CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

FACULDADE DE TECNOLOGIA DE BOTUCATU

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

SERGIO DI FIORE

Desenvolvimento de um Software de Inteligência Artificial para Reconhecimento de ALGARISMOS ARÁBICOS MANUSCRITOS

Botucatu - SP

Dezembro – 2020

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

FACULDADE DE TECNOLOGIA DE BOTUCATU

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

SERGIO DI FIORE

Desenvolvimento de um Software de Inteligência Artificial para Reconhecimento de ALGARISMOS ARÁBICOS MANUSCRITOS

Orientador: Prof. Dr. Gustavo Kimura Montanha

Coorientadora: Profa. Dra. Mônica Regina Gaiotto

Relatório de Iniciação Científica apresentado à FATEC - Faculdade de Tecnologia de Botucatu, como exigência para cumprimento do Trabalho de Conclusão de Curso no Curso Superior de Análise e Desenvolvimento de Sistemas.

Botucatu - SP

Dezembro – 2020

**RESUMO**

O projeto trata do desenvolvimento de um software que visa o reconhecimento de caracteres de algarismos arábicos manuscritos visando a automação de processos de leitura de números manuscritos para diversas finalidades, através de técnicas de Inteligência Artificial, Rede Neural e *Deep Learning*.

**Palavras-chave:** Deep Learning, Inteligência Artificial, Rede Neural,Software.

LISTA DE FIGURAS

[Figura 1. Dois neurônios transmitindo impulso nervoso por meio de sinapse 8](#_Toc65254986)

[Figura 2. Neurônio artificial 8](#_Toc65254987)

[Figura 3. Número 504.192 manuscrito 10](#_Toc65254988)

[Figura 4. Múltiplos diferentes exemplos manuscritos tornam-se exemplares para treinamento de uma rede neural 11](#_Toc65254989)

[Figura 5. Perceptron 12](#_Toc65254990)

[Figura 6. Percepron com mais de uma camada 13](#_Toc65254991)

[Figura 7. O efeito de pequenas alterações em pesos ou bias 14](#_Toc65254992)

[Figura 8. O Neurônio sigmóide é muiti próximo ao perceptron 15](#_Toc65254993)

[Figura 9. Gráfico da função sigmoide 15](#_Toc65254994)

[Figura 10. Função degrau ou de Heaviside 16](#_Toc65254995)

[Figura 11. Uma rede neural genérica 17](#_Toc65254996)

[Figura 12. Rede neural com mais de uma camada intermediária 18](#_Toc65254997)

[Figura 13. Exemplo de problema a ser reconhecido 19](#_Toc65254998)

[Figura 14. O problema dividido em subproblemas: os dígitos separados 19](#_Toc65254999)

[Figura 15. O algarismo 5 individualizado 20](#_Toc65255000)

[Figura 16. Rede neural a ser empregada para reconhecimento dos dígitos individuais 20](#_Toc65255001)

LISTA DE EQUAÇÕES

[Equação 1. Cálculo binário de saída de um Perceptron 9](#_Toc65255018)

[Equação 2. Cálculo do valor de saída de um Perceptron 12](#_Toc65255019)

[Equação 3 atualizada 13](#_Toc65255020)

[Equação 4. Cálculo da saída de um neurônio sigmoide 15](#_Toc65255021)

[Equação 5. Cálculo da saída sigmoide considerando pesos e bias 15](#_Toc65255022)

[Equação 6. Cálculo aproximado de ∆saída 17](#_Toc65255023)

Sumário

[1. INTRODUÇÃO 7](#_Toc65254923)

[2. REVISÃO DE LITERATURA 7](#_Toc65254924)

[2.1 Inteligência Artificial, Redes Neurais e *Deep Learning* 7](#_Toc65254925)

[2.2 Usando redes neurais para reconhecer algarismos manuscritos 10](#_Toc65254926)

[2.3 Perceptron 11](#_Toc65254927)

[2.4 Neurônios Sigmoides 13](#_Toc65254928)

[2.5 Arquitetura das redes neurais 17](#_Toc65254929)

[2.6 Uma rede simples para classificar algarismos manuscritos 19](#_Toc65254930)

[3. REFERÊNCIAS 22](#_Toc65254931)

# INTRODUÇÃO

A Inteligência Artificial em suas várias escolas tem sido cada vez mais presente na solução de problemas informáticos que necessitam tratar grandes quantidades de dados e a partir deles, prever alguma espécie de resultados. Redes neurais artificiais nada mais são do que a simulação das neurais biológicas por meio de componentes eletrônicos, algoritmos e software e o *Deep Learning* uma extensão das Redes Neurais mais tradicionais pelo aprofundamento de camadas intermediárias e algoritmos mais adequados de aprendizado. Através da técnica de Aprendizado de Máquina que se vale desses recursos supracitados, pretende-se desenvolver um sistema capaz de identificar algarismos arábicos manuscritos, uma aplicação que pode ser empregada para diversos fins práticos de entrada de dados a partir de documentos manuais.

# REVISÃO DE LITERATURA

## Inteligência Artificial, Redes Neurais e *Deep Learning*

Pode-se dizer que a Inteligência Artificial (IA) começou a ser estudada sistematicamente a primeira vez por Hebert Simon e Allen Newell ainda no início da década de 1950 na Universidade de Carnegie Mellon. Em seus primeiros estágios, a IA resolveu problemas que são intelectualmente difíceis para os seres humanos, mas simples para a lógica matemática eletrônica. O desafio se provou na resolução de problemas que resolvemos intuitivamente de uma forma simples, mas que humanos tem dificuldades em descrevê-los.

Inteligência artificial nada mais é do que estatística levada ao seu extremo, onde modelos matemáticos são calculados por estruturas computacionais que se inspiraram no cérebro humano. Isto é, “neurônios” que realizam um determinado tipo de operação, levando o resultado de um para outro, em uma organização que emula a transferência de informação de uma real célula nervosa para outra, no sistema animal (Figuras 1 e 2).

Essa estrutura de “células” nervosas artificiais e seus algoritmos permitem que padrões existentes em uma série, normalmente bem grande de dados, possam ser aprendidos para seu posterior uso, decodificando padrões. Por exemplo, mostrando diversas imagens de gatos e posteriormente verificando se uma determinada imagem submetida, sem que esse jamais a tenha analisado, se possui ou não uma imagem de um gato. É a chamada Aprendizagem de Máquina (*Machine Learning*). Com ela, pode-se encontrar padrões em quantidade enorme de dados. Esse modelo, que ainda funciona muito bem, mas que mesmo desde esse período inicial, deixava muitas questões sem solução aparente.

Figura 1. Dois neurônios transmitindo impulso nervoso por meio de sinapse

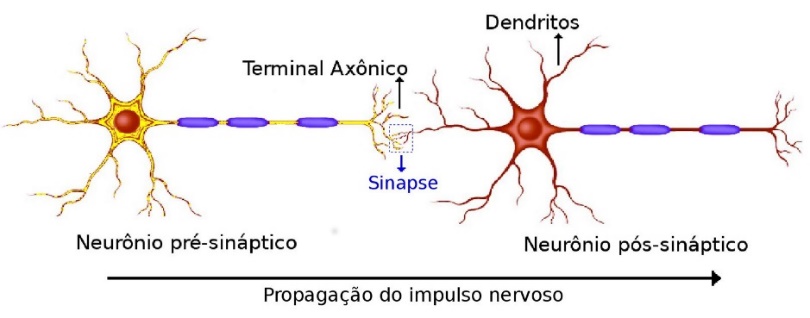
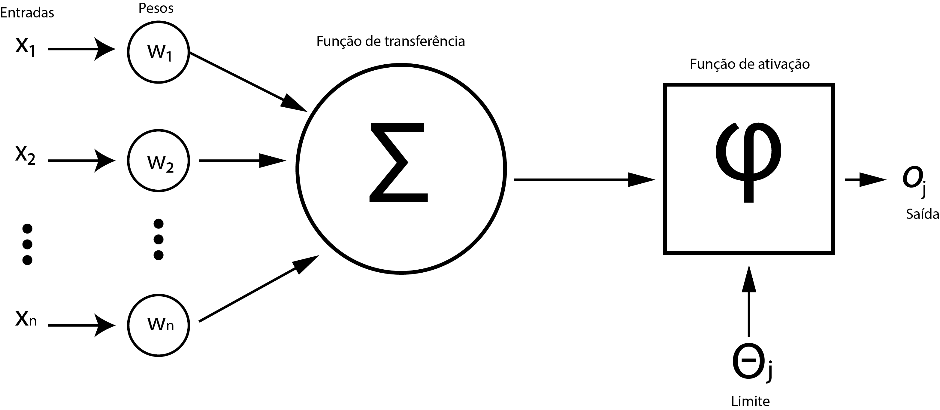


Figura 2. Neurônio artificial



É conveniente neste ponto ressaltar a importância desse aprendizado autônomo. Um sistema para ser entendido como inteligência artificial, deve ser capaz de aprender por si, não dependendo de instruções de programadores.

