Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Matemática Aplicada

Redes Neurais aplicadas

Eduardo Galvani Massino

Monografia Final

map2010 — Trabalho de Formatura

Orientador: José Coelho de Pina Junior

São Paulo
Dezembro de 2020

Esta seção é opcional e fica numa página separada;
ela pode ser usada para uma dedicatória ou epígrafe.

Agradecimentos

Do. Or do not. There is no try.
— Mestre Yoda

Texto texto. Texto opcional.

Resumo

Eduardo Galvani Massino. **Redes Neurais aplicadas**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2020.

Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas, em forma de texto. Deve apresentar os objetivos, métodos empregados, resultados e conclusões. O resumo deve ser redigido em parágrafo único, conter no máximo 500 palavras e ser seguido dos termos representativos do conteúdo do trabalho (palavras-chave). Deve ser precedido da referência do documento. Texto texto. Texto texto.

Palavras-chave: redes-neurais. perceptron.

Abstract

Eduardo Galvani Massino. **Redes Neurais aplicadas**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2020.

Elemento obrigatório, elaborado com as mesmas características do resumo em língua portuguesa. De acordo com o Regimento da Pós-Graduação da USP (Artigo 99), deve ser redigido em inglês para fins de divulgação. É uma boa ideia usar o sítio www.grammarly.com na preparação de textos em inglês. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

Keywords: neural-nets. perceptron.

Lista de Abreviaturas

CFT	Transformada contínua de Fourier (*Continuous Fourier Transform*)
DFT	Transformada discreta de Fourier (*Discrete Fourier Transform*)
EIIP	Potencial de interação elétron-íon (*Electron-Ion Interaction Potentials*)
STFT	Transformada de Fourier de tempo reduzido (*Short-Time Fourier Transform*)
ABNT	Associação Brasileira de Normas Técnicas
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de Símbolos

ω	Frequência angular
ψ	Função de análise *wavelet*
Ψ	Transformada de Fourier de ψ

Lista de Figuras

Lista de Tabelas

Lista de Programas

Sumário

Capítulo 1

Introdução

De tempos pra cá, ler e ouvir falar de **ciência de dados** tornou-se muito comum, tanto nos meios profissionais e científicos quanto na mídia. Existem atualmente aplicações em praticamente todas as áreas do conhecimento humano, da agricultura à indústria e ao entretenimento.

Uma busca rápida na *Wikipedia* (*Ciência de Dados* 2020) define ciência de dados como um conjunto de ferramentas que extrai informações ou previsões a partir de um grande volume de dados, que podem ser números, textos, áudio, vídeo, entre outros, para ajudar na tomada de decisões de negócios.

Apesar de não ser a única definição para o termo, Pedro A. Morettin e Julio M. Singer (Morettin e Singer, 2020) nos lembram que essa também é uma definição da estatística. Eles comparam o uso dos termos e apontam que o trabalho dos *cientistas de dados* diferem dos *estatísticos* apenas quando eles usam dados de natureza multimídia como áudio e vídeo, por exemplo. Mas que, uma vez que esses dados são processados e tornam-se números, as técnicas e conceitos utilizados pelos primeiros passam a ser basicamente os mesmos utilizados pelos segundos.

Na verdade, Morettin & Singer (Morettin e Singer, 2020) citam que na década de 80 houve uma primeira tentativa de aplicar o rótulo *ciência de dados*, (*Data Science*), ao trabalho feito pelos estatísticos aplicados da época, como uma forma de dar-lhes mais visibilidade. Curiosamente, fato mencionado pelos autores, existem atualmente cursos específicos de ciência de dados em universidades ao redor do mundo, mas a maioria deles situada em institutos de áreas aplicadas como engenharia e economia, e raramente nos institutos de estatística propriamente ditos.

Para entender um pouco mais de seu escopo, David M. Blei e Padhraic Smyth (Blei e Smyth, 2017) discutem ciência de dados sob as visões estatística, computacional e humana. Eles argumentam que é a combinação desses três componentes que formam a essência do que ela é e, assim como, do conhecimento que ela é capaz de produzir.

Em resumo, a estatística guia a coleta e análise dos dados. A computação cria algoritmos, técnicas de processamento e gerenciamento de memória eficazes para que sua execução seja efetiva. E o papel humano é o de avaliar quais tipos de dados, técnicas de análises,

algoritmos e modelos são apropriados para responder ao problema em questão. Este é o papel do *cientista de dados*.

Algoritmos de **aprendizado de máquina** vem sendo utilizados em grande parte dos modelos de ciência de dados. Mas o que é aprendizado de máquina? Ou então, o que significa dizer que o computador, neste caso a "máquina", está *aprendendo*?

Aurélien Géron (Géron, 2019) nos dá uma ideia geral lembrando que uma das primeiras aplicações de sucesso de aprendizado de máquina foi o filtro de *spam*, criado na década de 90. Uma das fases de seu desenvolvimento foi aquela em que os usuários assinalavam que certos e-mails eram *spams* e outros não eram. Hoje em dia, raramente temos que marcar ou desmarcar e-mails, pois a maioria dos filtros já "aprenderam" a fazer seu trabalho de forma muito eficiente, não temos mais nada a "ensiná-lo".

O conceito de aprendizado de máquina está intimamente ligado à ciência da computação. Porém, no contexto de ciência de dados, é definido por Joel Grus (Grus, 2016) como a "criação e o uso de modelos que são ajustados a partir dos dados". Seu objetivo é usar dados existentes para desenvolver modelos que possamos usar para *prever* possíveis respostas à consultas. Exemplos, além do filtro de *spams* podem ser: detectar transações de crédito fraudulentas, calcular a chance de um cliente clicar em uma propaganda ou então prever qual time de futebol irá vencer o Campeonato Brasileiro.

Como ficará claro ao longo deste trabalho, o aprendizado consiste na utilização de dados já conhecidos para ajustar parâmetros de modelos. Uma vez ajustados os parâmetros, o algoritmo que descreve o modelo passa a ser usado para responder às consultas. Essa fase de ajuste de parâmetros é chamada de aprendizado ou treinamento.

Uma **rede neural** é um exemplo de modelo preditivo de aprendizado de máquina que foi criado com inspiração no funcionamento do cérebro biológico. David Kopec (Kopec, 2019) descreve que apesar de terem sido as primeiras a serem criadas, elas vem ganhando nova importância na última década, graças ao avanço computacional, uma vez que exigem muito processamento, e também porque podem ser usadas para resolver problemas de aprendizagem dos mais variados tipos.

Atualmente existem vários tipos de redes neurais, porém este trabalho lida principalmente com aquele tipo que foi originalmente criado sob a inspiração do funcionamento do cérebro, chamado de **perceptron**, e que portanto tenta imitar o comportamento dos neurônios e suas conexões, aprendendo padrões a partir de dados existentes e tentando prever o comportamento de dados novos a partir do padrão aprendido.

