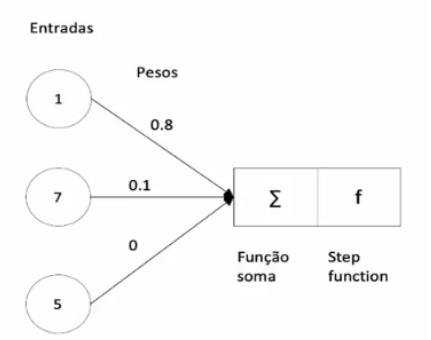
Relatório 13 - Prática: Redes Neurais (II)

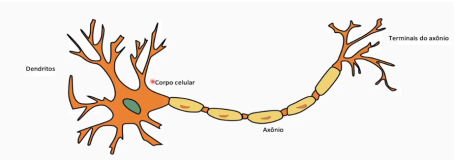
Lucas Augusto Nunes de Barros

Descrição das atividades

Aprendizagem supervisionada	Aprendizagem não supervisionada
Redes Neurais Artificiais classificação e regressão	Mapas auto organizáveis detecção de características e agrupamento
Redes Neurais Convolucionais visão computacional	Boltzmann machines sistemas de recomendação redução de dimensionalidade
Redes Neurais Recorrentes análise de séries temporais	Autoencoders redução de dimensionalidade
	Redes adversariais generativas geração de imagens



Neurônio Artificial



Neurônio Biológico

Associações entre as partes dos neurônios artificiais e biológicos:

- Entradas / Dendritos: No neurônio biológico, os dendritos recebem sinais elétricos de outros neurônios. No neurônio artificial, as entradas são os dados fornecidos ao modelo.

- Pesos / Sinapses: No neurônio biológico, as sinapses têm intensidades diferentes que afetam a transmissão do sinal. No neurônio artificial, os pesos determinam a importância (intensidade) de cada entrada.

- Soma / Corpo Celular : No neurônio biológico, o corpo celular une os sinais recebidos nos dendritos. No neurônio artificial é feita a soma das entradas (função soma).

- Função de Ativação / Potencial de Ação : No neurônio biológico, se o sinal recebido ultrapassar um limiar, um potencial de ação é disparado. No neurônio artificial, a função de ativação (por exemplo, a função degrau) decide se o neurônio "dispara" (1) ou não (0).

A rede Perceptron de uma camada é um dos modelos mais simples de redes neurais artificiais, criado para tarefas de classificação binária. Consiste em uma única camada de neurônios, onde cada entrada é multiplicada por um peso, somada e passada por uma função de ativação (geralmente uma função degrau) para produzir uma saída binária (0 ou 1). O Perceptron é capaz de aprender ajustando seus pesos durante o treinamento, utilizando um algoritmo de aprendizado supervisionado, porém essa rede é limitada para resolução de problemas linearmente separáveis.

A rede neural densa, também conhecida como rede neural multicamada, é um tipo de arquitetura de rede neural artificial onde cada neurônio em uma camada está conectado a todos os neurônios da camada seguinte. Essa estrutura permite que a rede capture relações complexas entre os dados, sendo amplamente utilizada em tarefas de classificação, regressão e reconhecimento de padrões. Cada conexão entre os neurônios possui um peso, que é ajustado durante o treinamento.

O cálculo de erro mais simples para uma rede Perceptron é baseado na diferença entre a saída esperada (label) e a saída predita pelo modelo. Esse erro é utilizado para ajustar os pesos e o bias durante o treinamento. utilizando a fórmula:

$$error = label - prediction$$

O gradiente é um vetor que aponta na direção de maior crescimento de uma função. No contexto de redes neurais é utilizado o gradiente descendente, uma ferramenta matemática utilizada com o objetivo de minimizar os valores da função de

custo que calcula o erro. É calculado utilizando derivadas parciais da função de custo em relação aos pesos e ao bias. O gradiente indica a direção e a magnitude da mudança necessária nos pesos para reduzir o erro.

Para alcançar esse objetivo, primeiro é calculado o parâmetro delta

Esse parâmetro será utilizado pela rede para calcular a atualização dos pesos entre cada neurônio. O delta possui 2 fórmulas, separando o cálculo em partes, o delta de saída e o delta oculto, pertencendo às respectivas camadas de saída e oculta.

Para o cálculo do valor $delta_{saída}$:

$$delta_{saída} = error \cdot \frac{d}{dx}(f(x))$$

Para o cálculo do valor $delta_{oculta}$:

$$delta_{oculta} = peso \cdot delta_{saída} \cdot \frac{d}{dx}(f(x))$$

Onde $f(x)$ é a função de ativação utilizada pela rede.

