1 Perceptron

Perceptron é o modelo de rede neural mais simples em que só há uma unidade/neurônio. Não há uma camada oculta. Então, ele recebe os inputs da camada de entrada com os seus devidos pesos, faz uma média ponderada, executa uma função de ativação, como degrau ou sinal e o resultado disso é a saída. O neurônio utilizado é chamado de unidade lógica de limiar (TLU) *Mãos à Obra: Aprendizado de Máquina co Scikit-learn, Keras & TensorFlow*.

O cálculo é bem simples. É utilizado a média ponderada das entradas, junto com os seus pesos *Mastering machine learning algorithms: expert techniques to implement popular machine learning algorithms and fine-tune your models*

$$a_j = \sum_i w_{ji} x_i \tag{1}$$

onde x_i representar o i-ésimo dado de entrada e w_{ji} é o respectivo peso da conexão daquele dado de entrada com a j-ésima unidade.

Depois é aplicado uma função de ativação, $h(\cdot)$, em a_j,

$$y = h(a_j), \tag{2}$$

e y é o dado de saída.

2 Multilayer perceptron - MLP

É o mesmo modelo porém agora com mais camadas, chamadas de hidden layers. Cada unidade tem uma função de ativação que recebe os inputs da camada anterior até chegar a camada de saída. O sinal circula apenas em uma direção (das entradas às saídas). A arquitetura de MLP é o exemplo de um rede neural *feedforwardMãos à Obra: Aprendizado de Máquina co Scikit-learn, Keras & TensorFlow*.

Uma rede neural calcula a média ponderada das entradas da seguinte forma

$$a_j = \sum_i w_{ji} z_i \tag{3}$$

onde

$$z_i = h(a_i)$$

é a transformação de a_i pela função de ativação $h(\cdot)$.

3 Métodos de otimização: gradiente descendente e estocástico

3.1 gradiente descendente

O método gradiente descendente é um método de otimização. É basicamente o método de Newton-Raphson. É dado por

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \mathcal{H}^{(\tau)^{-1}} \nabla C(\theta^{(\tau)}). \tag{4}$$

onde

- θ é o parâmetro a ser otimizado;
- $\mathcal{H}^{-1}$ é o inverso da hessiana;
- $\nabla C(\theta)$ é o gradiente da função de custo;

Porém calcular a matriz hessiana é muito cara computacionalmente. Então, é aconselhável substituir por η que é chamado de taxa de aprendizado. η tem de ser menor do que o maior autovalor de $\mathcal{H}^{-1}$.

$$\eta = \frac{1}{\lambda_{\max}}$$

Logo, a expressão fica

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \eta \nabla C(\theta^{(\tau)}) \tag{5}$$

Este método usa todo o conjunto de dados para o treinamento.

3.2 Gradiente descendente estocástico

O método de gradiente descendente estocástico utiliza uma parcela de dados em cada época para otimizar o parâmetro θ.

A ideia por trás do gradiente descendente estocástico é que a função de custo pode ser escrita como

$$C(\theta) = \sum_{i=1}^{n} c_i(x_i, \theta). \tag{6}$$

c_i é o valor da função de custo naquele conjunto de dado x_i. c_i também é chamado de função de perda e a sua soma é a função de custo $C(\theta)$ *Mastering machine learning algorithms: expert techniques to implement popular machine learning algorithms and fine-tune your models*. O gradiente pode ser calculado como

$$\nabla C(\theta) = \sum_{i=1}^{n} \nabla c_i(x_i, \theta). \tag{7}$$

A estocasticidade é introduzida porque os x_i são escolhidos aleatoriamente do conjunto de dados para treinar θ. Este subconjunto formado por x_i é chamado de minibatch. A quantidade de minibatches é um hiperparâmetro. O algoritmo vai otimizar θ iterando por esses minibatches. Como ilustrado na equação 8

$$\theta^{\tau+1} = \theta^{\tau} - \eta \sum_{x_i \in B_k} \nabla c_i(x_i, \theta), \tag{8}$$

onde B_k é o k-ésimo minibatch. Um outro hiperparâmetro é epochs que é a quantidade de vezes em que vai se repetir o processo de formar os minibatches e calcular θ *Applied Data Analysis and Machine Learning*.

Uma ilustração do algoritmo seria:

- escolha uma quantidade de minibatches. Por exemplo, 32 minibatches. É preferível que a quantidade de minibatches seja potências de 2 por razões de otimização computacional *Applied Data Analysis and Machine Learning*;
- escolha aleatoriamente n/M dados para cada minibatch;

- itere pelos minibatches calculando a equação (8);
- repita o ciclo até percorrer o número de epochs escolhido;

4 Backpropagation

Backpropagation é um algoritmo para avaliar o gradiente da função de custo, $C(w)$, para uma *feedforward neural network*. A avaliação da função de custo é realizada por meio de esquema de passar mensagens ao qual informação é passada progressivamente e retroativamente pela rede neural *Pattern Recognition and Machine Learning*.

Queremos saber como que a função de custo varia com os pesos. Utilizando a função de perda. É fácil ver que a função de perda varia da seguinte forma

$$\frac{\partial c_n}{\partial w_{ji}} = \frac{\partial c_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{9}$$

onde

$$a_j = \sum_i w_{ji} z_i. \tag{10}$$

Logo, temos duas relações:

$$\frac{\partial c_n}{\partial a_j} = \delta_j, \tag{11}$$

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{12}$$

onde δ_j vai ser chamado de j-ésimo erro. A função de perda vai variar com os pesos da seguinte forma

$$\frac{\partial c_n}{\partial w_{ji}} = \delta_j z_i. \tag{13}$$

Para avaliarmos os δ para a camada oculta, nós utilizamos novamente a regra da cadeia

$$\delta_j = \frac{\partial c_n}{\partial a_j} = \sum_k \frac{\partial c_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{14}$$

onde

$$\delta_k = \frac{\partial c_n}{\partial a_k}. \tag{15}$$

Para avaliarmos $\frac{\partial a_k}{\partial a_j}$, usamos a eq. (3) e obtemos

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial z_j}{\partial a_j}\frac{\partial a_k}{\partial z_j}, \tag{16}$$

$$\frac{\partial a_k}{\partial a_j} = h'(a_j) w_{kj}. \tag{17}$$

Logo, a eq (14) fica *Pattern Recognition and Machine Learning*

$$\delta_j = h'(a_j) \sum_k w_{kj}\delta_k. \tag{18}$$

5 Regularização em Redes Neurais

Regularização da rede neural são práticas para evitar com que a rede faça *overfitting*. Então, a discussão da práticas de regularização de redes neurais gira em torno da arquitetura da rede já que as unidades de entrada e de saída são determinadas pela dimensionalidade do conjunto de dados. O que resta para otimizar é a arquitetura como, por exemplo, o número de camadas ocultas M ou como otimizar a função de custo usando Ridge (norma ℓ^2) ou Lasso (norma ℓ^1) *Pattern Recognition and Machine Learning*.

Referências

Géron, Aurélion. *Mãos à Obra: Aprendizado de Máquina co Scikit-learn, Keras & TensorFlow*. O'Reilly, 2021.

Bonaccorso, Giuseppe. *Mastering machine learning algorithms: expert techniques to implement popular machine learning algorithms and fine-tune your models*. Packt Publishing Ltd, 2018.

Applied Data Analysis and Machine Learning. 2021. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html.

Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer, 2006.