiplicadas pela transposta dos parâmetros w . Normalmente temos a função degrau dado por:

$$heaviside_z = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \quad \text{ou} \quad sign_z = \begin{cases} -1, & \text{if } z < 0 \\ 0, & \text{if } z = 0 \\ +1, & \text{if } z > 0 \end{cases}$$

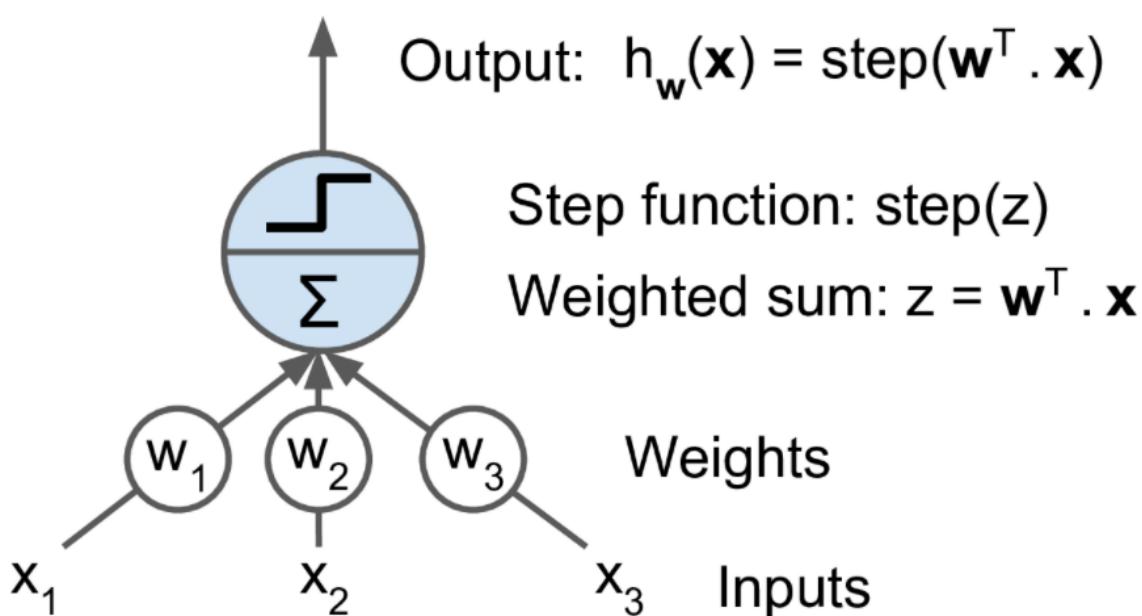
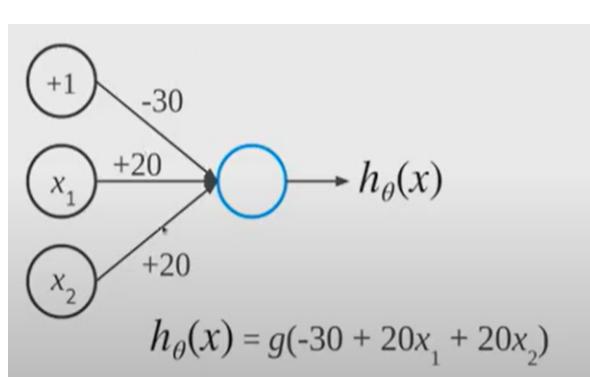


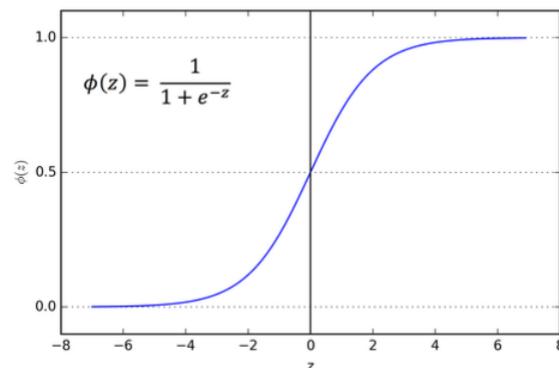
Figura 14: Linear threshold unit - Fonte: [4]

Assim, começou o interesse em modelar problemas pensando na ideia de neurônio, logo tentativas baseadas em portas digitais começaram a ser aplicadas como: AND e OR.

Suponha que tenhamos o Perceptron da Figura 15a e uma função de ativação sigmoide da Figura 15b conforme mostrados na Figura 15. Considerando que estamos trabalhando com um modelo AND onde $x_1, x_2 \in \{0, 1\}$ e $y = x_1 \text{ AND } x_2$, encontramos a tabela verdade mostrada na Tabela 1.