Uma visão geral da arquitetura do aprendizado de máquina, situando a posição das redes neurais e do algoritmo *perceptron* em toda esta estrutura, assim como exemplos de aplicações em cada uma das suas ramificações, estão no Capítulo 2.

Neste trabalho é utilizada como base uma versão simples do algoritmo *perceptron* feita e explicada por Kopec (Kopec, 2019), e a partir desta base, foram criados novos métodos de treinamento e de validação com algumas estratégias, como uma tentativa de automatizar e otimizar o processo de treinamento do algoritmo, que é comumente feito de forma heurística. Detalhes dessa implementação estão no Capítulo 3.

São apresentados os conceitos e usos das séries temporais de dados não-lineares no Capítulo 4, assim como alguns exemplos de aplicações na área financeira. As técnicas tradicionais de análise e de previsão de séries temporais de dados serão brevemente apresentadas neste capítulo.

Para servir de validação e aplicação do algoritmo criado, no Capítulo 5 serão feitas comparações de desempenho entre o *perceptron* e os modelos tradicionais de previsões de séries temporais apresentados no capítulo anterior, como o **SARIMA** (*seasonal auto regressive integrated moving average*). Tais comparações usam como inspiração trabalhos similares como o de Alsmadi, Omar, Noah e Almarashdah (Alsmadi *et al.*, 2009).

Ao final concluo sobre os modelos de previsões aqui estudados e comparados, destacando a qualidade e eficiência dos métodos de redes neurais, tão bons e às vezes melhores do que os métodos tradicionais estatísticos.

Capítulo 2

Redes neurais artificiais

Neste capítulo são apresentados alguns conceitos básicos de **aprendizado de máquina**, com foco nos algoritmos de redes neurais artificiais, em especial o **perceptron**, que teve seu desenvolvimento inspirado nas redes neurais biológicas, ou seja, os neurônios e suas conexões no cérebro, conforme descrito por Kopec (Kopec, 2019).

Pode-se classificar as técnicas de aprendizado de várias formas, de acordo com alguma de suas características. Por exemplo, Géron (Géron, 2019) utiliza o grau de supervisão humana durante o seu funcionamento para classificá-los em aprendizado supervisionado ou não-supervisionado. Durante o aprendizado podem ou não ser fornecidos um conjunto de consultas e de respostas esperadas. Tais respostas foram dadas por humanos, ao menos neste momento, daí o termo "supervisão humana".

Neste texto os termos "algoritmo" e "técnica" serão usados livremente como sinônimos, pois uma técnica de aprendizado de máquina, no contexto atual, é um algoritmo executado no computador que tem por objetivo ajustar parâmetros de modelos estatísticos.

2.1 Aprendizado de máquina supervisionado

Um algoritmo de **aprendizado supervisionado** é usado quando conhecemos características dos dados que estamos utilizando. De modo geral já temos de antemão as respostas às consultas para os dados utilizados no treinamento. Por exemplo, se estamos classificando fotos de animais, possuímos um conjunto de fotos em que já sabemos quais são de gatos, cachorros, etc.

O ato de rotular previamente os dados que usamos no treinamento é o que designamos de supervisão humana. Uma vez *treinado*, o algoritmo recebe uma foto, ou seja, uma nova consulta e então fornece a resposta, neste caso se essa é a foto de um gato, ou cachorro, ou qualquer outra resposta dentre aquelas que foram dadas como exemplos durante o treinamento.

Dentro do aprendizado supervisionado temos duas técnicas principais. A primeira é a regressão, usada para prever valores, ou seja, fornecer respostas a consultas ainda inéditas,

sejam dados do futuro ou valores de funções em pontos do domínio para os quais ainda não existem respostas.

A segunda técnica é a classificação, usada para rotular ou dividir os dados em classes pré-determinadas, a partir de exemplos, que é exatamente o caso dos exemplos descritos nos parágrafos anteriores. Se as classes não são conhecidas deve-se utilizar um algoritmo de aprendizado não-supervisionado, descrito na próxima seção.

Colocar exemplos...

2.2 Aprendizado não-supervisionado

Nesse tipo de aprendizado de máquina, não sabemos os rótulos dos dados que estamos lidando, assim o algoritmo poderá agrupar os dados de forma automática, por exemplo, se estivermos lidando com problemas de classificação. Aqui, as consultas podem ser coisas como "quantos são os perfis dos clientes" ou "quantas espécies de flores existem nestas fotos", e assim por diante.

Alguns métodos não-supervisionados de aprendizado foram enumeradas por Géron (Géron, 2019). O **agrupamento** de dados similares sob uma inspiração geométrica. Nesse caso os dados são agrupados conforme suas posições num determinado espaço e utiliza-se algoritmos como k-vizinhos, k-means, k-medians, etc. Exemplos de aplicações são agrupamento de produtos em supermercados, interesses comuns de clientes em sites de conteúdo digital, etc.

Outra técnica é a **detecção de anomalias**, cujo objetivo é ter uma descrição de como os dados considerados "normais" se parecem, e usa-se esse agrupamento para detectar se novos dados estariam "fora" desse padrão. Um exemplo é a detecção de fraudes.

Também pode-se citar sobre a técnica de **estimação de densidades**, que tem como objetivo a estimação da função densidade de probabilidade de um conjunto de dados gerados por algum processo aleatório.

Colocar exemplos...

2.3 Técnicas de classificação

É uma das técnicas principais do aprendizado supervisionado, problemas desse tipo buscam aprender com um conjunto de dados previamente rotulados. Consultas do tipo "a qual grupo pertence este cliente?" ou "que animal há nesta foto?" podem ser modeladas por esses algoritmos. De modo geral, ele responde a consultas que dizem respeito às classes dos dados, e atribui para novos dados alguma classe que pertence ao conjunto de classes que usamos para rotular os dados iniciais.

Existem vários tipos de algoritmos de classificação, e dentre os descritos por Géron (Géron, 2019) citamos a máquina de vetor suporte (*support vector machine*, SVM), árvores de decisão, florestas aleatórias que são um conjunto de muitas árvores de decisão aleatoriamente definidas e, finalmente, as redes neurais artificiais.

Todas essas técnicas podem ser usadas para classificação linear ou não-linear, no sentido em que valores eles estão classificando, assim como na forma que está sendo feita essa classificação. Se visualizarmos os dados num espaço bidimensional, um algoritmo de classificação linear irá separar as classes de dados por retas, enquanto que um classificador não-linear poderá usar outra curva qualquer para a separação.

Abstraindo o espaço bidimensional para os espaços multidimensionais dos dados que são comumente analisados, podemos pensar em hiperplanos, estruturas $(n-1)$-dimensionais de espaços n-dimensionais, para o caso dos classificadores lineares, ou subespaços quaisquer para os não-lineares.

Colocar exemplos de aplicações...

2.4 A rede neural perceptron

Uma rede neural artificial é um dentre vários métodos de classificação, ou seja, de aprendizado supervisionado, embora ela também possa ser usada para aplicaçõesd de aprendizado não supervisionado. De acordo com Kopec (Kopec, 2019), ele é utilizado como um classificador não-linear, e por isso pode ser utilizado para classificar ou prever quaisquer tipos de funções, que podem ou não ter uma relação linear com o tempo ou com qualquer outro domínio no qual estejam definidas.

Uma definição para uma rede neural artificial dada por Rosangela Ballini (Ballini, 2000) é a de um sistema de processamento paralelo e distribuído baseado no sistema nervoso biológico, sendo compostos por elementos computacionais chamados neurônios, arranjados em padrões semelhantes às redes biológicas.

Na figura 2.1 está uma representação de um neurônio biológico. Ele recebe impulsos elétricos de entrada através dos dentritos, que são transmitidos ou não através do núcleo, caso sejam ativados por ele, para os terminais de saída dos axônios. Os neurônios se comunicam através de sinapses, que são ligações entre os dentritos de um e os axônios de outro que realizam a transmissão dos sinais.

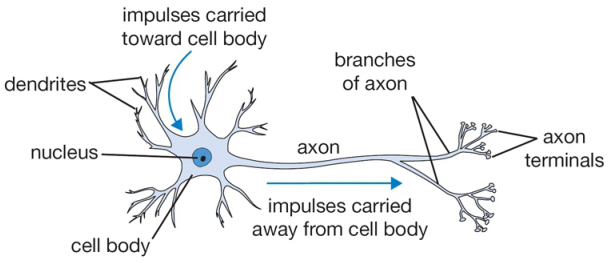

Figura 2.1: *Representação de um neurônio biológico.*[a]

[a]Extraído de https://cs231n.github.io/neural-networks-1/

Dá-se o nome de *perceptron* de camada única (*single-layer perceptron*) ou simplesmente *perceptron* a uma das primeiras e mais simples rede neural artificial a ser criada. Uma ilustração conceitual dela está na figura 2.2. Mais recentemente foram criadas várias

outras versões dessa rede, dentre as quais podemos citar o *perceptron* de multi-camadas (*multi-layer perceptron*).

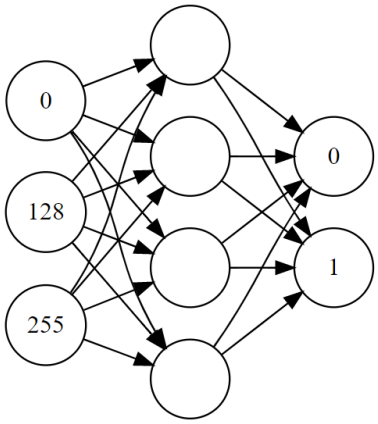

Figura 2.2: *Rede neural simples, o perceptron de camada única.*

Os neurônios são representados por círculos, dentro deles há um valor numérico que intuitivamente podemos atribuir ao nível ou grau de ativação do neurônio, mesmo que no caso biológico se restrinja aos valores 0 e 1, ou seja, ativados ou não. Cada coluna de neurônios representa uma camada, nesse caso, da esquerda para a direita temos a camada de entrada, a camada oculta e a camada de saída. As linhas representam as ligações entre os neurônios, sendo que cada neurônio de uma camada está ligado a todos da camada anterior.

O perceptron de camada única consiste de uma camada de neurônios de entrada, uma camada oculta de neurônios usados na otimização, e uma camada de saída, que irá conter os dados previstos, ou ainda as probabilidades do dado pertencer a alguma das classes que a rede poderá classificá-lo. E é o fato de haver uma camada oculta nesta rede que a define como sendo de "camada única". Caso houvessem mais do que uma camada oculta, ela seria do tipo "multi-camadas" mencionada acima.

De modo a entendermos as bases matemáticas do algoritmo, podemos começar de uma rede ainda mais básica, a partir um *perceptron* que seja constituído de apenas 1 neurônio na única camada oculta. Esta rede super simplificada, que está na figura 2.3, pode ser útil para para o entendimento uma vez que neste caso será possível acompanhar graficamente o resultado da execução do algoritmo.

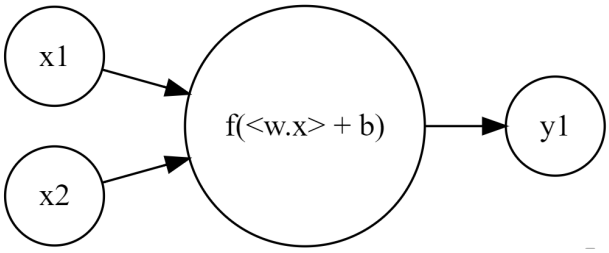

Figura 2.3: *Rede neural mais simples ainda, apenas um neurônio oculto.*

Esta rede possui 2 neurônios na camada de entrada, que são os números reais x_1 e x_2, 1 neurônio na camada oculta, no qual está a sua função de ativação $f(x_1 w_1 + x_2 w_2 + b)$,

e 1 neurônio na camada de saída, que neste caso é um número real y_1. Pode-se notar a semelhança dessa rede neural artificial com a sua inspiração biológica com a ajuda da figura 2.4.

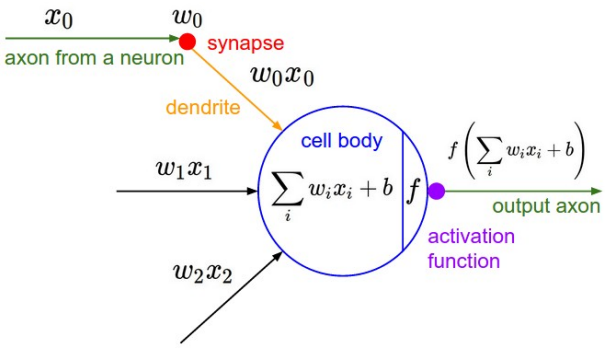

Figura 2.4: *Representação de um neurônio artificial.*[a]

[a]Extraído de https://cs231n.github.io/neural-networks-1/

Temos os sinais de entrada (como o x_1) vindos como viriam os sinais dos axônios de outros neurônios. Eles entram pela camada de entrada da rede, ou dentritos do neurônio. A camada oculta processa as entradas com os pesos, definindo o sinal através de sua função de ativação, similar ao funcionamento do núcleo do neurônio que ativa ou não o sinal recebido. Por fim o sinal é enviado à camada de saída, ou aos axônios do neurônio, concluíndo o processamento.

A partir desta analogia podemos compreender o funcionamento básico da rede artificial *perceptron*. Ela recebe uma lista de valores como entrada, que podemos representar por um vetor real x. O neurônio oculto representa uma transformação linear neste vetor, que podemos escrever como o produto escalar por um outro vetor real, o vetor de **pesos** w, ou seja, $< w.x >$, que é o produto escalar usual dos números reais. A seguir, somamos um outro número real b que é chamado de **viés**, que possui o mesmo papel que a constante de interceptação da reta com o eixo vertical de um ajuste linear.

Por fim, é aplicada uma função de ativação não-linear sobre esta transformação, o que configura a saída deste neurônio: $f(< w.x > + b)$, que é transmitida ao neurônio de saída, que pode aplicar uma transformação semelhante ou outra qualquer, dependendo da função de ativação utilizada em cada camada da rede. Por simplicidade mostramos uma rede bem simples, mas na prática podem haver muito mais camadas ocultas, e cada uma delas assim como a camada de saída, podem ter muitos neurônios cada.

Este processo de entrada, processamento e saída da rede é chamado de **feedforward**, e consiste no nível mais fundamental do *perceptron*. A partir daí, a forma como a rede será treinada, é o que define se ela será utilizada para um aprendizado supervisionado ou não-supervisionado.

Uma vez que estamos lidando com o aprendizado supervisionado, dever ser utilizado um algoritmo de treinamento que forneça à rede pares conhecidos de vetores de entradas e saídas esperadas, e um critério de avaliação de quão boa é a performance da rede para aproximar as suas saídas das saídas esperadas.

Este critério é uma função que fornece uma medida da distância entre as saídas obtidas pela rede e as saídas esperadas, que é genericamente chamada de função de custo (*cost function*). Se denotarmos por y uma saída conhecida, e por $a^{(L)}$ uma saída obtida pela última camada, exemplos comumente usados são as normas usuais como a distância euclidiana ($(a^{(L)^2} + y^2)^{1/2}$), a função de erro absoluto ($|a^{(L)} - y|$), e a função de erro quadrático médio ($(a^{(L)} - y)^2$) , (*mean square error*, MSE), que é a usada no algoritmo descrito por Kopec (Kopec, 2019) e que será usado neste trabalho.

2.5 Derivação matemática de um algoritmo de aprendizado

Um dos algoritmos de treinamento que minimizam uma função de custo é o gradiente descendente (*gradient descent*), que segundo Géron (Géron, 2019) é um algoritmo muito geral e que serve para encontrar soluções ótimas para uma grande variedade de problemas de otimização. A ideia geral é ajustar parâmetros iterativamente para otimizar a função de custo gradativamente.

O seu funcionamento utiliza a ideia fundamental do Cálculo em que utilizamos a derivada de uma função de forma a encontrar seus pontos extremos. Dada uma função $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, temos que se um ponto $x = \hat{x} \in \mathbb{R}$ é um ponto extremo de f então é condição necessária[1] que cada derivada parcial de primeira ordem de f exista e seja igual a zero. Denotando $x = (x_0, x_1, \dots, x_n)$ e $\hat{x} = (\hat{x_0}, \hat{x_1}, \dots, \hat{x_n})$, temos:

$$\frac{\partial f(\hat{x_0})}{\partial x_0} = 0, \quad \frac{\partial f(\hat{x_1})}{\partial x_1} = 0, \quad \dots, \quad \frac{\partial f(\hat{x_n})}{\partial x_n} = 0 \tag{2.1}$$

Usando a notação de vetores, podemos simplificar a equação acima, uma vez que o conjunto das derivadas parciais de uma função de várias variáveis é o vetor gradiente desta função, assim, denotando $\mathbf{0} \in \mathbb{R}^n = (0, 0, \dots, 0)$, temos equivalentemente à equação 2.1:

$$\nabla f(\hat{x}) = \mathbf{0} \tag{2.2}$$

onde $\nabla : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ é a função que calcula o gradiente para um dado ponto de uma função.

Geometricamente é nos dada a intuição, por Luis Hamilton Guidorizzi (Guidorizzi, 1986), de que o vetor gradiente de um dado ponto de uma função nos dá a direção de maior aumento da função naquele ponto. Como nosso objetivo é minimizar a função de custo, fica explicado o nome do algoritmo como "gradiente descendente", de forma que devemos utilizar o sentido negativo do vetor gradiente.

Assim, podemos dizer que a direção de minimização da função está na direção do vetor gradiente, o que significa dar um passo ($f(x + dx)$) nessa direção no domínio da função, tal passo com tamanho que seja *proporcional* a cada componente do vetor gradiente. Dessa

[1] Mais detalhes em Guidorizzi (Guidorizzi, 1986). Um curso de cálculo Vol. 2, pág. 894.

forma podemos escrever dx como sendo um passo na direção do mínimo da função de custo dessa forma:

$$dx = -\eta \nabla f(x) \tag{2.3}$$

onde η é a constante de proporcionalidade, que é conhecida como **taxa de aprendizado**, que tem por objetivo tornar a velocidade do treinamento ajustável durante a execução do algoritmo, sendo tarefa do cientista de dados testar e obter os valores que dêem os melhores resultados caso-a-caso.

Podemos observar as diferenças de utilizar uma taxa de aprendizado fixa ou variável (que mude a cada iteração do treinamento, por exemplo), utilizando gráficos. Suponha que estamos querendo minimizar uma função quadrática do tipo $f(x) = x^2$, essa restrição é particularmente útil uma vez que a função de custo que será utilizada, a MSE, é uma função quadrática deste tipo. Outra característica igualmente útil é que a segunda derivada deste tipo de função quadrática é sempre positiva[2], o que implica que o ponto extremo encontrado será necessariamente um ponto de mínimo.

Inicialmente, sorteamos um ponto inicial e calculamos o valor da função e o valor do gradiente neste ponto. Se as componentes do gradiente não forem todas nulas (ou arbitrariamente próximas de zero), então quer dizer que não atingimos o mínimo, e dessa forma obtemos um novo ponto a partir deste somado com dx definido acima na equação 2.3. Repetimos este processo até que o gradiente do ponto atual seja arbitrariamente próximo do vetor nulo. Uma visualização deste processo, utilizando taxa de aprendizado fixada, está na Figura 2.5.

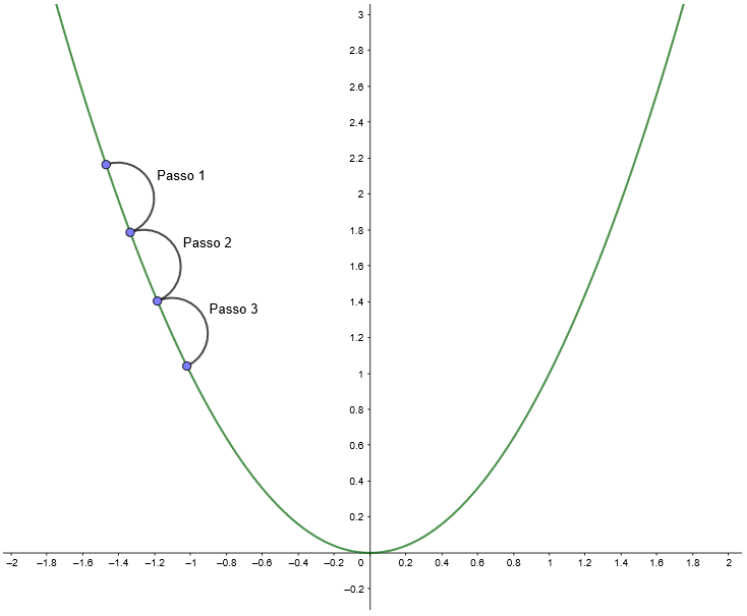

Figura 2.5: *Visualização do método do gradiente descendente com taxa de aprendizado única. Os pontos azuis representam candidatos a ponto mínimo em cada iteração do algoritmo.*

[2] A segunda derivada de uma função de muitas variáveis é a matriz Hessiana, neste caso ela seria uma matriz definida positiva.

Podemos notar que as estimativas aproximam-se a uma velocidade constante do ponto de mínimo, que nesse caso ilustrativo é bem conhecido. Esse é o comportamento gerado por uma taxa de aprendizado fixa, e além disso com uma magnitude mediana. O que poderia acontecer se utilizarmos uma taxa de aprendizado muito grande é que com um passo do algoritmo, o ponto estimado poderia ir para o outro lado do arco da função, e depois retornar, e assim por diante, nunca convergindo para o mínimo, uma ilustração disso está na Figura 2.6.

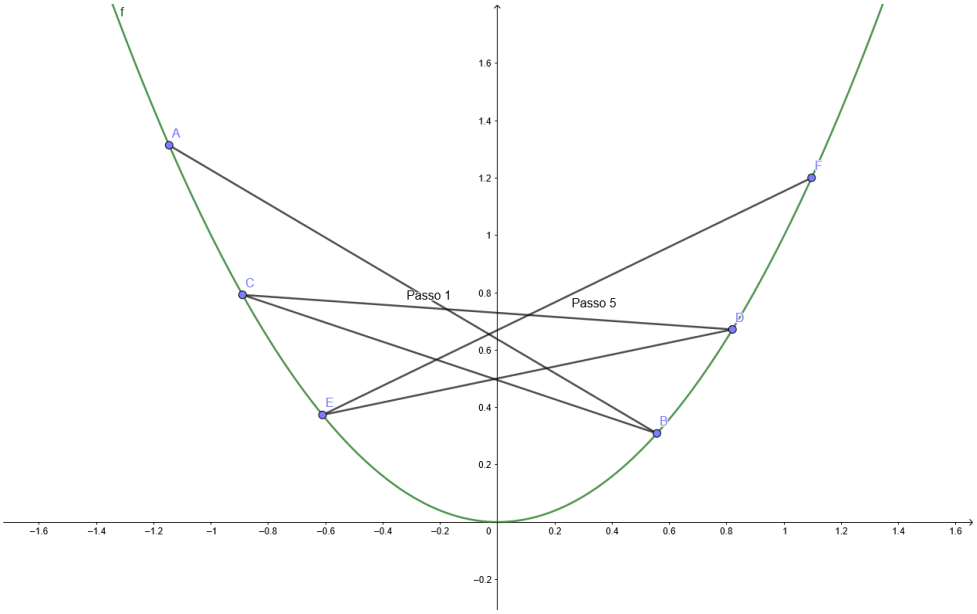

Figura 2.6: *Visualização do método do gradiente descendente com taxa de aprendizado única. Ilustração do uso de um valor de taxa de aprendizado muito grande.*

O caso oposto a este, ou seja, usar uma taxa muito pequena, claramente irá fazer com os passos dados sejam muito pequenos, e dessa forma o algoritmo demore muito a convergir, por isso é importante usar valores medianos que podem ser obtidos de forma heurística, embora na prática, conforme dito por Géron (Géron, 2019), utiliza-se $\eta = 0.1$, sendo este um valor consensualmente utilizado pelo menos como ponto de partida.

A outra abordagem é utilizar valores variáveis, sendo o caso mais comum utilizar uma taxa que começa até mesmo maior do que o valor comum de 0.1 mas que vai diminuindo a cada passo, numa tentativa de obter uma convergência mais rápida. Uma ilustração desse caso está na Figura 2.7.

Qualquer que seja o tipo de taxa de apredizado que venha a ser utilizado, permanece como melhor estratégia testar qual deles irá gerar o melhor resultado, a partir de algum problema com solução previamente conhecida, como a função $f(x) = x^2$, em que sabemos que o ponto de mínimo é atingido em $x = 0$, ou podemos testar em nosso problema-alvo, analisando diretamente os valores candidatos a mínimo obtidos pelo algoritmo como função do número do passo, criando assim outro tipo de gráfico, no qual não precisamos saber o formato da função objetivo, o que é razoável uma vez que não precisariamos de um método numérico para obter seu ponto de mínimo.

Podemos observar este comportamento genérico para os 4 casos acima mencionados,

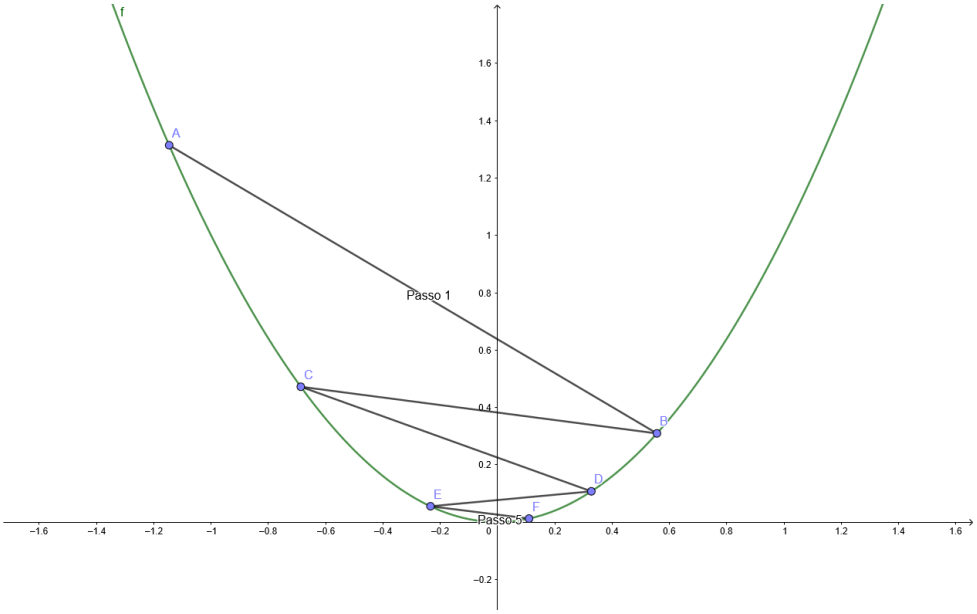

Figura 2.7: *Visualização do método do gradiente descendente com taxa de aprendizado variável que vai diminuindo passo-a-passo do algoritmo.*

na Figura 2.8 temos os gráficos de valores hipotéticos de candidatos a mínimo gerados por (a) taxa de aprendizado fixa e grande, (b) taxa de aprendizado fixa e pequena, (c) taxa de aprendizado fixa e mediana, (d) taxa de aprendizado decrescente.

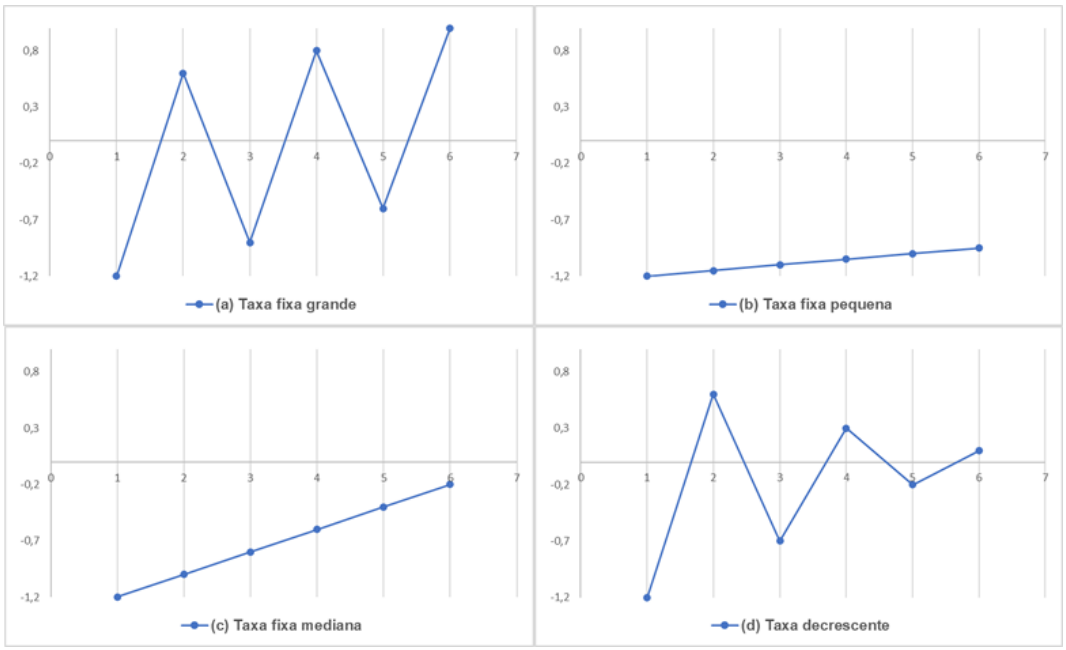

Figura 2.8: *Comportamento de diferentes taxas de aprendizado nos valores candidatos a mínimo.*

A partir deste comportamento geral, podemos testar nosso problema-alvo, verificar a qual comportamento ele mais se parece e assim decidir se devemos aumentar ou diminuir nossa taxa até obtermos um bom comportamento como aqueles vistos em (c) ou (d).

Colocar exemplos de outros algoritmos de otimização de função de custo...

2.6 Sistemas nebulosos

Os sistemas nebulosos (*fuzzy systems*) foram criados a partir da teoria dos conjuntos nebulosos (*fuzzy sets*) criada por Lotfi Aliasker Zadeh (Zadeh, 1965). Os conjuntos nebulosos diferem dos conjuntos clássicos na forma em que "avaliamos" se os elementos pertencem aos conjuntos ou não.

Num conjunto clássico um elemento pertence a ele ou não, não há meio termo. Já nos conjuntos nebulosos existe uma função associando o grau com que um elemento pertence ao conjunto, a função assume 0 se não pertence, ou seja, uma exclusão completa, e assume 1 se pertence completamente, mas também assume qualquer valor real dentro deste intervalo. Esta é a chamada função de pertinência, de forma que a frase "o elemento x percente mais ao conjunto A do que o elemento y", por exemplo, é a representação verbal da expressão matemática $f(x) > f(y)$ se usarmos a definição de ordem dos números reais e da notação $f(.)$ para a função de pertinência.

Os sistemas nebulosos podem ser usados para modelar situações do mundo real que não podem ser muito bem representadas pelos conjuntos clássicos com seus limites muito bem definidos, mas melhor representadas pelos conjuntos nebulosos. Um exemplo ilustrativo é o do conceito de *perto*.

Usar imagens e gráficos para mostrar esse exemplo aqui...

A partir disto, conforme descrito por Li-Xin Wang (Wang, 1992), os sistemas nebulosos são aproximadores universais. Isto significa que podem aproximar quaisquer funções numa grande variedade de de problemas. Neste artigo é demonstrado que com uma função de pertinência gaussiana é possível usar sistemas nebulosos para aproximar qualquer função real contínua definida num conjunto compacto.

Capítulo 3

Perceptron multi-camadas

Neste capítulo é descrita a implementação e funcionamento de uma versão do algoritmo *perceptron*, feito a partir de um núcleo básico disponibilizado no livro de Kopec (Kopec, 2019), e a partir do qual foram feitas modificações e criação de novos métodos de treinamento, de validação e de avaliação do treinamento.

O perceptron aqui implementado pode ser usado tanto para tarefas de classificações, quanto para tarefas de regressões de dados, e neste capítulo são apresentados exemplos de ambos os usos, embora o foco deste trabalho seja na versão de classificação.

3.1 Derivação matemática do algoritmo de retropropagação

Para a aprendizagem supervisionada foi utilizado o algoritmo de retropropagação (*retropropagation*), que consiste na minimização de uma função de custos, a partir do gradiente, ou seja, da derivada desta função de custos, neste caso o erro quadrático médio, conforme foi definido no capítulo anterior.

De acordo com Kopec (Kopec, 2019), o perceptron consiste de uma rede cujo sinal, ou seja, os dados, se propagam em uma só direção, da camada de entrada para a camada de saída, passando pelas camadas ocultas uma a uma, e por isso o nome de rede *feedforward* ao perceptron. Por sua vez, o erro que determinamos na camada final propaga-se no caminho inverso, sendo distribuídas correções da saída para a entrada, afetando aqueles neurônios que foram mais responsáveis pelo erro total. Por isso o nome de retropropagação.

Estendendo as definições já usadas no capítulo anterior, segue a derivação matemática do algoritmo de retropropagação. Como ficará claro mais à frente, podemos derivar as contas para apenas um neurônio por camada sem perda de generalidade. Dessa forma, se temos uma rede com L camadas, o erro quadrático para um neurônio da camada de saída (a camada L) será:

$$C_0 = (a^{(L)} - y)^2$$

onde y é a saída esperada, e $a^{(L)}$ é a saída de um neurônio da camada de saída.

Temos que C_0 é uma função de $a^{(L)}$, uma vez que y é um valor fixo conhecido. Por sua vez, temos que de modo geral a saída de um neurônio é uma função do tipo:

$$a^{(L)} = \sigma(w^{(L)}\ a^{(L-1)} + b^{(L)})$$

onde escrevemos $a^{(L-1)}$ é a saída do neurônio da camada anterior, $w^{(L)}$ é o **peso** atribuido a essa saída, o que seria o parâmetro angular A na Figura 2.3, e $b^{(L)}$ é o chamado **viés** desse neurônio, análogo ao parâmetro linear de uma reta. Por fim temos a **função de ativação** que escrevemos como σ que é aplicada à essa equação linear.

Nota-se que internamente à função de ativação, um neurônio se comporta como uma transformação linear dos neurônios da camada anterior. Caso tivéssemos n neurônios na camada anterior à de saída, teríamos então n pesos, denotados com índice i dessa forma: $\{w_i^{(L)}\}_{i=1}^{n}$. Cabe assim à função de ativação, dar o comportamento não-linear à rede perceptron.

Como o objetivo é minimizar C_0, temos que calcular a influência dos pesos e dos viéses nesse custo. Já sabemos que isso será obtido com o gradiente, isto é, a derivada dessa função em relação a esses parâmetros que, são os únicos que podemos otimizar. De forma mais clara, temos que no início do treinamento da rede, atribuímos valores aleatórios aos pesos e aos viéses, e então executamos o *feedforward*, de forma que a rede irá calcular sequencialmente os valores de saída em todas as suas camadas, obtidos a partir dos dados de entrada, que serão fixos, e desses parâmetros inicialmente aleatórios. A partir daí, poderemos otimizar esses parâmetros, exatamente da forma que estamos construindo.

O cálculo dessas derivadas é feito segundo a regra da cadeia, e adicionalmente iremos denotar a transformação linear interna à função de ativação por $z^{(L)} = w^{(L)}\ a^{(L-1)} + b^{(L)}$, de forma que $a^{(L)} = \sigma(z^{(L)})$. Assim, ficamos com as derivadas para a camada de saída:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \tag{3.1}$$

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \tag{3.2}$$

Para a camada de saída, podemos calcular diretamente cada termo dessas derivadas:

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y) \ \propto \ (a^{(L)} - y) \tag{3.3}$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma^{'}(z^{(L)}) \tag{3.4}$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)} \tag{3.5}$$

$$\frac{\partial z^{(L)}}{\partial b^{(L)}} = 1 \tag{3.6}$$

O que resulta, fazendo todas as substituições, em:

$$\frac{\partial C_0}{\partial w^{(L)}} = a^{(L-1)}\, \sigma'(z^{(L)})\, (a^{(L)} - y) \tag{3.7}$$

$$\frac{\partial C_0}{\partial b^{(L)}} = \sigma'(z^{(L)})\, (a^{(L)} - y) \tag{3.8}$$

Na equação 3.3 ocultamos o termo constante 2 sob um símbolo de proporção, que a seguir iremos também ocultar, uma vez que usaremos o algoritmo do gradiente descendente, e assim, em seu lugar, e na verdade, todas as derivadas aqui mostradas serão multiplicadas pelo termo η, a **taxa de aprendizagem**, conforme explicado no capítulo anterior.

Analogamente, podemos pensar numa forma de fazer esses cálculos para as camadas ocultas. A princípio, podemos calcular:

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \tag{3.9}$$

Usando o fato de que:

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = w^{(L)} \tag{3.10}$$

Agora, seja a i-ésima camada oculta tal que $1 < i < L$, se observarmos a equação 3.9, e fizermos $i = L - 1$, usando a equação 3.10, ficamos com:

$$\frac{\partial C_0}{\partial a^{(i)}} = w^{(i+1)} \frac{\partial a^{(i+1)}}{\partial z^{(i+1)}} \frac{\partial C_0}{\partial a^{(i+1)}} \tag{3.11}$$

Podemos observar que há um mesmo termo duplo que aparece tanto nas equações 3.1 e 3.2 quanto na equação 3.11 acima, de forma que apenas o índice da camada é diferente. Para simplificar podemos nomear esse termo de *delta da camada i*:

$$\Delta^{(i)} = \frac{\partial a^{(i)}}{\partial z^{(i)}} \frac{\partial C_0}{\partial a^{(i)}} \tag{3.12}$$

Simplificando todas as demais expressões usando essa definição, ficamos com:

$$\frac{\partial C_0}{\partial w^{(i)}} = a^{(i-1)} \Delta^{(i)} \tag{3.13}$$

$$\frac{\partial C_0}{\partial b^{(i)}} = \Delta^{(i)} \tag{3.14}$$

Como vemos, as derivadas que precisamos todas dependem desse termo Δ, que por sua vez depende do cálculo do termo $\frac{\partial C_0}{\partial a^{(i)}}$ que será calculado de 2 formas distintas:

$$\begin{aligned} &\frac{\partial C_0}{\partial a^{(i)}} = w^{(i+1)}\, \Delta^{(i+1)} \implies \\ &\Delta^{(i)} = \sigma'(z^{(i)})\, w^{(i+1)}\, \Delta^{(i+1)} \end{aligned} \tag{3.15}$$

para as camadas ocultas.

$$\begin{aligned} &\frac{\partial C_0}{\partial a^{(L)}} = (y - a^{(L)}) \implies \\ &\Delta^{(L)} = \sigma'(z^{(L)})\, (y - a^{(L)}) \end{aligned} \tag{3.16}$$

para a camada de saída.

Percebe-se a natureza recursiva do algoritmo, onde o caso base é calculado na camada de saída, e que o cálculo vai propagando-se para as camadas ocultas, em direção à camada de entrada. Por essa mesma razão, pudemos derivar as contas para uma camada, e no fim elas estão prontas pra serem implementadas para qualquer número de camadas ocultas.

Outro fato útil é que a expressão interna do neurônio é uma transformação linear, assim as contas podem ser facilmente ajustadas para o caso geral em que há n_i neurônios em dada camada i da rede, conforme já explicado, e que será detalhado diretamente nos trechos de código que serão mostrados a seguir na implementação propriamente dita.

3.2 Implementação do algoritmo de retropropagação

A implementação seguiu uma estrutura orientada a objetos, voltemos então à representação visual da rede perceptron, a partir da imagem 3.1. Cada círculo representa um neurônio, cada coluna vertical de neurônios é uma camada da rede, uma das camadas, a camada oculta nesse caso, está destacada em roxo na imagem. As setas representam as conexões entre as camadas de neurônios, cada neurônio de uma camada está ligado a todos os neurônios da camada anterior, o sentido dessa conexão é da esquerda pra direita, o que indica o processo de *feedforward* da rede.

A implementação do perceptron deste trabalho teve como base a implementação feita por Kopec Kopec, 2019, a partir da qual foram adicionados outros recursos, como o viés dos neurônios, não presentes na implementação de Kopec, como o uso da biblioteca *Numpy* para o uso de seus métodos mais eficientes para lidar com listas de números de ponto flutuante. Além dessa base, também estão implementadas várias outras classes, que serão explicadas de modo mais geral.

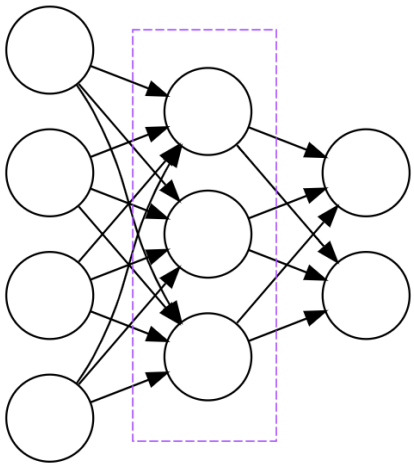

Figura 3.1: *Visão estrutural da rede perceptron. A linha tracejada destaca uma das camadas da rede.*

O primeiro passo é implementar a classe *Neuron* para representar cada neurônio. Esta é uma classe de entidade bem simples, contendo apenas um construtor, e o método *output* que recebe os valores de entrada para esse neurônio e faz o cálculo da transformação linear e a seguir aplica e retorna o valor da função de ativação utilizada, que é passada como parâmetro ao construtor da classe. A listagem 3.1 abaixo mostra este método, em conjunto com o construtor da classe.

```
1  class Neuron:
2      def __init__(self, weights, bias, learning_rate, ativacao, der_ativacao):
3          '''(list[float], float, float, Callable, Callable) -> None
4          Construtor do Neurônio
5          '''
6          self.weights = weights
7          self.bias = bias
8          self.ativacao = ativacao
9          self.der_ativacao = der_ativacao
10         self.learning_rate = learning_rate
11         self.output_cache = 0.0
12         self.delta = 0.0
13
14     def output(self, inputs):
15         '''(list[float]) -> float
16         Computa valor de ativação do neurônio, salvando o valor
17         antes da função de ativação ser aplicado e retornado
18         '''
19         self.output_cache = np.dot(inputs, self.weights) + self.bias
20         return self.ativacao(self.output_cache)
```

Programa 3.1: *Trecho da classe Neuron*

A função *np.dot* da biblioteca *Numpy* é utilizada para calcular o produto escalar entre os valores de entrada e os pesos desse neurônio. O valor da transformação linear é armazenado num atributo de classe antes da aplicação da função de ativação, pois será utilizado mais à frente durante o treinamento da rede.

Capítulo 4

Séries temporais

Neste capítulo são apresentados os conceitos básicos de séries temporais, com destaque para séries históricas de conversões de moedas estrangeiras. Tais séries estão disponíveis para *download* em diversos sites, incluindo o site do Banco Central do Brasil[1]. Também são discutidos brevemente os métodos tradicionais de análise das séries como a *transformada de Fourier* e de previsões de valores futuros da série, como o *SARIMA*.

[1] https://www.bcb.gov.br/

Capítulo 5

Comparação dos modelos

Neste capítulo serão feitas comparações entre os modelos preditivos de séries temporais, através de medidas de desempenho comuns como percentual de acerto das previsões e observações das funções de erros dos algoritmos utilizados.

Referências

[Alsmadi *et al.* 2009] M. k. Alsmadi, K. B. Omar, S. A. Noah e I. Almarashdah. "Performance comparison of multi-layer perceptron (back propagation, delta rule and perceptron) algorithms in neural networks". Em: *2009 IEEE International Advance Computing Conference* (mar. de 2009), pgs. 296–299. doi: 10.1109/IADCC.2009.4809024 (citado na pg. 3).

[Ballini 2000] Rosangela Ballini. "Análise e Previsão de Vazões Utilizando Modelos de Séries Temporais, Redes Neurais e Redes Neurais Nebulosas". Doutorado em Engenharia Elétrica. Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, 2000 (citado na pg. 7).

[Blei e Smyth 2017] David M. Blei e Padhraic Smyth. "Science and data science". Em: *PNAS* 114.33 (ago. de 2017), pgs. 8689–8692 (citado na pg. 1).

[*Ciência de Dados* 2020] *Ciência de Dados*. https://pt.wikipedia.org/wiki/Ci%C3%AAncia_de_dados. Mar. de 2020 (citado na pg. 1).

[Géron 2019] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2º. O'Reilly, 2019 (citado nas pgs. 2, 5, 6, 10, 12).

[Grus 2016] Joel Grus. *Data Science do Zero*. 1º. O'Reilly, 2016 (citado na pg. 2).

[Guidorizzi 1986] Hamilton Luiz Guidorizzi. *Um curso de cálculo*. v. 2. LTC, 1986. isbn: 8521604254 (citado na pg. 10).

[Kopec 2019] David Kopec. *Problemas Clássicos de Ciência da Computação com Python*. 1º. Novatec, 2019 (citado nas pgs. 2, 5, 7, 10, 15, 18).

[Morettin e Singer 2020] Pedro A. Morettin e Julio M. Singer. *Introdução à Ciência de Dados - Fundamentos e Aplicações*. Departamento de Estatística. Universidade de São Paulo, 2020 (citado na pg. 1).

[Wang 1992] Li-Xin Wang. "Fuzzy systems are universal approximators". Em: *[1992 Proceedings] IEEE International Conference on Fuzzy Systems* (mar. de 1992), pgs. 1163–1170. doi: 10.1109/FUZZY.1992.258721 (citado na pg. 14).

[Zadeh 1965] Lotfi Aliasker Zadeh. "Fuzzy sets". Em: *Information and Control* 8.3 (1965), pgs. 338–353. issn: 0019-9958. doi: https://doi.org/10.1016/S0019-

9958(65)90241-X. URL: http://www.sciencedirect.com/science/article/pii/S001999586590241X (citado na pg. 14).

Índice Remissivo