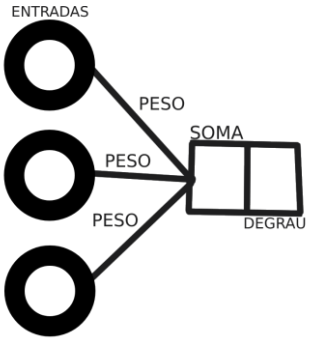
Redes neurais artificiais

Neurônio artificial

O princípio para "imitar" um real neurônio é analisar a possível forma com que ele se comporta, como representado no vídeo, podemos separar essas funções em entradas, pesos, soma e degrau.

Na qual suas entradas são as ações a serem tomadas, o peso é a importância de cada ação, a soma resultará do somatório de todas as ações tomadas corrigidas com o seu peso equivalente e o degrau será definido para 1 quando a soma for maior que zero e 0 caso contrário.



A rede neural não é criada com sua lógica implementada, para isso serve ao usuário que a está implementando corrigir e adaptar sua lógica ao redor do modelo.

Uma das formas explicadas pelo vídeo é supor um valor inicial do peso (mostrado no vídeo como zero) e conferir se a sua lógica está retornando o resultado desejado. Percebendo assim os pontos na qual está retornando uma resposta errada e corrigindo o mesmo.

Ele utiliza da função: peso(n + 1) = peso(n) + (taxaAprendizagem * entrada * erro) e após algumas iterações o modelo é capaz de prever quando a saída deve retornar 1.

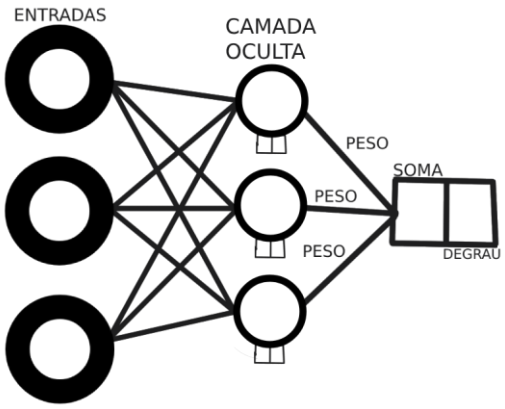
Para encontrarmos esses erros podemos utilizar uma comparação direta do resultado final com o resultado previsto e marcar como valor – previsao = resultado.

$$\begin{aligned} valorA &\rightarrow previsaoA = correto \\ valorB &\rightarrow previsaoB = correto \\ valorC &\rightarrow previsaoC = correto \\ valorD &\rightarrow previsaoD = correto \end{aligned}$$

Redes multicamadas

Partindo da explicação anterior, podemos observar que ela só é válida para problemas linearmente separáveis, fazendo que o contrário se torne impossível de se resolver.

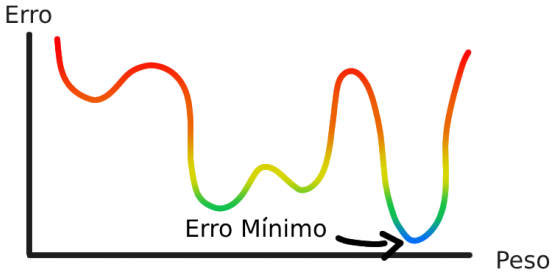
Para conseguir implementar a rede neural no problema que não é linearmente separável, precisamos colocar uma camada oculta intermediaria entre as entradas e a soma final.



A camada oculta será composta por uma soma e degrau parecido com o último passo, porém, ambos degraus serão feitos pela função sigmoide (y = 1/(1+e^(-x))).

Conforme esperado, a primeira iteração dos resultados será distante do resultado esperado, visto a declaração dos pesos de forma aleatória. Utilizando a média absoluta dos erros, treinamos a rede para se adaptar e melhorar os resultados até conseguir atingir uma média próxima de zero.

Para calcularmos esse erro e descobrirmos qual próximo passo tomar no valor do peso, utilizamos o princípio do gradiente, sendo o mesmo começar com um valor de erro elevado e ir "descendo" até chegar no valor de erro mínimo.



O ajuste dos pesos conforme o resultado inicial pode ser feito partindo da equação: peso n+1 = (peso n * momento) + (entrada * delta * taxa de aprendizagem), sendo a taxa de aprendizagem e o momento, variáveis responsáveis por acelerar o processo das iterações.

Para (entrada * delta), calculamos para cada neurônio e somamos todos resultados respectivos.

Considerando uma taxa de aprendizagem e momento, para cada peso calcularemos o resultado equivalente utilizando o cálculo anterior.

Considere por exemplo:

- Taxa de aprendizagem = 0.3
- Momento = 1
- Entrada * Delta do peso 1 = 0.032
- Peso 1 = -0.007

(-0.017 * 1) + 0.032 * 0.3 = -0.007

Sendo o novo valor do peso 1 equivalente à -0.007, repetindo o processo para cada neurônio.