(a) Exemplo Perceptron



(b) Função Sigmóide

Figura 15: Exemplo Perceptron AND - Fonte: [12]

Tabela 1: Tabela Verdade - AND.

x_1	x_2	$h_0(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Veja um exemplo de aplicação na seção 3.1.

2.3 Multi-layer Perceptron

Em [13] é definido o **Multi-layer Perceptron (MLP)** como um algoritmo de aprendizado supervisionado que aprende uma função $f(\cdot) : R^m \rightarrow R^o$ a partir do treinamento de uma base de dados. Assim a variável m representa o número de dimensões de entrada e o é o número de dimensões de saída. Dado um grupo de *Features* $X = x_1 \dots x_m$ é uma *Target* y_i .

o MLP é capaz de aprender um aproximador de função não Linear que pode ser aplicado na classificação de dados ou regressão, o que é diferente de uma regressão logística que também trata problemas de classificação porém utiliza um esquema conhecido como *one-vs-rest(OvR)*. No caso do MLP podem existir *camadas* não lineares, que são as nossas camadas ocultas. Um exemplo do nosso multi-layer pode ser visto na Figura 16.

Observe, é apresentado nossa camada de entrada com o conjunto de dados de entrada $X = x_1 \dots x_m$, nossa camada oculta realiza a soma ponderada de todos os nossos dados de entrada, seguindo uma função de ativação não linear, como a tangente hiperbólica, sigmoide, relu entre outras, ao final é feita a transformação para um valor de saída, que até agora no texto chamamos de $h_w(x)$. Na tabela 2 podemos ver algumas vantagens e algumas desvantagens do MLP.

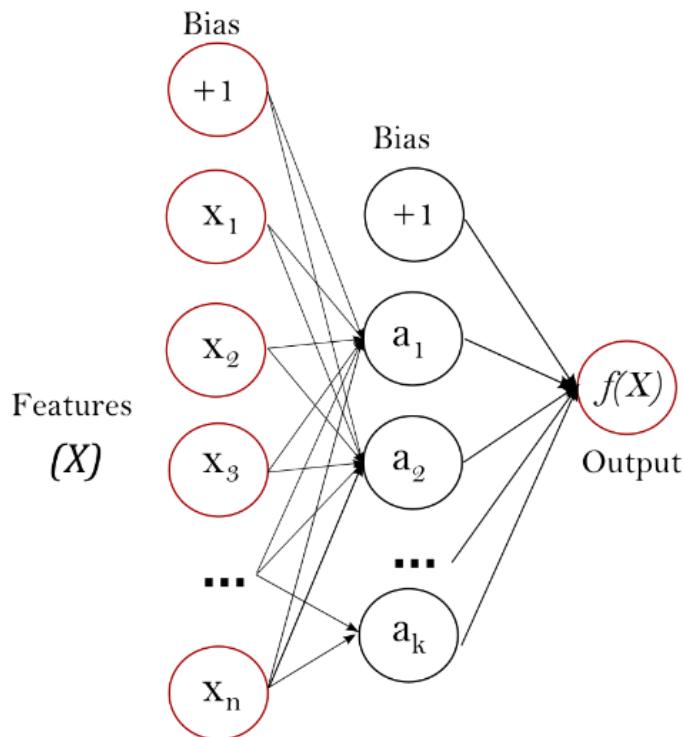


Figura 16: Multilayer Perceptron - Fonte: [13]

Tabela 2: Vantagens e Desvantagens do MLP

Multi-layer Perceptron	
Vantagens	Desvantagens
Aprender modelos não lineares Aprender modelos em tempo real	Vários Mínimos Locais Ajuste de hyper-parâmetros da rede Sensível à escala da Feature

Veja um exemplo de aplicação na seção 3.2.

2.4 Convolutional Neural Network (CNN)

Atualmente existem diversas arquiteturas de rede aplicadas ao modelo de CNN, nesse curso abordaremos a primeira arquitetura proposta em CNN.

Uma Rede Neural Convolucional (CNN), surgiu de questionamentos de como as imagens eram tratadas em um cenário de Rede Neural. Um deles é a necessidade de "achatar" a imagem em um Tensor de 1 dimensão, ou seja, transferir um dado 2D para 1D. Isso acarreta perda de abstração da estrutura espacial da imagem, além da quantidade de pesos e recurso de neurônios. Então vamos entender o que seria a convolução.

Convolução é o ato de adicionar cada elemento da imagem considerando sua vizinhança local e ponderando através de um filtro **kernel**. A Figura 17 apresenta o que seria essa convolução para uma imagem RGB. A convolução acontece *fatiando* a matriz da imagem original pela matriz do kernel, começando na extremidade superior esquerda e indo até o final, através de pequenos passos horizontais e verticais. Assim é realizado a soma da multiplicação dos pixels da imagem original e valores do kernel. A soma de cada canal dessa imagem é armazenada sequencialmente em uma nova matriz, levando a uma redução significativa desta imagem.