Calculados os valores de delta para cada um dos neurônios, a rede segue para a aplicação do algoritmo de *backpropagation* para realizar o cálculo de ajuste dos pesos. Primeira o valor de $delta_{saída}$ é encontrado e então o algoritmo realiza o cálculo de ajuste dos valores da direita para a esquerda, ou seja, da camada de saída para a camada oculta, atualizando os pesos de cada uma dessas conexões. Após essa

etapa, os valores de delta da camada oculta são calculados e então o algoritmo realiza a atualização dos pesos da esquerda para a direita, entre a camada de entrada e a camada oculta.

Com os pesos atualizados, a rede neural procede com o treinamento pelo número de épocas que foi programada pelo desenvolvedor. Cada época de treinamento da rede neural é uma execução do algoritmo de *backpropagation* que é realizado, calculando os parâmetros necessários e atualizando os pesos das conexões a cada iteração.

O bias é um termo adicional, uma constante, adicionado à rede neural para gerar uma melhor generalização do modelo. Ele permite que o modelo "ajuste" a função de ativação para melhor se adequar aos dados. Em outras palavras, o bias é um parâmetro que ajuda o modelo a aprender padrões mais complexos, mesmo quando os dados não estão centrados em torno de zero.

Batch Gradient Descent é um método de otimização usado no treinamento de modelos de *machine learning*, onde o gradiente da função de custo é calculado utilizando todo o conjunto de dados de treinamento em cada iteração. Isso garante uma direção de atualização dos parâmetros mais precisa, mas pode ser computacionalmente caro, especialmente com grandes conjuntos de dados, pois requer o processamento de todos os dados antes de cada atualização.

Stochastic Gradient Descent (SGD) é uma abordagem onde o gradiente da função de custo é calculado e os parâmetros são atualizados para cada item do conjunto de dados. Isso torna o processo muito mais rápido e eficiente em termos computacionais, especialmente para grandes conjuntos de dados. No entanto, as atualizações podem ser mais ruidosas e menos estáveis, o que pode levar a uma convergência mais lenta ou oscilações, sua grande vantagem é permitir escapar de mínimos locais com mais facilidade.

Funções de ativação usadas em redes neurais:

1. Step (Degrau): É uma função binária que retorna 0 se a entrada for menor que um limiar e 1 caso contrário. É simples, mas não é diferenciável, o que limita seu uso em redes neurais modernas.

2. Sigmoide: Mapeia a entrada para um valor entre 0 e 1. É amplamente usada em problemas de classificação binária, mas pode causar problemas como *vanishing gradient* em redes profundas. Que é o que ocorre quando os gradientes das funções de perda se tornam muito pequenos, fazendo com que a atualização dos pesos seja insignificante.

3. Tangente Hiperbólica (Tanh): Similar à sigmóide, porém mapeia a entrada para um valor entre -1 e 1.

4. ReLU (Rectified Linear Unit): Retorna a entrada se for positiva e 0 caso contrário. É simples, eficiente e resolve parcialmente o problema de *vanishing gradient*, sendo amplamente usada em redes neurais profundas.

5. Softmax: Usada principalmente na camada de saída para problemas de classificação multiclasse. Transforma as saídas em probabilidades, garantindo que a soma das saídas seja 1.

6. Linear: função linear retorna a entrada diretamente, sem nenhuma transformação. É representada por ***f(x)=x***. É usada em problemas de regressão ou em camadas intermediárias de redes neurais, mas não introduz não-linearidades, o que limita sua capacidade de modelar relações complexas sozinha.

De forma intercalada com as aulas teóricas, implementações de cada uma das funções foram feitas utilizando o Python. Bem como os algoritmos apresentados em aula foram postos em prática com exemplos usando *datasets* clássicos. Diversas abordagens de rede neural foram implementadas e minuciosamente detalhadas, desde classificação binária e multiclasse, até a regressão para uma ou mais variáveis de saída.

Conclusão

O card abordou de maneira prática implementações e aplicações das redes neurais, além de revisar boa parte do processamento de dados com pandas. O curso deixa evidente que é necessário não apenas saber utilizar os algoritmos certos para cada situação, mas é preciso ter uma visão crítica sobre cada conjunto de dados para realizar uma análise e um tratamento correto dos dados, e assim otimizar as entradas do modelo.

Referências

[1] card13 - Deep Learning com Python de A a Z - O Curso Completo.