Em 1969, Marvin Minsky e Saymour Papert publicaram um livro pelo MIT provando matematicamente que esse tipo de redes, chamadas Perceptron (Fórmula 1), poderiam somente executar funções básicas (SOMMERS, 2017). Elas, então, possuíam somente duas camadas de neurônios, uma camada de entrada de dados e uma camada para a saída, e embora, na teoria mais camadas pudessem resolver o problema, ninguém possuía conhecimento suficiente de como treiná-las.

Equação 1. Cálculo binário de saída de um Perceptron

Somente em 1986, Geoffrey Hinton, hoje conhecido como o pai do *Deep Learning*, trouxe novos avanços (HAO, 2018). Ele basicamente, conseguiu desenvolver o treinamento de mais camadas, uma técnica que, iria além do tradicional aprendizado de máquina, permitindo encontrar pequenos padrões, que anteriormente não conseguiriam ser detectados. Em 1990 essa técnica foi reforçada pela assim chamada Máquina de Suporte Vetorial (*Support Vector Machine*) onde estruturas espaciais e funções permitem as transições que simplificam o processamento (WILSON, 2020).

Tal técnica, porém, esbarrava na limitação da capacidade computacional dos sistemas na época e o campo toda da IA ficou por muito tempo parado, até em torno do ano 2000, quando a capacidade de processamento permitiu a retomada das pesquisas (SOMMERS, 2017).

Nesse mesmo período começou-se a empregar a Distribuição Circulares de Autovalores associada a um antigo método matemático conhecido como a Teoria da Matriz Aleatória (*Random Matrix Theory*) minimizando o obstáculo trazidos pelas complexas matrizes que o aprendizado de máquina precisava lidar (WILSON, 2020).

Redes Neurais são um dos mais belos paradigmas de programação jamais inventados. Na abordagem convencional de programação, o computador recebe informações detalhadas do que fazer, dividindo grandes problemas em diversos menores, definem-se precisamente as tarefas que o computador pode executar de maneira simples. Em contraste, em uma Rede Neural nenhuma informação é dada ao computador de como resolver um problema. Ao contrário, ele aprende a partir da observação de dados, identificando a sua própria solução ao problema em questão (Nielsen M. 2015)

Segundo Kelleher J. D. (2019),” *Deep Learning* é um subcampo da Inteligência Artificial que foca em criar grandes modelos de redes neurais que são capazes de fazer previsões precisas a partir de dados. *Deep Learning* é particularmente adequada aos contextos que possuem grandes quantidades de amostragem de dados disponíveis.

## Usando redes neurais para reconhecer algarismos manuscritos

Qualquer ser humano reconhece com facilidade os dígitos:

Figura 3. Número 504.192 manuscrito



A figura 3 exibe o número 504.192 manuscrito por uma pessoa e subsequentemente digitalizado em uma resolução não muito alta como pode-se perceber. O cérebro humano, possui em cada um de seus hemisférios o córtex visual primário, também conhecido como V1 contendo 140 milhões de neurônios, com dezenas de bilhões de conexões entre eles. Ainda assim, a visão não depende somente do V1, mas de toda uma série de córtices: V2, V3, V4 e V5 cada um fazendo progressivamente mais do que o anterior. Milhões de anos de evolução garantiram que o cérebro humano se tornasse um supercomputador para o qual a identificação de dígitos, como os representados, extremamente eficiente.

A dificuldade do reconhecimento de padrões visuais torna-se clara quando tenta-se escrever um programa convencional de computador para fazê-lo. Como reconhecer um “9”? Uma bolinha em cima e uma perna curvada. Expressar isso em termos de algoritmos é extremamente complexo, e ainda, essa representação pode ter uma infinidade de variáveis.

Redes neurais abordam esse mesmo problema de uma forma totalmente distinta. A ideia é tomar uma grande quantidade de algarismos manuscritos conhecidos como exemplares de treino (Figura 4) e desenvolver um sistema que pode aprender a partir destes. Ou seja, a rede neural usa os exemplos para ela própria inferir regras para reconhecer os algarismos manuscritos.

Figura 4. Múltiplos diferentes exemplos manuscritos tornam-se exemplares para treinamento de uma rede neural



Além disso, o incremento do número de exemplos propícia um melhor aprendizado à rede, melhorando o seu desempenho no reconhecimento. Assim, enquanto os 100 dígitos acima refletem um certo potencial de acerto nas inferências quando o sistema analisa dados desconhecidos, usando-se milhares, talvez milhões de exemplos melhoram imensamente essa capacidade.

O primeiro objetivo deste presente projeto é o de atingir uma meta de acerto de 96% sem a intervenção humana. Dentro deste primeiro objetivo, serão desenvolvidos os conhecimentos de dois importantes tipos de neurônios artificiais: o Perceptron e o Neurônio Sigmoide, além de um algoritmo básico para as redes neurais conhecido como Gradiente Descendente Estocástico.

## Perceptron

Desenvolvido entre os anos 1950 e 1960 por Frank Rosenblatt, com base em trabalhos anteriores de Watten McCulloch e Walter Pitts, Rosenblatt apresentou um algoritmo que nos trina anos subsequentes foram alvos de inúmeros trabalhos e estudos acadêmicos mostrando sua eficácia, mas sem que ele mesmo tenha de alguma forma justificado o resultado ao qual chegou.

O Perceptron toma diversas entradas binárias para produzir uma saída também binária como a figura 5 mostra:

Figura 5. Perceptron



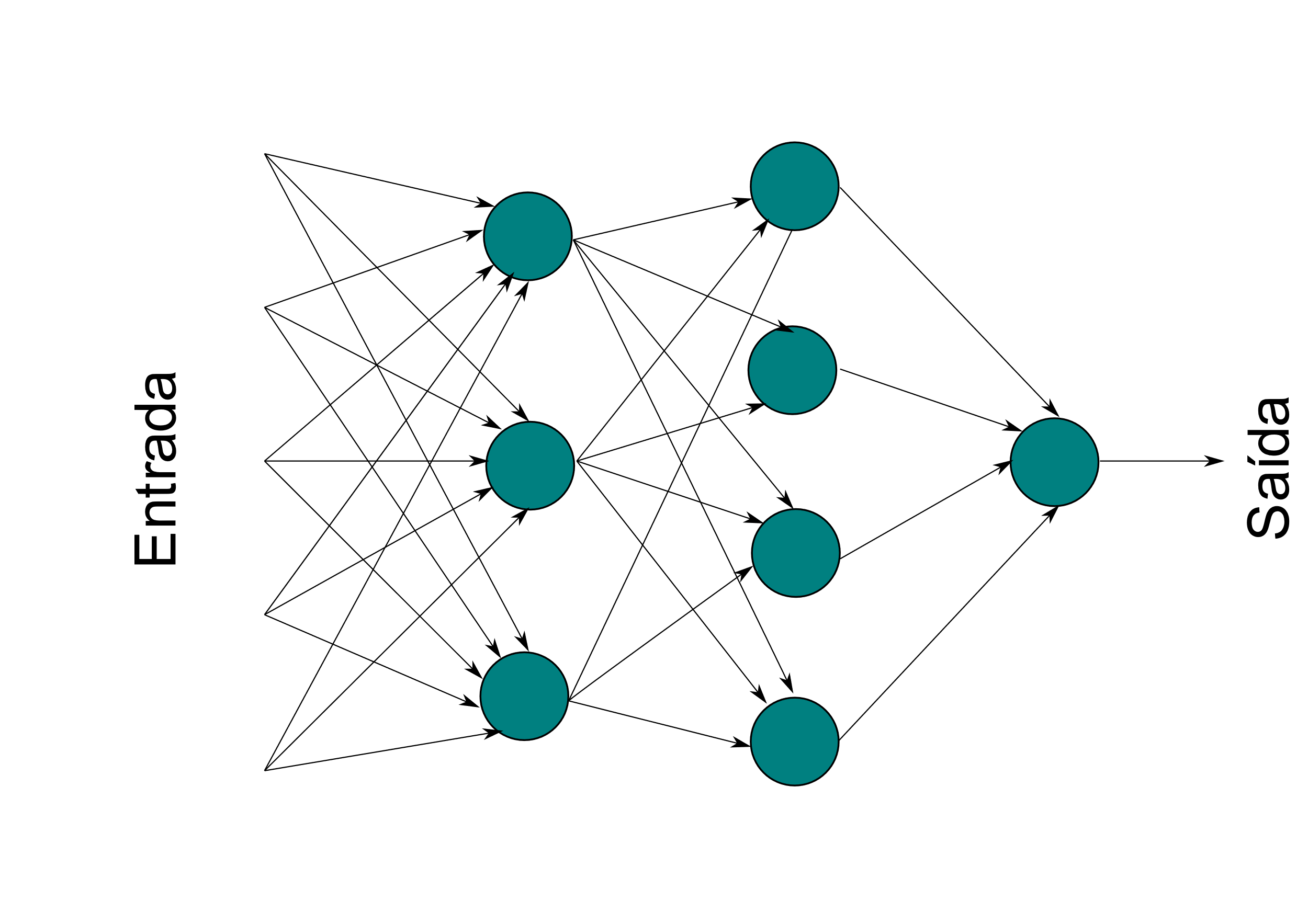
No exemplo, n entradas binárias são processadas para produzir uma saída também binária. Rosenblatt propôs pesos, *w*1, *w* 2, ..., *w* n, números reais indicando a importância de cada entrada. A saída é calculada pela soma do valor de entrada com o seu peso (*w*. *x*), para todas as entradas () atribuindo-se 0 se for menor ou igual que um determinado valor limite e 1 se for maior, como mostrado pela equação 2.

Equação 2. Cálculo do valor de saída de um Perceptron

Uma forma conveniente de entender o Perceptron é que é um dispositivo capaz de tomar decisões pesando evidências.

A figura 6 traz um perceptron com uma camada adicional às de entrada e de saída. A primeria camada pesa as entradas e toma alguma decisão. Neste exemplo, a saída dessas entradas são alimentadas numa nova camada ( a segunda exemplo) que com os seus próprios pesos toma alguma outra decisão e encaminha para mais uma camada (no exemplo, a terceira) que repete o processo, para só então determinar um resultado final como saída. Dessa forma um perceptron de muitas camadas pode decisões sofisticadas.

Figura 6. Percepron com mais de uma camada



Pode-se simplificar a descrição dos perceptrons. Inicialmente alterando , mudando para o produto escalar onde e são vetores respectivamente dos pesos e entradas. A segunda mudança é mudar o limite para o outro lado da inequalidade, substituindo-o pelo que é conhecido como tendência (o termo é mais usado em inglês: *bias*). Assim, podemos reescrever a equação 2 pela equação 3:

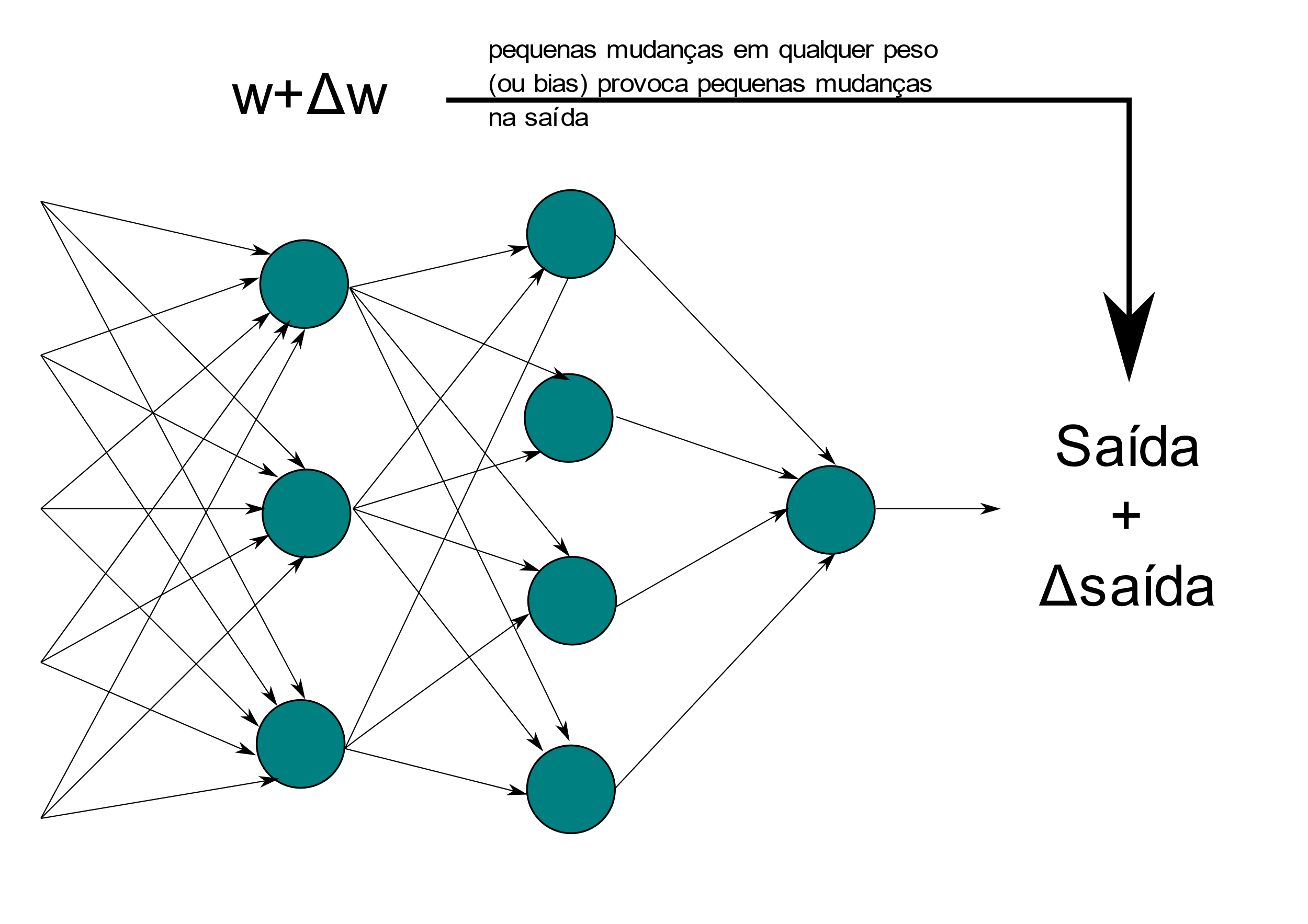
Equação 3 atualizada

Pode-se pensar no *bias* como quão facilmente um perceptron pode atingir o valor 1. Para *bias* com valores altos, a chance de se chegar ao valor 1 é alta, já se o *bias* for um grande valor negativo, isto será difícil. Deste ponto em diante do trabalho nós só usaremos o *bias* e não mais o limite.

## Neurônios Sigmoides

Suponha-se uma rede de perceptrons que é alimentada com o resultado bruto da digitalização de algarismos escritos à mão. É desejável que a rede aprenda pesos e *bias* de tal forma que na saída tenha-se os dígitos classificads de forma correta. Para entender como o processo de aprendizagem pode funcionar, imagina-se (Figura 7) conduzir pequenas mudanças em alguns dos pesos (ou dos *bias*) da rede. Deseja-se que às pequenas mudanças referidas somente pequenas mudanças ocorram correspondentemente na saída. Como será visto na sequência, é justamente essa propriedade que torna possível o processo de aprendizagem.

Figura 7. O efeito de pequenas alterações em pesos ou bias



Se fosse verdadeiro que pequanas modificações em pesos (ou *bias*) provocam somente pequenas modificaçõas no valor de saída, poderia-se usar essa fato para a rede se comportar mais como se deseja. Por exemplo, se é esperado um resultado com o valor 8 e está sendo obtendido o 9, pequenas alterações nos pesos e *bias* levariam mais próximos ao resultado desejado, e repetindo esse processo a precisão melhoraria.

Mas, não é o que acontece com os perceptrons. É possível que uma pequena alteração em um peso ou *bias* cause a mudança completa de estado de um perceptron, por exemplo, de 0 para 1.

Esse problema pode ser evitado com o uso do neurônio sigmóide (Figura 8). Semelhante ao perceptron mas permite justamente que pequenas mudanças em pesos e *bias* reflitam em pequenas mudanças no valor de saída.

Figura 8. O Neurônio sigmóide é muiti próximo ao perceptron



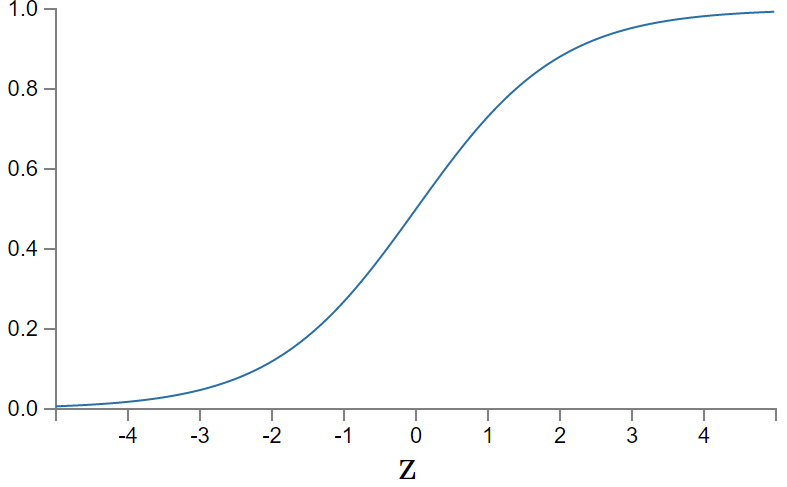
Como o perceptron, o sigmóide também tem pesos para cada entrada, w1, w2, ..., wn. A saída, ao invés de 0 ou 1 esta é calculada pela função sigmoide como ilustrado pela equação 4:

Equação 4. Cálculo da saída de um neurônio sigmoide

De forma mais explicita, agora com pesos e *bias* tem-se:

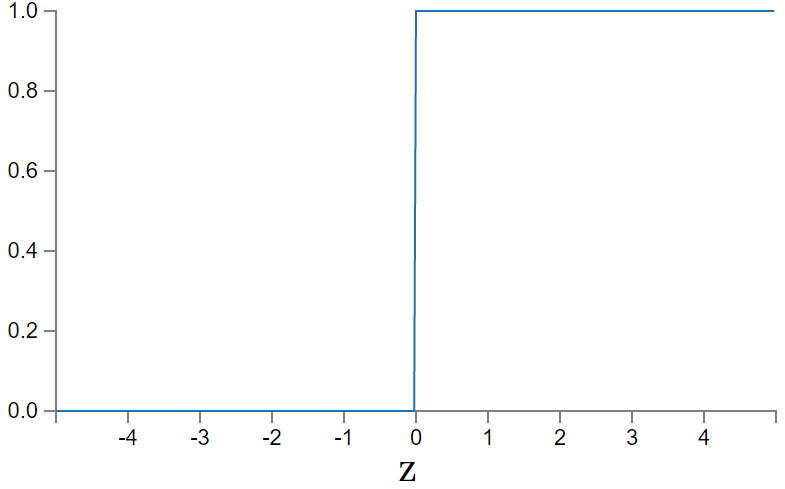
Equação 5. Cálculo da saída sigmoide considerando pesos e bias

Figura 9. Gráfico da função sigmoide



ertghyrPara entender a similaridade entre perceptron e sigmoide, suponha-se sendo um grande número positivo. Então e por isso, . Ou seja, quando é um número grande positivo, a saida da função sigmoide é aproximadamente 1, como ocorreria com um perceptron. Quando é um valor de grande magnitude negativa, então , causando, . Isto é, para valores grandes negativos, o comportamento da função sigmoide também parece a do perceptron. Somente quando possuir valores moderadamente intermediários é que vai haver um desvio considerável do perceptron.

Figura 10. Função degrau ou de Heaviside



Se fosse de fato uma função degrau, o neurônio sigmoide seria um perceptron. Pelo uso da função obtem-se um perceptron “suavizado”. E justamente essa suavização que é o fator crucial. Ela significa que pequenas alterações dos pesos ∆wj e das *bias* ∆b produziram uma pequena alteração na saída ∆saída. Pode-se saber por meio da disciplina do cálculo que ∆saída será aproximadamente:

Equação 6. Cálculo aproximado de ∆saída

onde a soma é entorno de todos os pesos, wj, e denotam as derivadas parciais a saída com relação a wj e b respectivamente. Portanto, ∆saída é uma função linear das mudanças que ocorrem com ∆wj e ∆b.

Como a neurônios sigmoides trabalham com qualquer valor entre 0 e 1, isso é útil, por exemplo, para se considerar o valor de saída como a intensidade média de um pixel da imagem que estamos trabalhando.

## Arquitetura das redes neurais

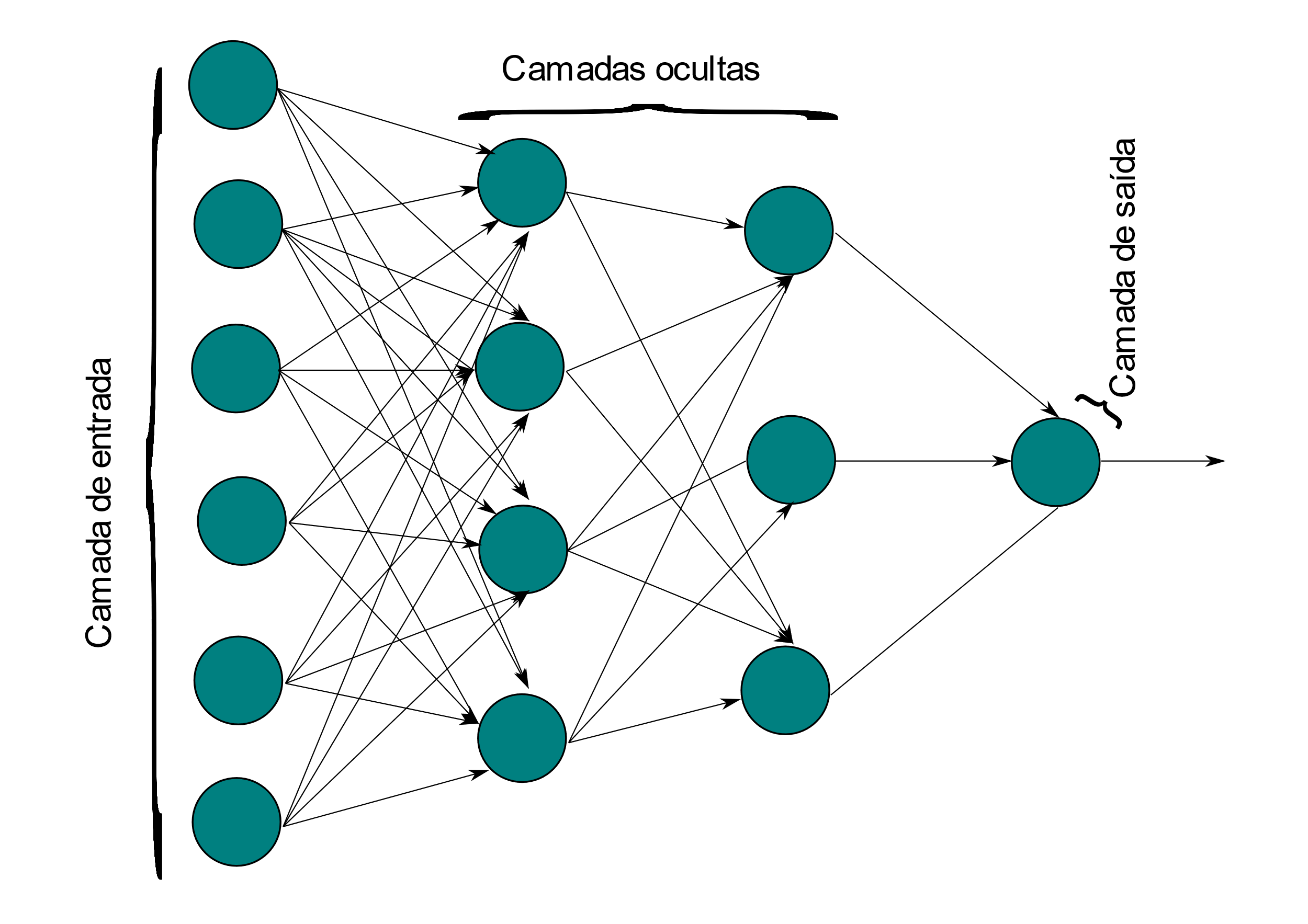
Suponha-se a rede da fixura 11 abaixo:

Figura 11. Uma rede neural genérica



Como anteriormente mencionado, a parte esquerda chamada de camada de entrada contém os denominados neurônios de entrada. A parte direita chamada camada de saída possui os chamados neurônios de saída (note que a figura mostra somente um, mas isso não é necessário). O meio é denominado de camadas escondidas *(hidden layer*) uma vez que não é de entrada nem de saída. A rede da figura 11 possui somente uma camada de saída, mas isso não é necessário, como é o exemplo da figura 12.

Figura 12. Rede neural com mais de uma camada intermediária



Vale ressaltar que, apesar da figura 12 exibir duas camadas ocultas, não existe um número predeterminado e portanto, pelo menos teoricamente, para a sua quantidade também.

Dando, com bom motivo, margem a discussões, essas redes multicamadas são muitas vezes denominadas perceptrons multicamadas (*multilayer perceptrons –* MLPs) apesar de serem constituídas de neurônios sigmoides. Esta terminologia não é usada no presente trabalho.

As camadas de entrada e saída são bem fáceis de entender. Imagine-se que se está tentando identificar se uma imagem é o algarismo “9”. Codificam-se os pixels da imagem da camada de entrada de acordo com a intensidade deste, por exemplo em uma escala de tons de cinza. Se a imagem for uma matriz de 64 por 64 pixels tem-se 4.096 neurônios de entrada com valores que variam entre 0 e 1. A camada de saída contém um único neurônio onde valores inferiores a 0,5 são entendidos com “não é um 9” e valores entre 0,5 e 1 como confirmação de que o algarismo é um “9”.

Já as camadas ocultas têm no seu funcionamento uma verdadeira arte. É extremamente complexo resumi-las em alguma regra simples e muitas heurísticas para tal foram desenvolvidas. Um exemplo considera a quantidade das camadas ocultas versus o tempo necessário para treinar a rede. Algumas dessas heurísticas serão discutidas neste trabalho.

Até o momento foram discutidas redes na qual as informações se propagam em um único sentido, da camada de entrada para a camada de saída. São as chamadas redes *feedforward*, onde não existe nenhuma espécie de laço dos dados. Caso houvesse ter-se-ia a entrada para a função dependendo da saída, o que não faz muito sentido.

No entanto deve-se ressaltar que existem redes neurais na qual laços de retroalimentação ocorrem. O conceito é o de se ter neurônios que se ativam por um período limitado antes de se tornarem quiescente. A sua atividade estimula outros que podem se tornar ativos um pouco depois e assim, ao longo do tempo tem-se uma cascata de atividade dos neurônios e, neste modelo, laços não afetam o funcionamento uma vez que a entrada será dada em algum momento posterior.

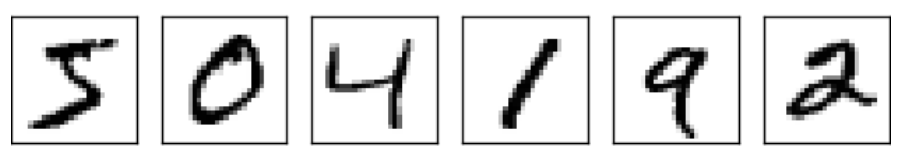
## Uma rede simples para classificar algarismos manuscritos

Inicialmente divide-se o problema em unidades menores, por exemplo, toma-se o número manuscrito, tomando-se por exemplo o úmero 504.192 que pode ser manuscrito como o exemplo da figura 13 e divide-se em nível dos dígitos como exibido pela figura 14, onde separou-se cada um dos dígitos compondo seis unidades distintas.

Figura 13. Exemplo de problema a ser reconhecido



Figura 14. O problema dividido em subproblemas: os dígitos separados



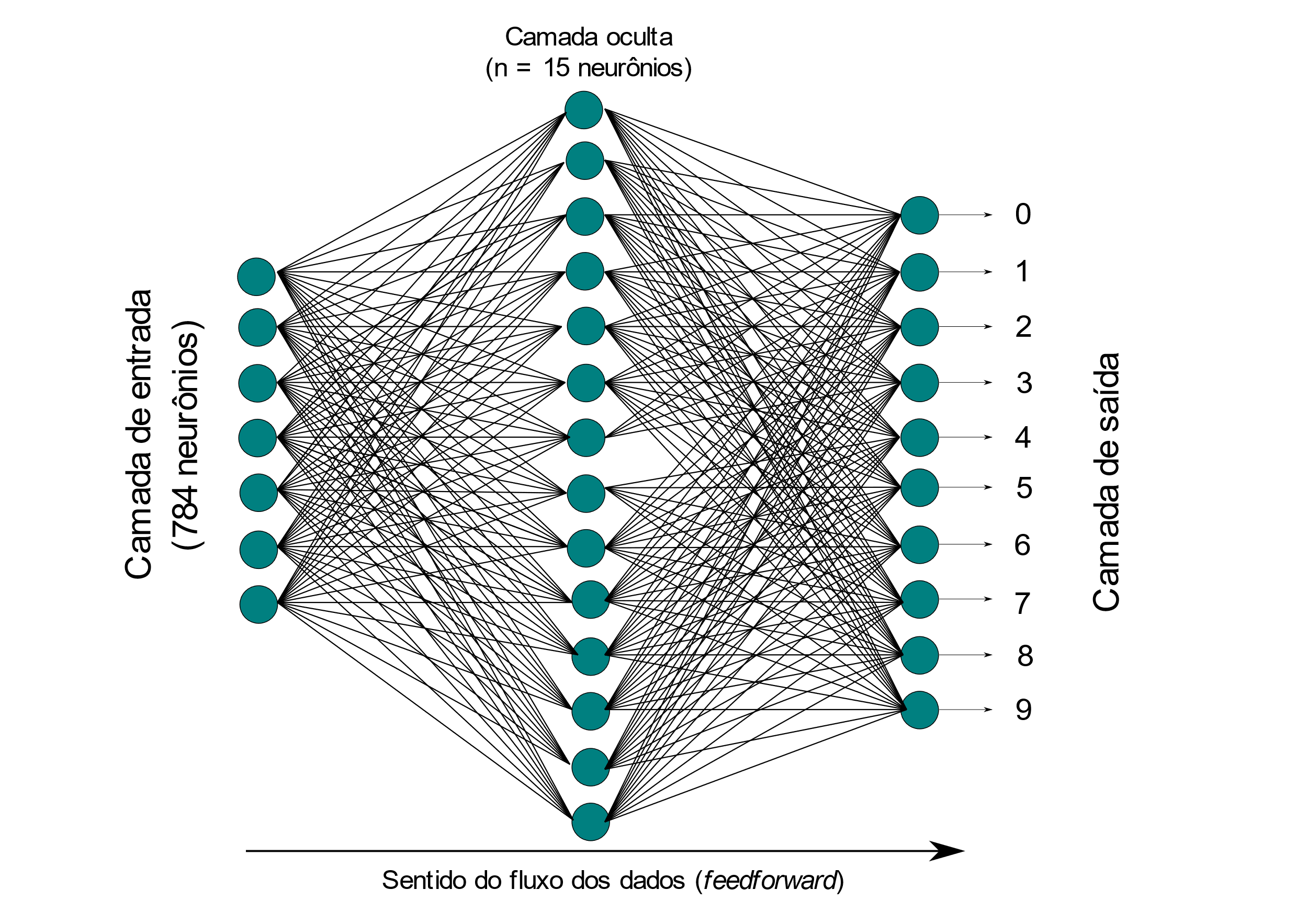
Humanos fazem essa segmentação de forma extremamente simples e natural, mas isso se constitui um verdadeiro desafio em termos computacionais. Uma vez feita essa divisão, o prgrama precisa classificar cada componente (dígito) individualmente. Tome-se por exemo o dígito que fácilmente um humano reconhece como o valor 5, à figura 15.