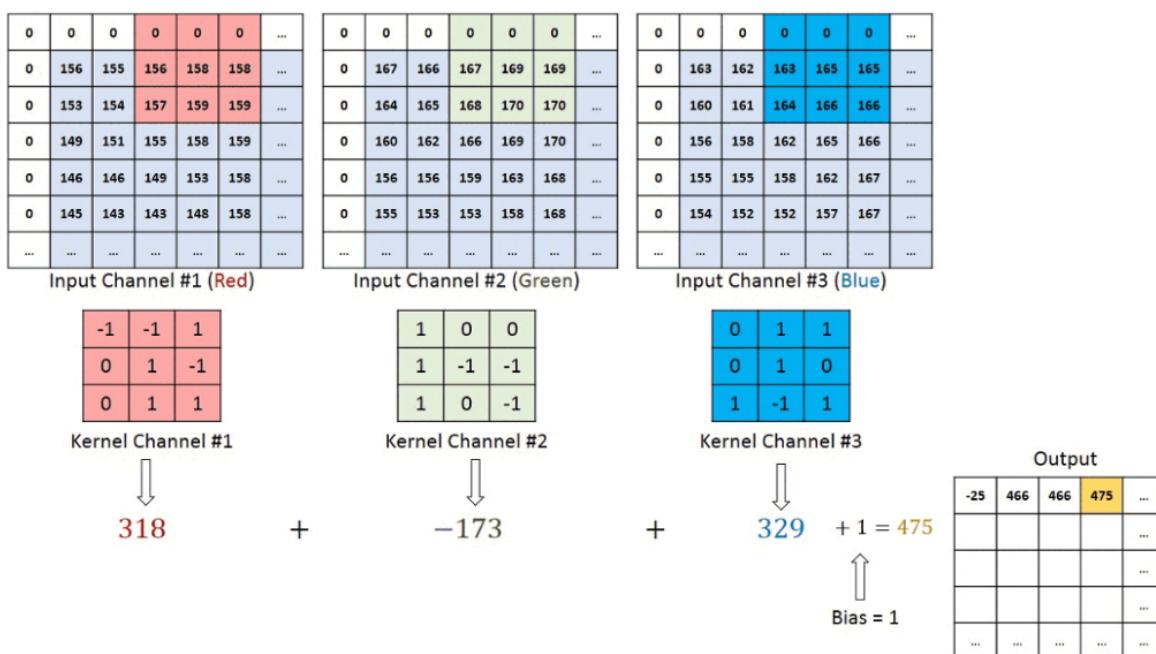


Figura 17: Convolução - Fonte: [2]

Cada filtro de kernel tem um efeito particular na imagem original, como: destacar bordas, borrar imagem, níveis de contorno entre outros. A Figura 18 ilustra o efeito de alguns filtros quando aplicado em uma imagem de cachorro.

Assim é parte do processo de aprendizagem de uma rede convolucional buscar compreender qual são os valores que devem existir em um filtro kernel para abstrair uma determinada característica da imagem original.

