Universidade Federal de Uberlândia – UFU
Bacharelado em Sistemas de Informação – Monte carmelo
GSI556 Mineração de Dados – 2019/1
César Henrique Marçal Cardoso – 31711BSI006

Trabalho 2: Algoritmos para Mineração de Dados

1 – Introdução:

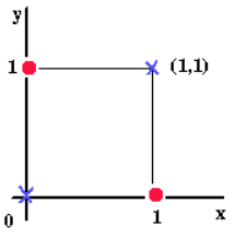
Neste trabalho foi desenvolvido um pequeno código na linguagem python 3.7 que utiliza a tabela da verdade de um XOR para aplicar os conceitos aprendidos em sala de aula e desenvolver um algoritmo que representasse uma rede neural MLP, Multilayer Perceptron (Perceptron de Multicamadas). Uma rede neural é formada por uma camada de entrada, 1 ou mais camadas escondidas e 1 camada de saída.

O objetivo deste trabalho é simular o funcionamento de uma rede neural que utiliza 3 camadas: 1 camada de entrada, 1 camada escondida e 1 camada de saída para resolver um problema não linear como a função XOR. Além disso é importante ressaltar que é necessário utilizar uma função de ativação, e que neste caso será utilizada a função sigmoid.

Quando colocamos os nossos objetos em um plano e conseguimos separar suas classes desenhando apenas uma única reta, dizemos que o problema é linear. Tal situação não representa o problema que estamos tratando, pois um XOR não pode ter seus objetos (A e B) separados entre 0 e 1 traçando apenas uma reta em seu plano. Portanto a rede neural realizará uma regressão para tentar separar esses objetos da melhor forma possível.

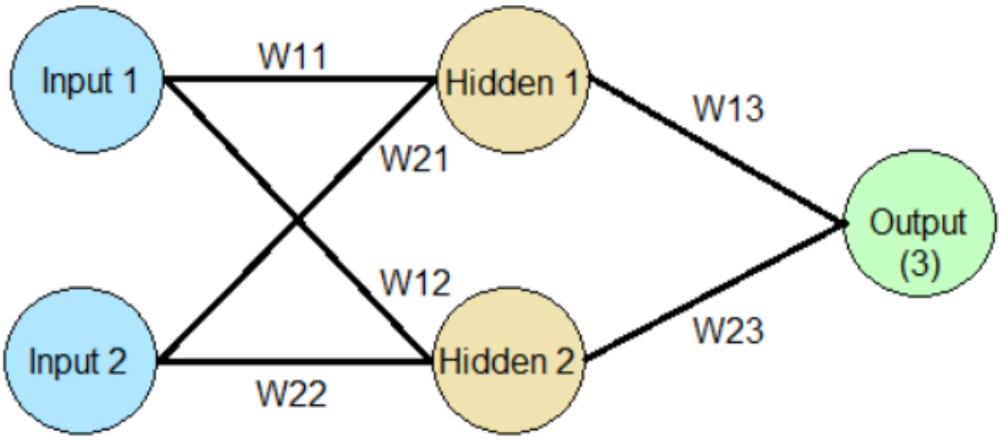
A base de dados utilizada é um XOR cuja tabela da verdade é a seguinte:

A	**B**	**A ⊕ B**
0	0	0
0	1	1
1	0	1
1	1	0



A função XOR também pode ser representada por: $(A \vee B) \wedge \neg(A \wedge B)$

2 – Desenvolvimento:



Na figura acima podemos ver a estrutura da rede neural, na qual os elementos de cor azul representam a camada de entrada (i), amarelo a escondida (j) e verde a de saída (k). Além disso, também é possível notar que temos 2 neurônios na camada J e 1 neurônio na camada K. Todo neurônio possui uma soma ponderada como entrada e propaga como saída uma função de ativação aplicada sob essa soma.

Como estamos representando um Multilayer Perceptron, todos os neurônios são conectados aos neurônios da camada anterior por meio de pesos e bias. Para atualizar esses pesos e bias vamos utilizar o algoritmo backpropagation que possui duas fases: a propagação e a retro propagação. Além disso esse algoritmo precisa de um coeficiente de aprendizado, um valor alpha que dirá o quanto a rede irá "aprender" em n iterações, que damos o nome de épocas, neste trabalho foram utilizadas dez mil épocas.

A primeira fase é quando aplicamos uma função de ativação sob o somatório das entradas que chegam no neurônio vezes o peso da ligação entre estes neurônios (vale ressaltar que devemos adicionar o bias nesse somatório). Em outras palavras a primeira fase (ou foward) é quando fazemos o processo descrito anteriormente até chegar no resultado da camada de saída, que será o valor resultante de todo o processo do foward para um objeto.

Na segunda fase precisamos ver o quanto o algoritmo "errou", subtraindo o resultado obtido do valor esperado (y – y2, sendo y o valor esperado e y2 o valor resultante do foward). Este valor encontrado nada mais é do que o reflexo dos pesos e dos bias no resultado da rede neural, e a cada iteração serão atualizados propagando esse erro obtido para as camadas anteriores.

Além disso, é necessário também compreender o comportamento da função de ativação utilizada, a sigmoid. O valor de y é calculado a partir da seguinte expressão: y = 1/(1 + e^(-x)), e esta função possui uma característica bem interessante, independente do valor que for atribuído a x, y sempre será um valor entre 0 e 1. Tal valor entre 0 e 1 representa a probabilidade entre as classes, na situação de um MLP esse valor retornado significa a probabilidade do objeto ser da classe 1, ou seja, se a sigmoid retornar um valor próximo a 1, representa que existe uma alta chance de termos classificado um objeto da classe 1, e caso o valor seja próximo a 0, representa uma baixa probabilidade de termos classificado um objeto da classe 1.

Foi definido os seguintes valores para alpha: 0,1 ; 1 ; 10; e ao executar o algoritmo para 10.000 épocas foi obtido os seguintes resultados, respectivamente:

```
Epoch:  10000
y2 [0 0] :  [0.07351323]
y2 [0 1] :  [0.9006124]
y2 [1 0] :  [0.90237161]
y2 [1 1] :  [0.10648994]
```

```
Epoch:  10000
y2 [0 0] :  [0.01072925]
y2 [0 1] :  [0.98881139]
y2 [1 0] :  [0.98874226]
y2 [1 1] :  [0.01380625]
```

```
Epoch:  10000
y2 [0 0] :  [0.00326717]
y2 [0 1] :  [0.9966712]
y2 [1 0] :  [0.99661715]
y2 [1 1] :  [0.00411337]
```

3 – Conclusão:

O comportamento da sigmoid é um fator crucial para a rede neural funcionar. A combinação entre o fato da rede neural realizar uma regressão (ou seja, ela retornará uma probabilidade do objeto que foi classificado pertencer a determinada classe), e a sigmoid retornar um valor compreendido entre 0 e 1 para qualquer valor de entrada, é um dos conceitos principais que tornam possível a implementação de um MLP. Esse comportamento da função sigmoid é de fundamental importância as redes neurais, por trazer a não linearidade ao sistema neural.

Através das conexões entre os neurônios de forma que a saída de um representa a entrada de outro, realizamos uma série de funções em cadeia, aplicando a sigmoid em toda saída e obtendo a cada final de uma iteração do backpropagation, um melhor arranjo de pesos a fim de aproximar cada vez mais a saída do foward com a saída esperada.

Além disso ao executar o programa para valores de alpha diferentes é possível perceber que o mesmo influência diretamente o resultado da rede neural. Quanto maior o valor de alpha mais "rápido a rede aprende", ou seja, ela precisa de menos interações do algoritmo backpropagation para que atualize os pesos com valores que dê bons resultados. Entretanto existe um limite para esse valor, na figura abaixo ilustra execuções para o valor de alpha igual a 35, 48 e 50, respectivamente:

```
Epoch:  10000
y2 [0 0] :  [0.00042255]
y2 [0 1] :  [0.99790151]
y2 [1 0] :  [0.99788901]
y2 [1 1] :  [0.99790394]
```

```
Epoch:  10000
y2 [0 0] :  [0.0001405]
y2 [0 1] :  [0.77874204]
y2 [1 0] :  [0.95632978]
y2 [1 1] :  [0.95991867]
```

```
Epoch:  10000
y2 [0 0] :  [0.99999905]
y2 [0 1] :  [0.99999905]
y2 [1 0] :  [0.99999905]
y2 [1 1] :  [0.99999896]
```

Ao analisar as figuras acima nota-se que na primeira o objeto 4, cuja saída esperada era zero, obteve um valor muito próximo a 1, ou seja, houve um erro quase de 100%, ao passo que na figura 2 além do problema do objeto 4 o objeto 2 alterou bastante seu resultado também, e por fim a figura 3 mostra todos os objetos com valores próximos de 1, errando completamente os objetos 1 e 4.

Este valor de alpha representa o que chamamos de gradient descent, que é um algoritmo de otimização com o objetivo de encontrar o valor mínimo de uma função. Neste caso quando menor o valor deste coeficiente de aprendizado, mais devagar vamos chegar ao valor mínimo, porém como os saltos são bem curtos isso garante que quando o algoritmo estiver bem próximo de encontrar esse valor, ele não passe muito desse ponto. Já na situação de um alpha muito grande, ele realizará saltos maiores, chegando mais rápido ao valor desejado porém quando o algoritmo passar desse valor, ele avançará uma quantidade consideravelmente maior.

Ao analisar todos os fatos apresentados, podemos concluir que o MLP implementado se mostrou bastante eficiente para classificar os objetos de uma base dados XOR por meio de uma regressão, conseguindo uma acurácia de 100%. Porém vale ressaltar que tal resultado é diretamente dependente do valor atribuído ao coeficiente de aprendizado (alpha), no geral para todos os valores maiores que 0 e menores que 34, ele obteve um bom desempenho.