Figura 15. O algarismo 5 individualizado



Neste momento será dado enfoque em escrever um programa que resolva esse segundo problema, isto é, classificar os dígitos separadamente. Existem muitas abordagens que resolvem esse problema. A intensão é desenvolver uma rede neural que possa reconhecer os dígitos manuscritos individualmente. Será empregada uma rede neural com três camadas, como exibido pela figura 16.

Figura 16. Rede neural a ser empregada para reconhecimento dos dígitos individuais



O dígito a ser trabalhado está contido em uma matriz de 28 x 28 pixels, totalizando 784 pixels. Para cada um destes existe um neurônio na cada de entrada. Por questão de simplicidade, foram omitidos a maioria destes na figura 16. Cada um dos neurônios de entrada tem um valor de escala de cinza onde 0,0 representa branca e o valor 1,0 representa a preta. Valores intermediários representam tons de cinza escalando dos mais claros aos mais escuros.

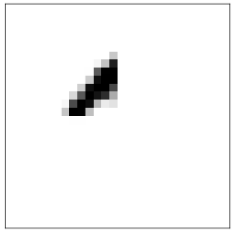
A segunda camada é a camada oculta. O número de neurônios desta é representado por *n* e diversos valores de *n* serão experimentados. O exemplo traz uma camada oculta modesta, com *n* = 14 neurônios.

A camada de saída possui 10 neurônios. Se, por exemplo, o primeiro neurônio possuir um valor de aproximadamente 1, isso irá indicar que a rede acredita que o dígito em questão é o “0”. Mas, se o valor próximo de 1 for o do segundo neurônio, então a rede acredita que o dígito é o “1”. E assim sucessivamente.

Uma questão que pode surgir é porque utilizar 10 neurônios de saída. Um exemplo de alternativa poderia utilizar apenas 4 e estes mostrarem uma representação binária do dígito que a rede acredita estar representado, afinal 24 = 16, o que é mais do que suficiente para representar dígitos de 0 a 9. Afinal, por que 10? A resposta dá-se por motivos empíricos. Várias possibilidades de solução para esse problema já foram testadas e se concluiu que essa forma com 10 neurônios de saída é a que tem o melhor desempenho. Isso acaba levantado a questão: Por que 10 neurônios de saída e não 4? Qual é a heurística por trás?

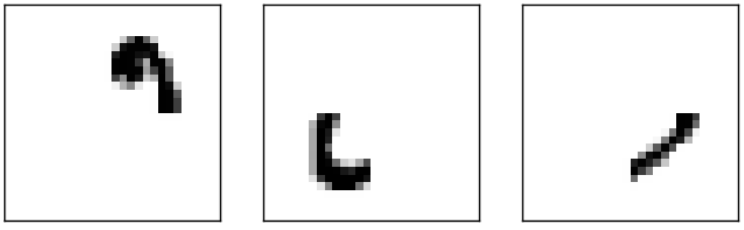
Para a resposta, inicie-se a analisar primeiramente uma camada de saída com 10 neurônios. Considere-se inicialmente ter o foco no primeiro neurônio de saída, o que considera a possibilidade de que seja o dígito “0”. Essa possibilidade é avaliada em função dos valores dos neurônios da camada intermediária e, o que estes estão fazendo? Neste momento, meramente por questão argumentativa, imagine-se que o primeiro neurônio da camada intermediária detecte se a imagem parece ou não a da figura 17:

Figura 17. Suposição de imagem que pode ser detectada pelo primeiro neurônio da camada intermediária



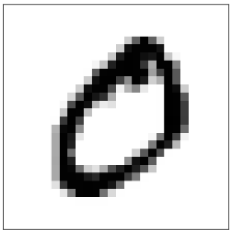
Isso pode ser obtido pelo uso de um peso muito significativo (elevado) aplicado aos pixels da camada de entrada os quais sobreponham-se a essa imagem. De forma similar, novamente meramente por questão argumentativa, suponha-se que o terceiro e o quarto neurônio da camada oculta detectem se a imagem em questão tem elementos comuns com as imagens da figura 18:

Figura 18. Suposição de possíveis imagens que seriam detectadas pelos neurônios 2, 3 e 4 da camada oculta



Colocando as quatro imagens juntas temos o dígito “0” como mostrado pela figura 19:

Figura 19. Sobreposição das imagens da das figuras 17 e 18



Então, se esses quatro neurônios tiverem um valor elevado, pode ser assumido que o dígito sendo analisado é o “0”. Apesar de essa não ser a única maneira possível de se concluir que se trata de fato de um “0”, pela presunção considerada até aqui, é seguro assumir que sim.

Mantendo a suposição que essa explicação seja válida, fica fácil entender por que 10 valores de saída são melhores que 4, caso em que, por exemplo, a primeira saída deveria representar o bit mais significativo e não existe um meio fácil de associar forma com esse valor.

Convém ressaltar que até este ponto tudo é heurística, nada leva a conclusão que uma rede neural de três camadas opera dessa forma.

## Aprendendo o método do gradiente

### A base de dados MINST

Supondo-se saber como desenhar uma rede neural, agora é hora de aprender a reconhecer dígitos. A primeira coisa necessária é ter uma amostra de dados da qual se possa aprender, a chamada *data set*. Neste trabalho será usada a base de dados MNIST *data set,* que contém dezenas de milhares de imagens digitalizadas de algarismos manuscritos.

Os dados da base MINST são divididos em duas partes. A primeira, contém 60.000 imagens com o propósito de serem dados de treinamento. São imagens digitalizadas a partir de manuscritos de 250 pessoas, representadas em tons de cinza em uma matriz de 28 x 28 pixels. A segunda parte possui um conjunto de 10.000 imagens tomadas de 250 pessoas diferentes, para serem utilizadas como dados de teste, novamente em tons de cinza em uma matriz de 28 x 28 pixels. Essa será utilizada para determinar quão bem a rede neural gerada aprendeu a reconhecer os dígitos.

### O treinamento

Será empregada a notação x para identificar as entradas de treinamento. Cada uma delas é formada por um vetor de 784 posições (matriz 28 x 28 = 784) representando um tom em uma escala de cinza para cada pixel da imagem. A saída é identificada por y = y(x), onde y é um vetor de dimensão 10. Por exemplo, se uma imagem de treinamento, x, representa um “6”, então y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)*T* é a saída desejada. Note-se o *T* indicando a operação transposição de vetor, que faz da linha um vetor coluna.