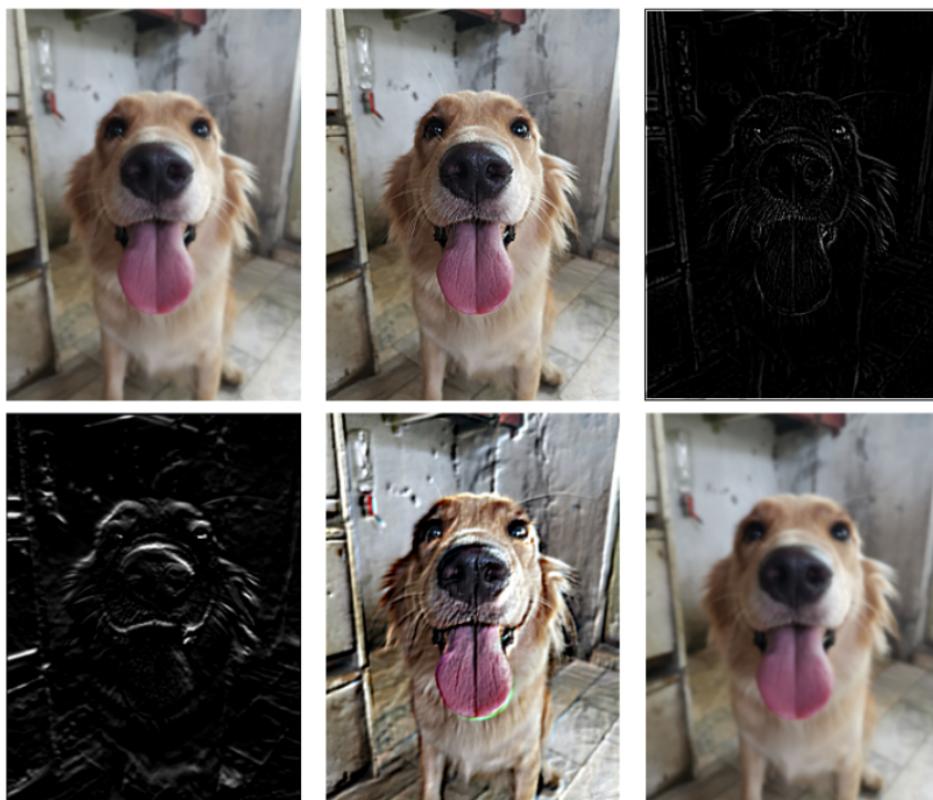


Figura 18: Filtro Kernel - Fonte: [3]

Na Figura 19 podemos verificar a arquitetura de uma CNN para dígitos manuscritos. A camada convolucional, é uma camada com vários filtros de convolução que devem respeitar o volume dos dados (exemplo o RGB com profundidade 3), similar ao que foi feito na Figura 17. Cada resultante desse processo é chamado de **mapa de ativação**, que tem uma dimensão que representa a resultante da multiplicação do filtro e da imagem original. Dessa forma, podemos aplicar diversos filtros para uma mesma imagem, o que gera vários mapas de ativação, levando a variar abstrações distintas da imagem, essa é a **Camada de convolução**. Além disso, podemos alterar parâmetros com stride, padding, função de ativação e dilatação convolucional.

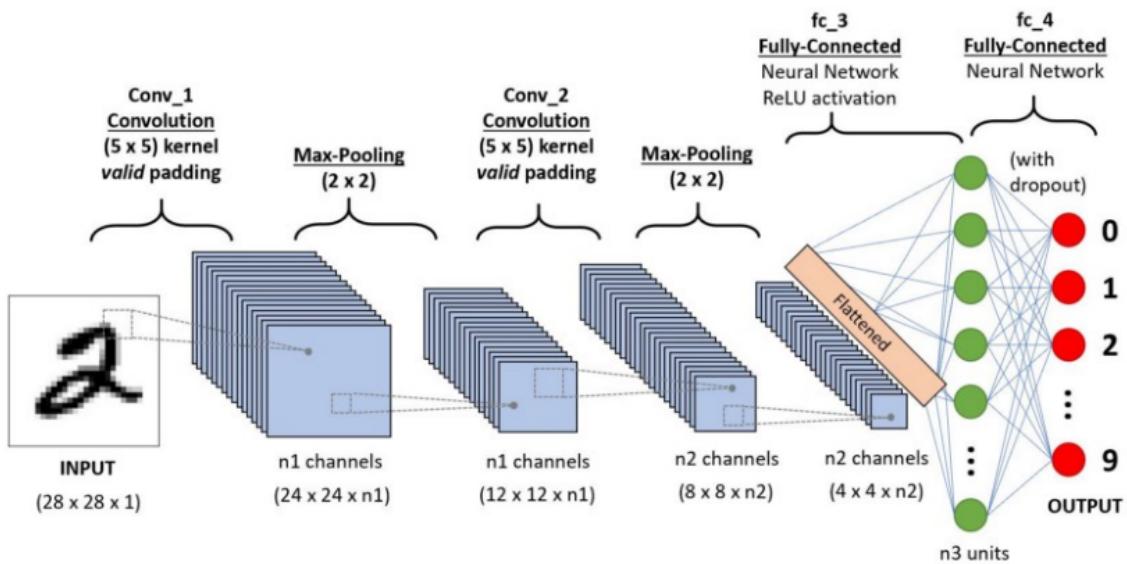


Figura 19: Arquitetura CNN para dígitos manuscritos - Fonte: [3]

A próxima camada é a **Camada de Pooling**, responsável por reduzir todos os canais de convolução, geralmente pela metade. Realizando operações individuais em cada mapa de ativação. Existem dois tipos de Pooling: **Pooling máximo** e **Pooling médio**. Max Pooling retorna o valor máximo da parte da imagem coberta pelo Kernel e Average Pooling retorna a média de todos os valores. Em comparação o Max Pooling tem um desempenho muito melhor do que o Average Pooling, devido uma vez que atua como um supressor de ruído, descartando ativações ruidosas junto com a redução de dimensionalidade. A Figura 20 apresenta as 2 abordagens.