Um possível problema encontrado é a falta de utilidade quando todas entradas são zero, dificultando assim o modelo de "aprender" com essa iteração, para contornar este problema, é utilizado uma unidade de bias, semelhante a uma entrada e camada oculta, adicionado para modificar o valor caso todas entradas sejam equivalentes a zero. Normalmente a bias é configurado automaticamente com a biblioteca utilizada.

Cálculo do erro

A forma apresentada anteriormente neste relatório é simples e com pouca acurácia, quando consideramos um projeto de maior escala devemos utilizar maneiras melhores de calcular este erro.

Duas maneiras melhores de calcular ele é utilizando do erro quadrático médio (MSE) e raiz quadrada do erro médio (RMSE).

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

MSE: Uma fórmula um pouco mais robusta para calcular o erro, o MSE coloca a diferença entre o resultado correto e o previsto ao quadrado antes de calcular a sua média.

Essa ação de elevar ao quadrado tem grande utilidade para "punir" quando um erro ocorrer, facilitando a visualização e agilizando o processo da rede neural funcional.

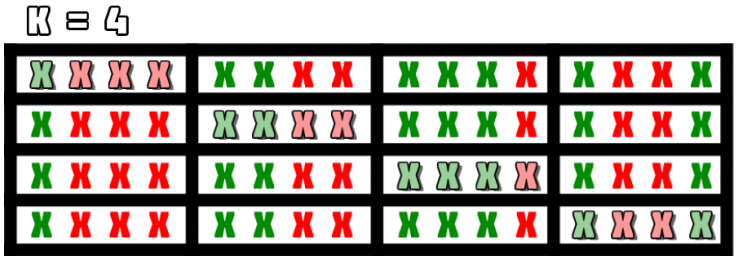
$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f_i - o_i)^2}$$

RMSE: Seu cálculo é parecido com o MSE, porém coloca o resultado em uma raiz quadrada, fazendo com que o erro seja ainda mais perceptível.

Avaliação de Algoritmos

Validação cruzada

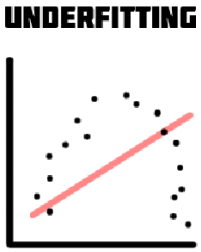
Também conhecido por K-fold Cross Validation, esse método de treinamento de dados consiste em determinar um valor K e dividir o seu banco de dados nessa mesma quantidade, fazendo K rodadas de treinamento/teste para no final medir o desempenho geral da aplicação.



Underfitting e Overfitting

Underfitting acontece quando o problema é subestimado, fazendo uma solução simples/pequena para um problema complexo/grande. Geralmente apresenta resultados ruins durante o treinamento.

Overfitting é o completo oposto, utilizar uma solução complexa/grande para um problema simples/pequeno. Geralmente apresenta resultados bons durante o treinamento e ruins durante o teste.



Atividades

Partindo do conteúdo apresentado e explicado, o vídeo utiliza de três exemplos para mostrar a eficiência e a facilidade de colocar uma rede neural em prática, partindo de diferentes níveis de dificuldade é mostrado como importar, tratar e utilizar os dados presentes em datasets pela internet.

É mostrado e explicado exemplos sobre problemas de classificação simples (breast cancer), classificação complexo (base iris), regressão simples (base de carros usados) e regressão complexo (base de vídeo games).

Durante a explicação dos códigos é apresentado diversas maneiras de melhorar a eficiência geral do código, utilizando de otimizadores, mudanças de parâmetros, tratamentos de dados ou aumento de repetições.

Atividade criada

Utilizando do conhecimento adquirido pelas aulas, fiz um projeto de pequena escala para prever valores de casas na índia, utilizando do banco de dados "House Price India.csv", na qual conta com diversos atributos que influenciam no preço final.

Com alguns tratamentos e períodos de testes, cheguei no erro de aproximadamente 24%. É um valor alto para erro e existe diversas maneiras de otimizar o código, por se tratar de um projeto simples para aplicar os conceitos não dediquei tanto tempo quanto deveria.

Conclusão

A aula explica o funcionamento básico das redes neurais artificiais e como imitam o comportamento dos neurônios. Discute redes multicamadas para resolver problemas complexos e a importância do cálculo de erros, utilizando métodos como MSE e RMSE. A validação cruzada é mencionada como uma técnica de avaliação, e os problemas de underfitting e overfitting são explorados.

Exemplos práticos mostram a aplicação de redes neurais em classificações e regressões. O projeto final, que prevê preços de casas na Índia, destaca a utilidade prática das redes neurais e a necessidade de otimização para melhorar a precisão das previsões. Em resumo, a aula demonstra a utilidade das redes neurais em resolver diversos problemas.