Trabalho Prático 2 - Perceptron

Marcos Felipe Vendramini Carvalho - Ciência da Computação, Puc Minas - Brasil
marcos.vendramini@sga.pucminas.br

RESUMO

Uma das redes neurais mais conhecidas é o *perceptron*. Sua implementação mais simples, é capaz de solucionar problemas linearmente separáveis com apenas um neurônio. Esse trabalho mostra sua implementação para resolver operações lógicas *and* e *or* e mostrar que ele não é capaz de solucionar *xor*.

PALAVRAS-CHAVE

Perceptron, operações lógicas, aprendizado de máquina.

INTRODUÇÃO

O aprendizado de máquina tem como objetivo construir sistemas que são capazes de aprender através da experimentação. Os tipos de aprendizado são divididos em cinco grupos: supervisionado, onde possuem a saída esperada para realizar o treino, não supervisionado, que não possuem a saída esperada, semi supervisionado, que usa dados rotulados e não rotulados para o treino, reforço, que busca maximizar a recompensa final, e deep learning, que visa reconhecer padrões em situações complexas.

Os problemas supervisionados podem ser resolvidos usando diversas ferramentas como redes neurais, árvores de decisão, etc. Dentre as redes neurais existe o *perceptron* simples, rede de um neurônio que a partir das entradas recebidas dá uma saída binária verdadeira ou falsa baseada no treino recebido.

O objetivo deste trabalho é fazer uma implementação do *perceptron* de um neurônio capaz de resolver as operações lógicas *and* e *or*. Para isso, o relatório apresenta 3 seções: contextualização, implementação e resultados. Na contextualização será explicado o funcionamento do *perceptron* e os casos que ele soluciona. Na implementação será explicado o funcionamento do programa, mostrando detalhadamente suas principais funções. No resultado será mostrado os valores obtidos em cada teste, os parâmetros usados e a conclusão tirada na solução.

1 CONTEXTUALIZAÇÃO

1.1 Perceptron

O *perceptron* simples é uma rede neural com um neurônio capaz de solucionar problemas linearmente separáveis, através de um aprendizado supervisionado. Sua estrutura é compostas por “n” entradas, um bias (cujo valor é -1), seus respectivos pesos, e com eles, é gerado uma saída binária.

Para seu funcionamento ele possui duas etapas: treinamento e execução. Para isso é usado as seguintes variáveis: número de entrada (*m*), vetor de entrada (*x(m+1)*) (insere-se o bias na posição 0), vetor de peso (*w(m+1)*) (insere-se o peso do bias na posição 0), inicializado com valores reais randômicos entre 0 e 1, bias (*b*), que possui valor -1, resposta real (*y*), resposta desejada (*d*), erro (*e*), taxa de aprendizagem (*t*), número de ciclos (*ni*) e ciclos máximo (*n*).

O treinamento funciona da seguinte forma, para cada massa de treino, faz-se o somatório da multiplicação da entrada pelo seu peso (*x(m)*w(m)*). O valor resultante (*y*) é jogado em uma função de limiar que retorna 1 para entradas maiores que 0 e 0 para as demais (*f(y)*) e então é calculado o erro, subtraindo da resposta desejada a resposta encontrada (*d-f(y)*). Caso o erro seja diferente de 0 é calculado os novos pesos usando a fórmula: *w(m)=w(m)+(e*x(m)*t)* e então passa para o próximo treino. Quando todos os treinos passarem pelo processo acima verifica-se se algum deles não obteve a resposta correta, caso verdadeiro passa-se para o próximo ciclo onde todo o processo será refeito para toda a massa de teste. Para evitar loops infinitos, determina-se um número máximo de ciclo permitido, que quando atingido, interrompe o treino.

A execução do *perceptron* se dá após o treino ser concluído. Nela é feito o somatório de todas as entradas multiplicadas pelo seus pesos (incluindo o bias) (*x(m)*w(m)*) e calculando a função limiar na resposta encontrada (*f(y)*). O valor resultante é a resposta encontrada pela rede.

2 IMPLEMENTAÇÃO

2.1 Massa de treino

A massa de treino é um conjunto de entrada que já possui os valores esperados de saída. Ela é usada para treinar o *perceptron* de forma a alterar seus pesos caso o valor obtido pela entrada não seja igual ao valor esperado.

O algoritmo implementado para gerar a massa de treino é iniciado quando se chama o método *mtreino* que recebe como parâmetro número total de entradas. Nesse método é calculado a tabela verdade para "n" entradas. Para gerar essa tabela, cria-se duas repetições para preencher a matriz, preenchendo uma coluna por vez de acordo com o index da coluna, de forma que o número de

vezes que um true e false repete antes de trocar é igual a dois elevado ao index.

Tabela 1:

Index J	**0**	**1**
	false	**false**
	true	**false**
	false	**true**
	true	**true**
	2 elevado a 0 = 1	**2 elevado a 1 = 2**

Tabela 1:Exemplo da tabela gerada. No index 0, o valor altera quando ele aparece 1 vez, no index 1 quando ele aparece 2.

```
public static boolean[][] mtreino(int k) {
      int mp = (int) Math.pow(2, k);
      boolean mat[][] = new boolean[mp][k];
      int aux;
          for (int i = 0; i < k; i++) {
          int count = 0;
          boolean val = false;
         aux = (int) Math.pow(2, i);
          for (int j = 0; j < mp; j++) {
            if (i == 0) {
                mat[j][i] = val;
                val = !val;
          } else if (count < aux) {
                mat[j][i] = val;
                count++;
          } else {
                val = !val;
                mat[j][i] = val;
                count = 1;
          }
      }
   }
   return mat;
}
```

Ao finalizar o cálculo da matriz, o método à retorna.

2.2 Operações

Após calcular a massa de teste é calculado os valores esperados para ela para uma das três operações solicitadas, *and*, *or* e *xor*. As operações são chamadas pelos métodos, *andtreino*, *ortreino* e *xortreino*, respectivamente, e todas recebem como parâmetro a matriz de treino e o tamanho de entrada e retorna o vetor de resposta esperada.

O método *andtreino* percorre a matriz linha a linha fazendo a operação lógica and (&) com todos os valores da linha. Quando termina uma linha, salva o resultado em um vetor e passa para a próxima linha, até não ter mais entrada para teste.

```
public static boolean[] andtreino(boolean[][] k, int a) {
      boolean resp = true;
      int tam = (int) Math.pow(2, m);
      boolean[] vet = new boolean[tam];
      for (int i = 0; i < tam; i++) {
          for (int j = 0; j < a; j++) {
              resp = resp & k[i][j];
          }
          vet[i] = resp;
          resp = true;
      }
return vet;
}
```

Após calcular todos as respostas retorna um vetor com elas.

O método *ortreino* percorre a matriz linha a linha fazendo a operação lógica or (|) com todos os valores da linha. Quando termina uma linha, salva o resultado em um vetor e passa para a próxima linha, até não ter mais entrada para teste.

```
public static boolean[] ortreino(boolean[][] k, int a) {
      boolean resp = false;
      int tam = (int) Math.pow(2, m);
      boolean[] vet = new boolean[tam];
      for (int i = 0; i < tam; i++) {
          for (int j = 0; j < a; j++) {
             resp = resp | k[i][j];
          }
       vet[i] = resp;
       resp = false;
   }
   return vet;
}
```

Após calcular todos as respostas retorna um vetor com elas.

O método *xortreino* percorre a matriz linha a linha contando o número de 1 na linha. Quando termina uma linha, se a contagem der ímpar ele salva 1 no vetor resposta, caso contrário 0. Repete o processo para duas as linhas.

```
public static boolean[] xortreino(boolean[][] k, int a) {
     int tam = (int) Math.pow(2, m);
     boolean[] vet = new boolean[tam];
     int count=0;
     for (int i = 0; i < tam; i++) {
          for (int j = 0; j < a; j++) {
            if(k[i][j]) count++;
          }
       if(count%2==0)
         vet[i] = false;
       else
         vet[i] = true;
       count=0;
  }
  return vet;
}
```

2.3 Treino

O treino é feito a partir do método treino, que recebe como parâmetro a massa de teste e o vetor de resposta. Nele é calculado, até que todas as respostas sejam corretas ou execute 1000 ciclos, o resultado obtido fazendo o somatório da multiplicação da entrada pelo peso seguido da função de limiar sobre o resultado. Caso o valor esperado menos o resultado de diferente de zero, os pesos são recalculados (novo peso = peso + (erro * taxa de aprendizagem * entrada)) e o controlador de ciclo é trocado para houve erro (para executar outro ciclo), e então vai para a próxima entrada do treino. Quando todos os treinos são feitos, é verificado se o cálculo errou em alguma linha e se já tenha ocorrido 1000, caso tenha erro e não tenha ocorrido 1000 iterações, outro ciclo é iniciado e o processo recomeça. Vale destacar que o peso inicial para o treino é um valor aleatório entre 0 e 1

```
public static void treino(boolean[][] matriz, boolean[] corretas) {
        do {
        te = false; //ponto de parada do treino
        for (int i = 0; i < matriz.length; i++) {
        for (int j = 1; j < x.length; j++) {
                x[j] = matriz[i][j - 1] ? 1 : 0;
        }
        d = corretas[i];
          int aux = d ? 1 : 0;
        y = 0;
        for (int p = 0; p < x.length; p++) {
                y = y + (x[p] * w[p]);
         }
          if (y > 0)
                y = 1;
          else
                y = 0;
        e = aux - y;
          if (e == 0) {
        } else {
                te = true;
                for (int p = 0; p < x.length; p++) {
                w[p] = w[p] + (e * t * x[p]);
                }
        }
      }
      ni++;
  } while (te&&ni<n);
 }
}
```

Quando o método termina, significa que o *perceptron* está apto a decidir corretamente sobre qualquer entrada enviada (da operação determinada).

2.4 Teste

Quando o treino é concluído, chama-se o método rodar para executar o *perceptron* com a entrada recebida. Esse método recebe como parâmetro um vetor com as entradas escolhidas e com ele faz-se o somatório das entradas multiplicadas pelos seus respectivos pesos, passa pela função de limiar e então imprime a resposta na tela.

```
public static void rodar(double[] entrada) {
        double resp = 0;
        for (int i = 0; i < entrada.length; i++) {
            resp = resp + (entrada[i] * w[i + 1]);
        }
        resp = resp + x[0] * w[0];
        if (resp > 0)
                resp = 1;
        else
        resp = 0;
        System.out.println("resposta: " + resp);
}
```

3 RESULTADOS

3.1 Entrada

Para o programa funcionar espera-se uma entrada na estrutura (pode ter várias entrada no mesmo arquivo):
Número de entradas (inteiro)
Taxa de aprendizado (real)
Operação (String: *and,or,xor*)
Entradas separadas por linha (double: 0 ou 1)
Exemplo:

```
2
0,3
and
1
1
```

4.2 Resultados

Após a entrada do arquivo o *perceptron* realiza o treino e o teste conforme demonstrado acima e tem como saída os dados entrados, a resposta obtida e o número de ciclos do treino.
Exemplo:
Número de entrada: 2 Taxa: 0.3 Operação: and ciclos de treino: 7.0 entradas:

```
1.0
1.0
resposta: 1.0
```

Os testes feitos variaram entre 2 entradas à 9 e os resultados obtidos foram os seguintes:

3.2.1 And

Tabela 2:

Entrada	Resposta	Taxa	Ciclos
00	0	0.2	4

111	1	0.3	7
1101	0	0.2	18
01100	0	0.1	3
010000	0	0.1	20
0101110	0	0.1	100
00011101	0	0.1	8
000111010	0	0.1	81

Tabela 2: Resultados do teste do and.

Pode-se notar que a saída do *and* obtém a resposta esperada. Porém para taxas de aprendizagem maiores que 0.3 o resultado para mais de 4 entradas pode apresentar alguns erros.

3.2.2 Or

Tabela 3:

Entrada	Resposta	Taxa	Ciclos
10	1	0.3	2
110	1	0.3	1
0100	1	0.1	3
01100	1	0.3	3
010000	1	0.1	2
0001111	1	0.1	3
00011101	1	0.1	2
000111010	1	0.1	2

Tabela 3: Resultados do teste do or.

Pode-se notar que a saída do *or* obtém a resposta esperada. Pode-se destacar que o número de ciclos para o teste or tende a não ser muito grande.

3.2.2 Xor

Tabela 4:

Entrada	Resposta	Taxa	Ciclos
01	0	0.3	1000
001	1	0.3	1000
1010	1	0.1	1000
11000	0	0.2	1000
110001	0	0.3	1000
1101001	1	0.2	1000
11010011	0	0.2	1000
111010011	1	0.2	1000

Tabela 4: Resultados do teste do xor.

Pode-se notar que a saída do *xor* nem sempre obtém a resposta esperada. Pode-se destacar que o número de ciclos para o teste é sempre 1000 o que significa que ele nunca consegue achar um peso que resolva todo o problema. Essas informações mostram que o *xor* não é um problema linearmente separável, e por isso não acha um conjunto de peso que sempre resolva o problema.

REFERÊNCIAS

[1] Perceptrons Disponível em: <https://www.dca.ufrn.br/~lmarcos/courses/robotica/notes/perceptrons.pdf> Acesso em 25 mai. 2018.

[2] Ronaldo Prati. Aprendizagem por Reforço Disponível em: <http://professor.ufabc.edu.br/~ronaldo.prati/InteligenciaArtificial/reinforcement-learning.pdf> Acesso em 25 mai. 2018.