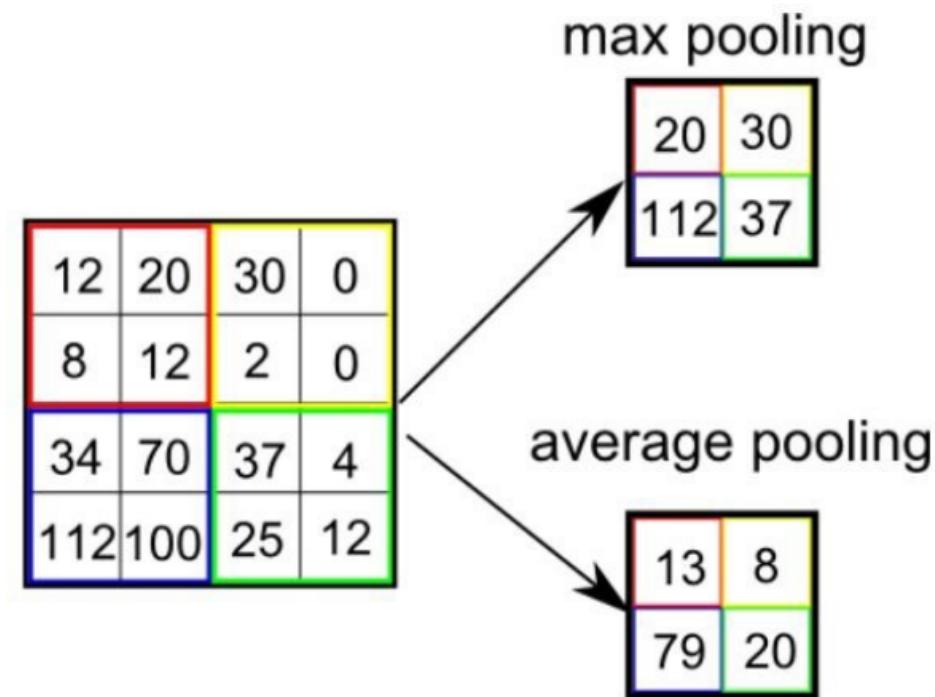


Figura 20: Pooling Layer - Fonte: [3]

A última camada chamada de **Fully-Connected** foi proposta na primeira versão da CNN, ela é responsável por "achatar", ou seja, colocar todos os dados em uma dimensão (1D) e conectá-los a todos os neurônios da camada de saída. Um exemplo seria uma rede de classificação onde teríamos 10 classes, todos os neurônios da camada fully-connected seriam conectados aos 10 neurônios da camada de saída.

Veja um exemplo de aplicação na seção [3.3](#).

3 Exemplos de Código

Nesta seção será apresentado alguns algoritmos que colocam em prática os modelos apresentados anteriormente.

3.1 Exemplo Perceptron

Exemplo de classificação de dígitos manuscritos, retirado da documentação do Scikit-Learn [14] e [15].

Código : Execute o código do exemplo pelo [Colab](#), ou acesse

https://colab.research.google.com/drive/1mewI_znw9z7rRYH5T0Zgs1b28-DiAfic?usp=sharing.

```

1
2 from sklearn.datasets import load_digits
3 digits = load_digits()
4 print(digits.data.shape)
5
6 import matplotlib.pyplot as plt
7 plt.gray()
8 plt.matshow(digits.images[0])
9 plt.show()
```

Código 1: Exemplo Perceptron - Digits - Fonte [14]. [15]

```

1
2 from sklearn.datasets import load_digits
3 from sklearn.linear_model import
4 X, y = load_digits(return_X_y=True)
5 clf = Perceptron(tol=1e-3, random_state=0)
6 clf.fit(X, y)
7 clf.score(X, y)
```

Código 2: Exemplo Perceptron - Classificação Digits - Fonte [14]. [15]

3.2 Exemplo Multilayer-Perceptron

Este exemplo foi retirado da documentação do Scikit-Learn, consultar [16]. Ele mostra como plotar alguns dos pesos da primeira camada em um MLPClassifier treinado no conjunto de dados MNIST.

Os dados de entrada consistem em dígitos manuscritos de 28x28 pixels. Para tornar o exemplo mais rápido, usaram poucos neurônios na camada oculta e baixo número de iteração.

Código : Execute o código do exemplo pelo [Colab](#), ou acesse https://colab.research.google.com/drive/1DPk07JDkpAkuKmwjk_8xcbtkQUi0-yWv?usp=sharing.

```

1 import warnings
2
3 import matplotlib.pyplot as plt
4 from sklearn.datasets import fetch_openml
5 from sklearn.exceptions import ConvergenceWarning
6 from sklearn.neural_network import MLPClassifier
7
8 # Load data from https://www.openml.org/d/554
9 X, y = fetch_openml("mnist_784", version=1, return_X_y=True)
10 X = X / 255.0
11
12 # rescale the data, use the traditional train/test split
13 X_train, X_test = X[:60000], X[60000:]
14 y_train, y_test = y[:60000], y[60000:]
15
16 mlp = MLPClassifier(
17     hidden_layer_sizes=(50,),
18     max_iter=10,
19     alpha=1e-4,
20     solver="sgd",
21     verbose=10,
22     random_state=1,
23     learning_rate_init=0.1,
24 )
25
26 # this example won't converge because of CI's time constraints, so we
27 # catch the
28 # warning and are ignore it here
29 with warnings.catch_warnings():
30     warnings.filterwarnings("ignore", category=ConvergenceWarning,
31                             module="sklearn")
32     mlp.fit(X_train, y_train)
33
34 print("Training set score: %f" % mlp.score(X_train, y_train))
35 print("Test set score: %f" % mlp.score(X_test, y_test))
36
37 fig, axes = plt.subplots(4, 4)
38 # use global min / max to ensure all weights are shown on the same
39 # scale
40 vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
41 for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
42     ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=0.5 * vmin,
43                vmax=0.5 * vmax)

```

Fit-F.24.

```

40     ax.set_xticks(())
41     ax.set_yticks(())
42
43 plt.show()

```

Código 3: Exemplo MLP - MNIST- Fonte [16]

3.3 Exemplo Rede Neural Convolucional

Este é um simples exemplo, proposto por [17], que demonstra em funcionamento um classificador CNN usando o *dataset* de imagens CIFAR. Usando o framework TensorFlow que utiliza Keras Sequential API, que reduz o treinamento em poucas linhas de código.

Código : Execute o código do exemplo pelo [Colab](#), ou acesse

<https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/cnn.ipynb>.

```

1 import tensorflow as tf
2
3 from tensorflow.keras import datasets, layers, models
4 import matplotlib.pyplot as plt
5
6 (train_images, train_labels), (test_images, test_labels) = datasets.
7     cifar10.load_data()
8
9 # Normalize pixel values to be between 0 and 1
10 train_images, test_images = train_images / 255.0, test_images / 255.0
11
12 class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
13                 'dog', 'frog', 'horse', 'ship', 'truck']
14
15 plt.figure(figsize=(10,10))
16 for i in range(25):
17     plt.subplot(5,5,i+1)
18     plt.xticks([])
19     plt.yticks([])
20     plt.grid(False)
21     plt.imshow(train_images[i])
22     # The CIFAR labels happen to be arrays,
23     # which is why you need the extra index
24     plt.xlabel(class_names[train_labels[i][0]])
25
26 model = models.Sequential()
27 model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,
28                         32, 3)))
29 model.add(layers.MaxPooling2D((2, 2)))

```

Fit-F24. [model.fit\(x_train, y_train, epochs=10\)](#)



```

29 model.add(layers.Conv2D(64, (3, 3), activation='relu'))
30 model.add(layers.MaxPooling2D((2, 2)))
31 model.add(layers.Conv2D(64, (3, 3), activation='relu'))
32
33 model.summary()
34
35 model.add(layers.Flatten())
36 model.add(layers.Dense(64, activation='relu'))
37 model.add(layers.Dense(10))
38
39 model.summary()
40
41 model.compile(optimizer='adam',
                 loss=tf.keras.losses.SparseCategoricalCrossentropy(
                     from_logits=True),
                 metrics=['accuracy'])
42
43
44 history = model.fit(train_images, train_labels, epochs=10,
                      validation_data=(test_images, test_labels))
45
46
47 plt.plot(history.history['accuracy'], label='accuracy')
48 plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
49 plt.xlabel('Epoch')
50 plt.ylabel('Accuracy')
51 plt.ylim([0.5, 1])
52 plt.legend(loc='lower right')
53 test_loss, test_acc = model.evaluate(test_images, test_labels, verbose
54                                     =2)
55
56 print(test_acc)

```

Código 4: Exemplo Classificador CNN - Fonte [17]

4 Exercícios Avaliativo

Nesta seção é apresentado o exercício avaliativo, explore ao máximo tudo que foi visto durante o curso. A avaliação será disponibilizada em um notebook Colab (link abaixo). O aluno deve gerar uma cópia do notebook fornecido e se atentar na configuração de visualização pública, para que o professor tenha acesso. Informe o link do notebook no campo disponível na plataforma.

Aluno: Acesse a avaliação pelo [Colab](#), ou pelo link

<https://colab.research.google.com/drive/1VFqfkXwSGpwrQBRKSyofhS2k5Bup97I8?usp=sharing>.

· Avaliação - Redes Neurais

Fit - Instituto de Tecnologia

Prof. Adson Nogueira Alves

```
[ ] # TODO: Coloque seu nome e e-mail
print(f'Nome: ..SUBSTITUA PELO SEU NOME..')
print(f'E-mail: ..SUBSTITUA PELO SEU E-MAIL..')

Nome: ..SUBSTITUA PELO SEU NOME..
E-mail: ..SUBSTITUA PELO SEU E-MAIL..
```

Objetivo

Explorar técnicas de classificação, com intuito de atingir o melhor modelo possível para o problema, evitando overfitting. De forma particular construir um sistema de reconhecimento e classificação de imagens usando o [PathMNIST dataset](#).

· Dataset

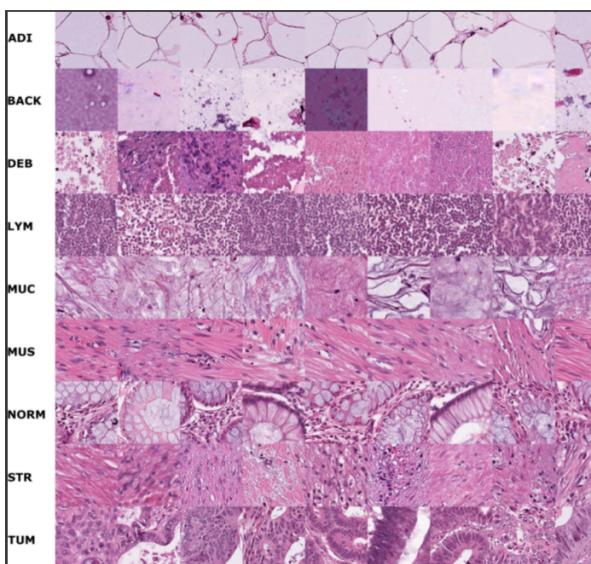
PathMNIST é um conjunto de dados de imagens de artigos do MedMNIST, consistindo em um conjunto de treinamento de 89.996 exemplos, um conjunto de validação de 10.004 exemplos e um conjunto de teste de 7.180 exemplos.

Cada exemplo de patologia do cólon é uma imagem RGB 28x28, associada a um rótulo de 9 classes.

Cada exemplo é atribuído a um dos seguintes rótulos:

Descrição da etiqueta

- ADI adipose tissue
- BACK background
- DEB debris
- LYM lymphocytes
- MUC mucus
- MUS smooth muscle
- NORM normal colon mucosa
- STR cancer-associated stroma
- TUM colorectal adenocarcinoma epithelium.



▼ Entenda o código e execute

```
[ ] # Download dataset
! wget https://zenodo.org/record/5208230/files/pathmnist.npz

--2021-11-10 10:24:57-- https://zenodo.org/record/5208230/files/pathmnist.npz
Resolving zenodo.org (zenodo.org)... 137.138.76.77
Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 205615438 (196M) [application/octet-stream]
Saving to: 'pathmnist.npz'

pathmnist.npz      100%[=====] 196.09M  19.3MB/s    in 13s

2021-11-10 10:25:12 (14.6 MB/s) - 'pathmnist.npz' saved [205615438/205615438]
```

```
[ ] import numpy as np

[ ] # preparing data to use with sklearn
pathmnist = np.load('pathmnist.npz')

x_train, y_train = pathmnist['train_images'], pathmnist['train_labels']
x_val, y_val = pathmnist['val_images'], pathmnist['val_labels']
x_test, y_test = pathmnist['test_images'], pathmnist['test_labels']

x_train = x_train.reshape(x_train.shape[0], -1)
x_val = x_val.reshape(x_val.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)

y_train = y_train.squeeze()
y_val = y_val.squeeze()
y_test = y_test.squeeze()
```

▼ Questões

1. (2 pontos) Execute um modelo de classificação (Regressão logistica) usando Scikit-Learn.