O desejável é que o algoritmo que se está buscando permita definir todos os pesos e *bias* de tal forma que o resultado de saída da rede aproxime y(x) para todos os x na entrada. Isto é quantificado pela função custo:

Equação 7. Função Custo

onde n é o conjunto de todos os pesos na rede, b todos os *biases*, n o número total de entradas de treinamento, a é o vetor de saída da rede quando x é a entrada, e a soma é sobre todos os x de entrada. A notação ||v|| simplesmente indica o comprimento de um vetor. C é chamada de função quadrática do custo, muitas vezes conhecida também pela denominação erro médio quadrático ou simplesmente MSE (*mean squared error*). Verifica-se que esta função não é negativa uma vez que cada termo da soma também não o é. Nota-se que torna-se menor, isto é, , quando y(x) é de forma aproximada igual à saída, a, para todas as entradas de treino x. Depreende-se que o algoritmo é eficiente quando encontra pesos e *biases* de tal forma que . Já, se esse valor for alto, entende-se que y(x) não se aproxima da saída a para muitos dos valores da entrada. O objetivo é então minimizar a função custo, sendo esta uma função de pesos e *biases*. Para fazê-lo, será utilizado o algoritmo conhecido como método do gradiente.

### O algoritmo método do gradiente

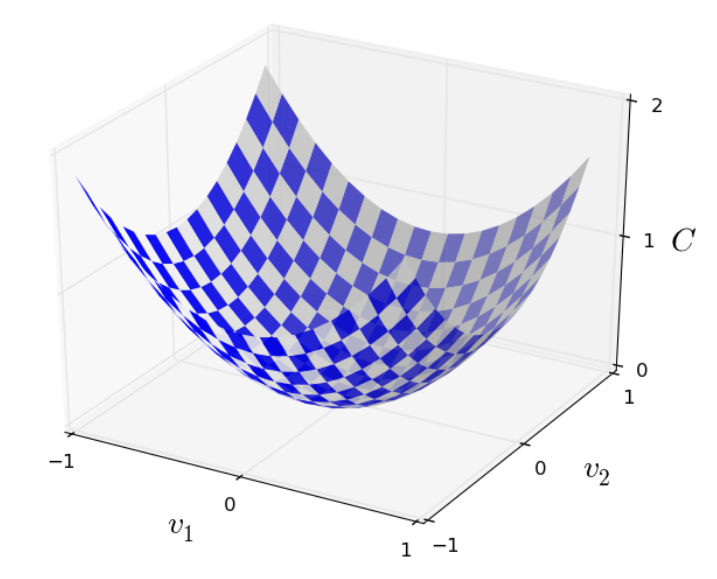
Deve-se considerar é que o número de imagens classificadas de forma correta não é uma função suave dos pesos e *biases*. Pequenas mudanças nestes não provocam nenhuma mudança no número de imagens classificadas de forma correta, o que torna difícil decifrar como mudá-los. Na situação em que se usa a função quadrática do custo, foca fácil perceber como efetuar pequenas modificações nos pesos e *biases* de forma e ter um custo mais bem otimizado. Por isso o foco neste e somente depois se considerará a precisão da classificação.

Em outro momento se revisará a função do custo e algumas modificações serão feitas. Mas, para o momento, ela funciona adequadamente e assim será utilizada.

Resumindo, o objetivo imediato é em treinar uma rede neural é encontrar valores para pesos e *biases* que minimizem a função quadrática do custo . Iniciar-se-á trabalhando com o conceito de uma equação de diversas variáveis e o problema de sua minimização. Será desenvolvida a técnica conhecida como método do gradiente ou método do máximo declive que pode ser empregado em diversos problemas de minimização como este. Só após isto se retornará à função específica que se quer minimizar para a rede neural.

Assim, deseja-se minimizar uma função , que pode ser qualquer função de valor real com múltiplas variáveis v = v1, v2, ..., vn. Para fazê-lo é útil o exercício mental de se imaginar como uma função de somente duas variáveis, v1 e v2 que pode ser facilmente representada graficamente como na figura 20:

Figura 20. Função de duas variáveis v1 e v2



O que se deseja é encontrar o mínimo global da função, que no exemplo plotado pode ser facilmente visualizado. Mas, a figura 20 traz uma imagem somente com o intuito de ilustrar o que se deseja. As funções que devem ser analisadas para determinação de mínimo possuem muitas mais variáveis.

A disciplina do cálculo permite encarar esse problema de forma a se encontrar o mínimo de uma forma analítica. Calculando-se as derivadas e usando-as para determinar onde em existe um extremo. Mas, as maiores redes neurais têm muitas variáveis, chegando a uma ordem de grandeza de bilhões de pesos e *biases*, o que não faz desse método o mais adequado.

Tomando como objeto de partida para análise a função representada pela figura 20, um pequeno deslocamento 1­ na direção v1 e um pequeno deslocamento 2­ na direção v2 pode ser descrita por meio do cálculo como sendo:

Equação 8. Mudança de no exemplo da função representada pela figura 20

Inicia-se encontrando uma forma de escolher 1 e 2 de forma a fazer negativo, ou seja, em direção e sentido como se “descesse” o gráfico da função. Para facilitar define-se como um vetor de mudança em v, 1, 2)*T*, onde *T* novamente é a operação de transposição. Define-se também o gradiente de C como sendo o vetor das derivadas parciais . O vetor gradiente é representado por , desta forma tendo a equação 8:

Equação 9

Com essas definições pode também ser entendido:

Equação 10. também pode ser entendido como:

Suponha-se que é escolhido:

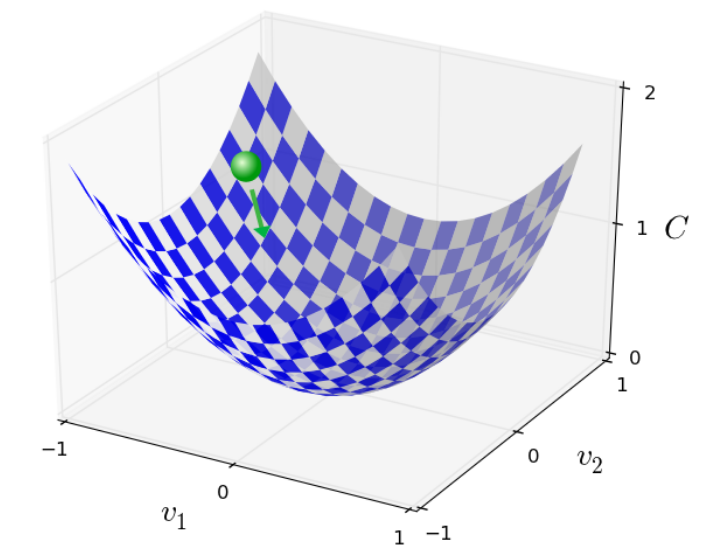
Equação 11. Escolha de taxa de aprendizado

onde é um parâmetro de pequena magnitude, positivo que é conhecido como taxa de aprendizado. A equação 10 diz que . Como , é garantido que ou seja, C sempre será decrescente. Caso efetue-se a mudança de v de acordo com a equação 11, entre os limites da aproximação da equação 10, pode-se usar a equação 11 para calcular o valor de quando se move para a posição v:

Equação 12. Cálculo de com movimentação para a posição v

E essa mesma regra será empregada então sucessivamente para os próximos deslocamentos. Repetindo-se esse deslocamento é esperável que se atinja o mínimo global, o que pode ser visualizado como exibido na figura 21, que imagina uma suposta esfera caminhando pela superfície procurando o ponto mais baixo, que é um paralelo com certa razoabilidade com como o caminho supra descrito pode ser imaginado.

Figura 21. O deslocamento em direção ao mínimo global



Para que o método do gradiente ocorra de forma adequada, é necessário escolher uma taxa η pequena o suficiente para que a equação 10 seja uma boa aproximação, caso contrário pode acontecer de , o que obviamente não é adequado. É importante também que η uma vez que isso causaria pequenos e consequentemente um algoritmo de método de gradiente muito lento. Na prática, η é comumente variado de tal forma a combinar que a equação 10 tenda a ser uma boa aproximação, enquanto o algoritmo não seja muito lento.

Até o presente instante foi feita a análise com uma função de duas variáveis, mas deve se ressaltar que tudo funciona sem nenhum problema com um maior número delas. Suponha-se um exemplo com m variáveis: v1, ..., vm. A mudança de no valor de C produzido por uma pequena variação é onde o gradiente é o vetor de acordo com a equação 13.

Equação 13. Vetor gradiente para múltiplas variáveis

Da mesma forma que no caso de duas varáveis, pode-se usar a equação 11 e ter-se a garantia que a aproximação indicada pela equação 10 será negativa, o que conduz o gradiente a um ponto de mínimo mesmo que C seja uma função de muitas variáveis, reaplicando-se repetidamente a regra da equação 12.

Esta última versão pode ser entendida como a que define o algoritmo do método do gradiente. Ela permite com que repetidas vezes mude-se a posição de v tal que se chegue ao mínimo da função. No entanto, não é uma regra a prova de falhas, o que será visto um pouco mais adiante, mas na prática, o método do gradiente funciona muito bem e é uma poderosa ferramenta à disposição das redes neurais para que se minimize a função custo.

Tanto é que existe um sentimento de que o método do gradiente seja uma ótima estratégia para encontrar mínimos. Suponha-se que se deseja mudar a posição de de forma que se decremente C tanto quanto possível. Isso equivale a minimizar . Fixando-se o tamanho da movimentação de tal forma que para algum pequeno , ou seja, deseja-se um movimento que seja um pequeno passo de tamanho fixo enquanto procura-se decrementar C o máximo possível. Pode-se provar que a escolha de tal que minimize é , onde é determinada pelo tamanho da restrição Então, o método do gradiente pode ser entendido como uma forma de se tomar pequenos passos nas direções que imediatamente mais decrementam C.

Um paralelo para facilitar a compreensão é o de se imaginar uma esfera descendo uma região ondulada, algo como uma bola em uma região montanhosa. Tem-se investigado diversas possibilidades para o cálculo do método do gradiente, inclusive alguns métodos que imitem uma bola física real. Essa abordagem de imitação de uma bola traz algumas vantagens, mas também uma grande desvantagem: torna-se necessário o cálculo da segunda derivada parcial de C, e isso pode ser muito custoso em termos de processamento. Para entender este custo, imagine-se que se deseja calcular a segunda derivada parcial de . Se existirem um milhão de variáveis então far-se-ão algo como um trilhão de segundas derivadas parciais. Na realidade, até mais uma vez que também é válido, e tudo isso tem um custo computacional muito elevado. Porém, ao longo deste trabalho será usado somente o método do gradiente e algumas de suas varações.

A questão é: Como aplicar o método do gradiente para determinar os pesos wk e os *bias* bl de tal forma que se minimize a equação do custo (equação 7)? Para ver isso em ação, reescreva-se a regra de atualização do método do gradiente com os pesos e *bias* substituindo as variáveis v­j, ou seja, a “posição” possui componentes wk e bl e o vetor gradiente tem os componentes correspondentes e . Temos assim as equações 14 e 15:

Equação : Componente do peso

Equação : Componete do *bias*

Aplicando-se repetidamente essa regra, consegue-se “rolar para baixo” e espera-se, atingir o mínimo. Esta deve ser, portanto, a estratégia a ser usada para a aprendizagem da rede neural neste trabalho.

Deve-se perceber um primeiro desafio, que para entender melhor voltaremos à equação quadrática do custo, a equação 7. Perceba-se que esta tem a forma geral , ou seja, é o cisto médio sobre os custos para as práticas de treinamento individuais. Isso implica, na prática que para se calcular os gradientes separados para cada entrada x, e então calcular-se a média . Acontece que, se o número de entradas de treinamento for muito grande, e quanto maior, melhor, essa abordagem implicará em um alto tempo de processamento e, portanto, longo tempo de aprendizagem.

Um conceito chamado gradiente de descida estocástico pode ser empregado para acelerar o aprendizado. A ideia é basicamente a de se estimar o gradiente através do cálculo de x para uma pequena amostra de treinamento escolhida de forma aleatória. Quando se faz a média sobre essas pequenas amostras, acaba sendo uma maneira rápida de se estimar o real a e, portanto, de se acelerar o aprendizado.

Para precisar melhor, o gradiente de descida estocástico funciona através da seleção de uma pequena quantidade de entradas de treinamento m escolhidas de forma aleatória. Rotulando-as x1, x2, ..., xm serão referenciadas como um minilote. Admitindo-se que o tamanho da amostra m seja grande o suficiente, espera-se que o valor de seja aproximadamente igual a média sobre todos os x, isto é:

Equação : O gradiente de descida estocástico sobre uma pequena amostra de dados de treino é aproximadamente o de toda a amostra

onde a segunda somatória é de todos os dados de treinamento. Mudando-se os lados tem-se:

Equação Estimativa do gradiente geral por seleção de minilote escolhido aleatóriamente

o que confirma que se pode estimar o gradiente geral por meio do cálculo de um minilote escolhido de forma aleatória.

Para que se relacione isso explicitamente com o aprendizado de redes neurais, assuma-se que wk e bl denotem os pesos e *bias* na rede neural em questão. O gradiente de decida estocástico, como consequência do que foi visto acima, é operado pela escolha aleatória de um minilote de dados de entrada de treinamento e fazneod-se o treinamento com estes:

Equação : Componente peso com o gradiente de descida estocástico

Equação : Componente *bias* com o gradiente de descida estocástico

onde a soma é sobre todos os exemplos de treinamento xj no minilote em questão. Na sequência, seleciona-se, sempre aleatoriamente, um segundo minilote e treina faz-se o treinamento. E prossegue-se assim até a exaustão dos dados de treinamento. Isso recebe como denominação completar uma época de treino.

# REFERÊNCIAS

Nielsen, Michael A., **Neural Network and Deep Learning**, Determination Press, 2019, <http://neuralnetworksanddeeplearning.com/>.

Goodfellow et al, **Deep Learning**, MIT Press, 2016, <http://www.deeplearningbook.org>.

**6.036 Introduction to Machine Learning**, MIT Open Learning Library, Massachusetts Institute of Technology, 2021

Kelleher, John D., Deep Learning (The MIT Press Essential Knowledge series) (p. 3). MIT Press. Kindle Edition.

Glassner, Andrew., **Deep Learning, Vol. 1: From Basics to Practice** (p. a). Kindle Edition.

Glassner, Andrew., **Deep Learning, Vol. 2: From Basics to Practice** (p. a). Kindle Edition.

Mining, Ethem., **Machine Learning: 4 Books in 1: Basic Concepts + Artificial Intelligence + Python Programming + Python Machine Learning. A Comprehensive Guide to Build Intelligent Systems Using Python Libraries**. Kindle Edition.

**18.01 Single Variable Calculus**, MIT Open Learning Library, Massachusetts Institute of Technology, 2021

**18.06 Linear Algebra**, MIT Open Learning Library, Massachusetts Institute of Technology, 2021

Marc Peter Deisenroth Et al., **Mathematics for Machine Learning**. Pre-publication version to be published by Cambridge University Press.

Dawani, Jay., **Hands-On Mathematics for Deep Learning: Build a solid mathematical foundation for training efficient deep neural networks**. Packt Publishing. Kindle Edition.

Tom Pratt Et al., **Docs/.NET/C# guide/Language reference/C# 6.0 draft specification**, [Introduction - C# language specification | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction?source=docs);

Price, Mark J. **C# 9 and .NET 5 – Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code** (p. 1). Packt Publishing. Kindle Edition.

Sam Wood, **A Brief History of Python**, Packt Books, [A Brief History of Python | Packt Hub (packtpub.com)](https://hub.packtpub.com/brief-history-python/);

Sanderson, Grant, **Neural Networks**, 3Blue1Brown, [(2799) Neural networks - YouTube](https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi).