Identificando perfis de reprovação na disciplina de Lógica de Programação usando os Mapas Auto-Organizáveis de Kohonen (SOM)

Introdução

Este experimento foi desenvolvido por Vilson Rodrigues Câmara Neto. Um problema recorrente na Universidade Federal do Rio Grande do Norte (UFRN) é a dificuldade dos estudantes em compreender lógica de programação. Visto tal problemática, foi realizado este experimento com os estudantes da Escola de Ciências e Tecnologia (ECT) da UFRN, de forma tentar identificar padrões de reprovação de alunos que já cursaram a disciplina em anos anteriores e usaram a plataforma LoP. Foi realizado um tratamento dos dados para unir as tabelas do LoP com as submissões por semana e a sua situação final. Com essa base de dados foi escolhido o algoritmo de redes neurais, Mapas Auto-Organizáveis.

Proposto por Teuvo Kohonen em 1982, a rede neural artificial (RNA) de aprendizado não-supervisionado, SOM, busca diminuir a dimensionalidade de um grupo de dados mas mantendo as relações reais, e ao fim, um conjunto de multidimensões vira um conjunto bidimensional, adicionando valores de forma que os similares fiquem próximos. Diferente de outras RNA's que usam aprendizagem de correção de erros, a rede SOM utiliza aprendizagem competitiva. Os princípios do aprendizado competitivo são:

- Um conjunto de neurônios que são todos iguais, exceto por alguns pesos sinápticos distribuídos aleatoriamente e que, portanto, respondem diferentemente a um determinado conjunto de padrões de entrada;
- Um limite imposto à "força" de cada neurônio;
- Um mecanismo que permite aos neurônios competir pelo direito de responder a um determinado subconjunto de entradas, de modo que apenas um neurônio de saída (ou apenas um neurônio por grupo) esteja ativo (por exemplo, "ativado") de cada vez. O neurônio que vence a competição é chamado de neurônio "vencedor leva tudo" .

Os neurônios individuais da rede aprendem a se especializar em conjuntos de padrões semelhantes e, assim, tornam-se "detectores de recursos" para diferentes classes de padrões de entrada. Quando um exemplo é apresentado à rede, sua distância euclidiana para os vetores de peso é calculada. O neurônio no qual o vetor de peso é mais semelhante à entrada é chamado de melhor unidade correspondente ou vencedor. Os pesos do vencedor e dos neurônios próximos a ele são ajustados para o vetor de entrada. A taxa de aprendizagem da rede diminui ao passar das iterações/épocas. A fórmula de atualização dos pesos para um neurônio v com vetor de peso ***Wv(s)*** é:

$$W_v(s+1) = W_v(s) + \theta(u, v, s) \cdot \alpha(s) \cdot (D(t) - W_v(s))$$

Figura 1: Fórmula de atualização dos pesos dos Mapas Auto-Organizáveis

sendo ***s*** o índice das etapas, ***t*** o índice na amostra de treinamento, ***u*** é o índice do vencedor para a entrada ***D, α(s)*** é a taxa de aprendizado que diminui com o tempo e ***U(u,v,s)*** é a função de vizinhança que fornece a distância entre o neurônio ***u*** e o neurônio ***v*** na etapa ***s.***

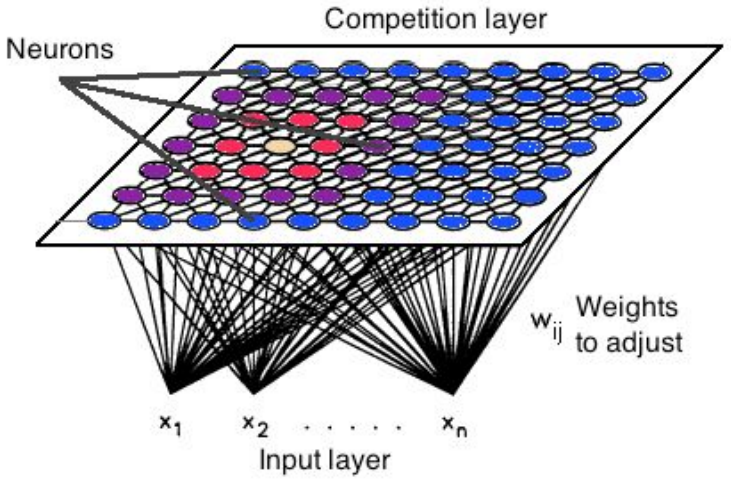

Figura 2: Representação do funcionamento dos Mapas Auto-Organizáveis

Metodologia

A base de dados contém as submissões de questões durante 18 semanas e a situação final dos alunos que cursaram a disciplina em semestres anteriores, ficou dividido como: 0 para reprovados e 1 para aprovados.

O motivo de utilizar as submissões é poder visualizar como as interação do aluno como a plataforma LoP influenciam no seu resultado final na disciplina.

Como é uma RNA de aprendizado não-supervisionado, então existe apenas o processo de treinamento. Foram utilizados 948 ocorrências, que corresponde a toda a base de dados, com os alunos que cursaram entre 2017.2 e 2019.1.

O objetivo é tentar visualizar através de um gráfico de pizza quais os neurônios apresentam mais porcentagem de aprovados ou reprovados. Assim, poderia mostrar ao estudante que ele está se dirigindo a situação de reprovação.

Códigos

```
[55] !pip install minisom

  Collecting minisom
    Downloading https://files.pythonhosted.org/packages/75/66/a5863c1a3bc5993
  Building wheels for collected packages: minisom
    Building wheel for minisom (setup.py) ... done
    Created wheel for minisom: filename=MiniSom-2.2.2-cp36-none-any.whl size=
    Stored in directory: /root/.cache/pip/wheels/4a/46/e1/605a71c70f6a19f16b4
  Successfully built minisom
  Installing collected packages: minisom
  Successfully installed minisom-2.2.2
```

Figura 3: Instalando o módulo minisom

```
[ ] tabela = table.iloc[:,[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,-1]]
    tabela.head(2)
```

	semana 1	semana 2	semana 3	semana 4	semana 5	semana 6	semana 7	semana 8	semana 9	semana 10	semana 11	semana 12	semana 13	semana 14	semana 15	semana 16	semana 17	semana 18	final
0	0.0	0.0	0.0	5.0	14.0	5.0	0.0	15.0	10.0	0.0	0.0	16.0	3.0	0.0	0.0	20.0	0.0	9.0	0
1	0.0	0.0	0.0	5.0	0.0	6.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

```
[ ] X_train = tabela.iloc[:, :-1].values

[ ] Y_train = tabela.iloc[:,-1].values
    Y_train
```


Figura 4: Selecionando o conjunto de teste e o conjunto de validação

```
[ ]  tamanhoXdaRede = 10;
     tamanhoYdaRede = 10;
     quantidadeCaracteristicas = 18

[ ]  from minisom import MiniSom

[ ]  som = MiniSom(x = tamanhoXdaRede, y = tamanhoYdaRede, input_len = quantidadeCaracteristicas, sigma = 1.0, learning_rate = 0.4)
```

Figura 5: Modelagem da rede SOM

```
som.train_random(data = X_train, num_iteration = 1000000)
```

Figura 6: Treinamento da rede

```
[ ]  # matriz de zeros para contador de reprovados
     MContRe = np.zeros((tamanhoXdaRede,tamanhoYdaRede))
     # matriz de zeros para contador de aprovados
     MContAp = np.zeros((tamanhoXdaRede,tamanhoYdaRede))
     # matriz de zeros para o número total de alunos
     MContT = np.zeros((tamanhoXdaRede,tamanhoYdaRede))

     cont = 0;

     for x in X_train:
       pos = som.winner(x)
       if (Y_train[cont] == 0): #Reprovado
         MContRe[pos] += 1
       if (Y_train[cont] == 1): #Aprovado
         MContAp[pos] += 1
       MContT[pos] += 1
       cont= cont+1
```

Figura 7: Criando 3 Dataframes

```
[158] cont = 1;
      for i in range(len(MContT)):
        for j in range(len(MContT)):
          plt.subplot(tamanhoXdaRede,tamanhoYdaRede,cont)
          cont=cont+1
          sizes = [MContAp[i][j],MContT[i][j]-MContAp[i][j]]
          plt.pie(sizes)
      plt.show()
```

Figura 8: Plotando o gráfico de pizza para exibição dos neurônios

Experimentos

Na figura 5 eu escolho o parâmetro Learning Rate (taxa de aprendizado) como 0.4 para evitar que a rede “esqueça” rapidamente das outras épocas.

Na figura 7 eu crio 3 matrizes de zeros, 1 que contém todas as ocorrências, aprovado e reprovado, 1 para aprovados e outro para reprovados, que são preenchidas conforme se encontra o 1 ou 0.

Na figura 8 eu preparo o plot de pizza, onde o número total de ocorrências menos o de aprovados vai colorir a pizza da região. Com isso eu consigo visualizar em que regiões que os alunos caem que estão mais sugestivas a reprovação.

Na figura 9 fica nítido que a região de reprovados (laranja) está concentrada da parte de central inferior. Na região superior direita é uma região de extremo sucesso, onde poucas pessoas que caíram nessa situação reprovaram.

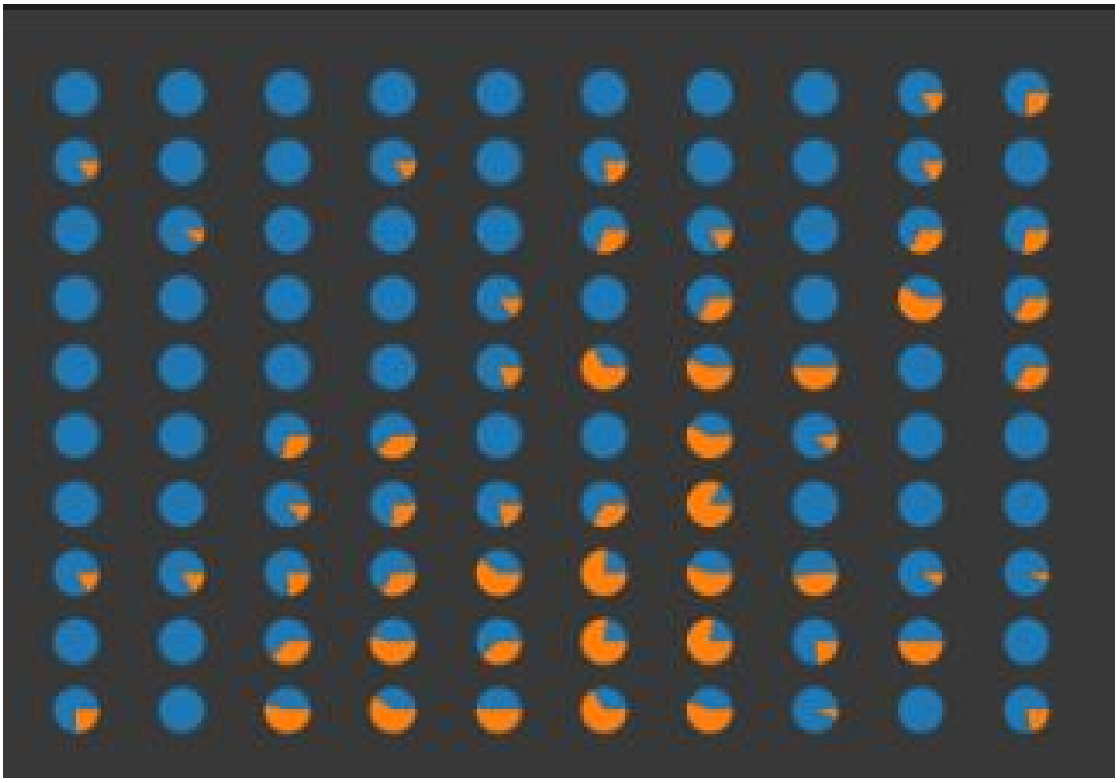

Figura 9: Plot de aprovados x reprovados

Com este resultado, dá para se intervir o quanto antes para evitar que os alunos entrem em região de aprovação. Também é de se destacar o trabalho realizado pela rede SOM, que conseguiu agrupar aprovados e reprovados, mesmo sem saber o seu alvo. Mostrando o quão eficiente é a rede neural.