Use Logistic Regression

[]

2. (1 ponto) Faço o treinamento do modelo utilizando apenas 1000 imagens

[]

3. (1 pontos) Mostre as:

- Classes de saída da rede. (Utilize os atributos da função)
- Número de iterações antes da tolerância ser atingida
- Pesos da imagem na classe [3]
- O bias para as 9 classes
- A penalty de regularização para evitar o overfitting

[]

4. (1 ponto) Mostre o vetor de predição para os valores de x_val , score de treinamento e o score de validação

[]

5. (1 pontos) Gere a matriz de confusão, dos dados de validação

[]

6. (1 ponto) Execute uma rede neural, usando uma ou duas camadas ocultas. Você deve conseguir escolher peso e bias de inicialização, função de ativação, número de neurônios ocultos, função de perda ... mantenha simples. (More o score de treinamento e validação)

[]

7. (3 pontos) Aplique técnicas de preprocessamento de imagem para melhorar o desempenho da rede neural (RGB para Escala Cinza, Padronização e Normalização) - Use MLPClassifier

Escala cinza

[]

Normalização

[]

Padronização

[]

8. (1 ponto) Utilize uma nova arquitetura para a rede (taxa de aprendizado constante, valor da taxa de aprendizado = $1e^{-4}$, maximo de iterações = 1000, otimizador = adam, função de ativação = 'relu', camada oculta unica com 150 neurônios

[]

9.(1 point) Aplique um peso de regularização para evitar o overfitting (alpha) e melhorar o desempenho da rede neural. Utilize o seu melhor resultado de preprocessamento.

[]

10. (3 pontos) Quais suas conclusões ? Fale sobre as arquiteturas, preprocessamento ... (Campo livre)

##

Conclusão

Parabéns por ter chegado ao final de mais um curso! Neste curso exploramos o tópico de Redes Neurais e abordamos algumas arquiteturas de redes bastante conhecidas e utilizadas, como o **Perceptron**, **Multilayer Perceptron (MLP)** e **Convolutional Neural Network (CNN)**. Você está apto a criar os seus primeiros modelos de Redes Neurais e gerar aplicações reais dentro das áreas de: **Robótica**, **Data Science**, **Big Data**, **Estratégias de Negócios**, entre outros.

O caminho de **Aprendizado de Máquina** é longo, você deu os primeiros passos, então explore nossas trilhas, esperamos por vocês nos próximos cursos.

Referências

- [1] MachineLearningMastery. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>, 2017. Acessado: 03.11.2021.
- [2] Medium. <https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10383a2a2>, 2018. Acessado: 03.11.2021.
- [3] TowardsDataScience. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>, note = Acessado: 03.11.2021, 2018.
- [4] Chapter 1. Introduction to Artificial Neural Networks. <https://www.oreilly.com/library/view/neural-networks-and/9781492037354/ch01.html>, 2019. Acessado: 03.11.2021.
- [5] The Neural Network Zoo. <https://www.asimovinstitute.org/neural-network-zoo>, 2019. Acessado: 03.11.2021.
- [6] A System Based on Artificial Neural Networks for Automatic Classification of Hydro-generator Stator Windings Partial Discharges. https://www.researchgate.net/figure/Mathematical-model-of-artificial-neuron_fig1_320270458, 2021. Acessado: 03.11.2021.
- [7] Aishelf. <http://aishelf.org/sgd-learning-rate/>, 2021. Acessado: 03.11.2021.
- [8] Breve historia de Deep Learning. <https://www.deeplearningbook.com.br/uma-breve-historia-das-redes-neurais-artificiais/>, 2021. Acessado: 03.11.2021.
- [9] Educative. <https://www.educative.io/edpresso/a-deep-dive-into-linear-regression-3-way-implementation>, 2021. Acessado: 03.11.2021.
- [10] Neurônio. <https://mundoeducacao.uol.com.br/biologia/neuronios.htm>, 2021. Acessado: 03.11.2021.
- [11] Oracle. <https://developer.oracle.com/databases/neural-network-machine-learning.html>, 2021. Acessado: 03.11.2021.
- [12] Sandra Avila - Machine Lerning. <https://www.ic.unicamp.br/~sandra/>, 2021. Acessado: 03.11.2021.
- [13] Scikit-learn. https://scikit-learn.org/stable/modules/neural_networks_supervised.html, 2021. Acessado: 03.11.2021.
- [14] Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html, note = Acessado: 03.11.2021, 2021.
- [15] Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits, note = Acessado: 03.11.2021, 2021.

- [16] Scikit-Learn. https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py, note = Acessado: 03.11.2021, 2021.
- [17] TensorFlow. <https://www.tensorflow.org/tutorials/images/cnn/>, note = Acessado: 03.11.2021, 2021.

Controle de revisão do documento

Revisão <i>Review</i>	Descrição <i>Description</i>	Razão <i>Reason</i>	Autor <i>Author</i>	Data <i>Date</i>
A	-	Revisão inicial	Larissa Alves	29/11/21
B	Alteração de logotipia do rodapé do documento	Revisão do modelo do formulário FITF.24.1.01-04Rev. B	Adson Alves	27/07/22



MCTI
FUTURO

FUTURO DO TRABALHO, TRABALHO DO FUTURO

Bom curso



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÕES

