Inteligência Computacional

Luís A. Alexandre

UBI

Ano lectivo 2023-24

Conteúdo

Aprendizagem não supervisionada com NNs: Introdução

- Nesta aula vamos estudar redes neuronais que aprendem de forma não supervisionada.
- A ideia desta abordagem é que a rede seja capaz de associar elementos que de alguma forma sejam semelhantes uns aos outros.
- Esta associação de vários elementos toma a forma normalmente de agrupamentos (clusters) no espaço de entrada (de características) do problema.
- Este tipo de algoritmos divide um conjunto com n pontos em m grupos ($m < n$) de tal forma que os elementos de um grupo são mais parecidos entre si do que com elementos de outros grupos.

Aprendizagem de Hebb

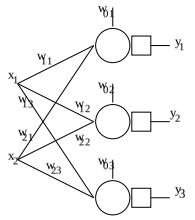
Aprendizagem de Hebb

- Esta é a regra mais antiga de aprendizagem não supervisionada em NNs.
- É baseada na hipótese proposta por Hebb (um neuropsicólogo) de que se um neurónio A excita outro neurónio B, então a ligação entre ambos é reforçada.
- Dito de outra forma, quando um neurónio dispara e leva outro que lhe está ligado a disparar também, o peso entre eles é reforçado (aumentado).

Aprendizagem de Hebb

- As redes que são treinadas com este algoritmo são percetrões (só possuem uma camada com neurónios).
- No que se apresenta de seguida $x_{i,p}$ representa a característica i do ponto (ou padrão) p.
- Quando não for necessário tornar explícita a dependência do ponto, escreveremos apenas x_i. O mesmo iremos fazer relativamente a $y_{i,p}$.
- Quando apenas quisermos referir o ponto p sem indicar uma das suas características usaremos $x_{,p}$.
- Usaremos ainda x para nos referirmos a um padrão de entrada genérico e y à saída produzida pela rede quando lhe é fornecido x.

Aprendizagem de Hebb



- Usando a notação da figura acima, a mudança nos pesos na iteração t, segundo a aprendizagem de Hebb, é dada por

$$\Delta w_{ik}(t) = \eta x_i(t) y_k(t)$$

onde o η é a taxa de aprendizagem.

- Os pesos são atualizados da forma comum

$$w_{ik}(t) = w_{ik}(t-1) + \Delta w_{ik}(t) \; . \tag{1}$$

Algoritmo para a aprendizagem de Hebb

1. Inicializar os pesos com valores iguais a zero.
2. Para cada padrão de entrada x, achar o correspondente valor de saída y.
3. Ajustar os pesos usando a expressão (1).
4. Parar quando as mudanças nos pesos sejam pequenas ou o número máximo de épocas tenha sido atingido. Senão voltar ao ponto 2.

Aprendizagem de Hebb: fator de esquecimento

- Um problema com a aprendizagem de Hebb é que, dada a forma da expressão da atualização dos pesos, ao repetirmos em cada época a apresentação dos mesmos à rede, existe a possibilidade de os mesmos atingirem valores arbitrariamente elevados.
- Uma possível solução para este problema é impor uma limitação ao aumento dos pesos.
- Uma destas limitações pode ser obtida com um fator de esquecimento, α, que retira parte do potencial aumento dos pesos em cada iteração:

$$\Delta w_{ik}(t) = \eta x_i(t) y_k(t) - \alpha y_k(t) w_{ik}(t-1)$$

onde α é uma constante positiva.

Aprendizagem de Hebb: abordagem de Sejnowski

- Uma outra possível solução para o problema do crescimento ilimitado dos pesos foi proposta por Sejnowski e consiste em fazer o crescimento dos pesos ser proporcional aos desvios relativamente à média das quantidades que aparecem na expressão original de Hebb.
- Assim, esta proposta fica:

$$\Delta w_{ik}(t) = \eta (x_i(t) - \bar{x}_i)(y_k(t) - \bar{y}_k)$$

onde

$$\bar{x}_i = \frac{1}{m} \sum_{p=1}^{m} x_{i,p}$$

e

$$\bar{y}_k = \frac{1}{m} \sum_{p=1}^{m} y_{k,p}$$

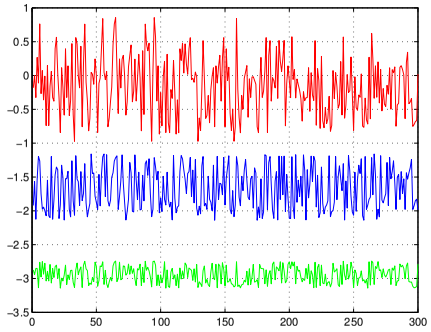
onde m é o número de pontos do conjunto de treino.

Aprendizagem de Hebb: exemplo

- Como a rede é treinada para reforçar as ligações vistas anteriormente, quando apresentamos um padrão conhecido, ou parecido com um conhecido, a sua saída vai assumir valores elevados.
- Quando os padrões apresentados na entrada forem diferentes dos conhecidos pela rede, as suas saídas irão assumir valores pequenos.
- Podemos tirar partido deste comportamento para construir um detetor de novidade: quando apresentamos padrões desconhecidos, eles podem ser vistos como sendo novidade para a rede.

Aprendizagem de Hebb: exemplo

- Exemplo (retirado de `http://blog.peltarion.com/2006/05/11/the-talented-dr-hebb-part-1-novelty-filtering`): Dados duma fábrica:

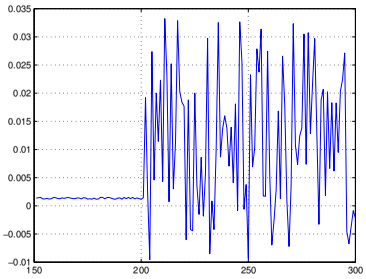


- Os padrões de 1 a 200 são normais e os de 201 a 300 são anormais.

Aprendizagem de Hebb: exemplo

- Queremos construir uma rede de Hebb que aprenda a detetar o comportamento anormal que surge nos dados a partir do padrão 200.
- Treinamos a rede com os padrões 1 a 150 (comportamento normal da fábrica) e vamos ver o que a rede produz quando recebe os padrões de 151 a 300.
- Como queremos que ela produza um valor alto na saída quando encontrar padrões diferentes dos que conhece, vamos usar um valor para eta negativo: treinamos usando aprendizagem anti-hebbiana.

Aprendizagem de Hebb: exemplo



- Note-se que os padrões de 151 a 200 são ainda normais, e só os de 201 a 300 são os anormais.

Aprendizagem por quantização vetorial

Aprendizagem por quantização vetorial

- Iremos estudar o algoritmo Learning Vector Quantization - I (LVQ-I).
- Cada neurónio da camada de saída representa um grupo (ou cluster).
- A construção dos grupos com a quantização vetorial é baseada na competição entre os grupos.
- Durante o treino, o grupo cujo vetor de peso seja mais próximo do ponto de entrada, ganha.
- Esta vitória faz com que o seu vetor de pesos e os dos seus vizinhos sejam ajustados de forma a se aproximarem mais ainda do ponto de entrada.
- A medida de proximidade entre um padrão de entrada $x_{,p}$ e o vetor de pesos associado ao neurónio k é normalmente a distância Euclidiana:

$$d_{p,k} = \sqrt{\sum_{i=1}^{n}(x_{i,p}(t) - w_{ik}(t-1))^2} \qquad (2)$$

Aprendizagem por quantização vetorial

- A atualização dos pesos é feita de acordo com

$$\Delta w_{ik}(t) = \begin{cases} \eta(t)(x_{ip}(t) - w_{ik}(t-1)) & \text{se} \quad k \in K_p(t) \\ 0 & \text{caso contrário} \end{cases} \qquad (3)$$

onde $\eta(t)$ é uma taxa de aprendizagem que vai diminuindo com a iteração t e $K_p(t)$ é o conjunto de vizinhos do neurónio vencedor (incluindo o próprio), para o ponto $x_{,p}$, na iteração t.

- Pode-se simplificar considerando que a vizinhança é constituída apenas pelo neurónio vencedor.

Algoritmo para a aprendizagem por quantização vetorial

1. Inicialização da rede:
   - 1.1 Escolher o número de grupos a obter (= n. de neurónios a usar).
   - 1.2 Os pesos podem ser inicializados com valores pequenos aleatoriamente; ou
   - 1.3 Usando os primeiros pontos de entrada para o efeito.
   - 1.4 Inicializar a taxa de aprendizagem e o raio da vizinhança.
2. Aprendizagem:
   - 2.1 Para cada ponto p fazer:
     - 2.1.1 Achar a distância Euclidiana $d_{p,k}$ entre o ponto de entrada $x_{,p}$ e os pesos w_k de cada neurónio da rede.
     - 2.1.2 Achar a saída para a qual a distância $d_{p,k}$ é menor.
     - 2.1.3 Atualizar todos os pesos da vizinhança $K_{p,k}$ usando a expressão (3)
   - 2.2 Atualizar a taxa de aprendizagem
   - 2.3 Reduzir o raio da vizinhança $K_{p,k}$
   - 2.4 Parar se as condições de paragem forem atingidas, senão voltar ao ponto 2.1.

Algoritmo para a aprendizagem por quantização vetorial

- As condições de paragem para o algoritmo anterior podem ser quaisquer das seguintes (ou suas combinações):
  - ter sido atingido o número máximo de épocas
  - parar quando os ajustes nos pesos sejam muito pequenos
  - parar quando o erro da quantização seja suficientemente pequeno
- O erro de quantização é dado por

$$EQ = \sum_{p=1}^{P} d_{p,k}^2$$

em que o neurónio k considerado é o que vence em cada apresentação de um ponto $x_{,p}$ à rede.

Fator de consciência

- Um problema que pode ocorrer na aprendizagem por quantização vetorial é um dos grupos dominar o problema vencendo com muita frequência.
- Para tentar evitar isto pode ser introduzido um fator chamado de consciência, que evita que um neurónio domine na decisão de vencedor.
- A forma de determinar as saídas da rede neste caso fica

$$y_{k,p} = \begin{cases} 1 & \text{para o } k \text{ que} \quad \min_{\forall k}\{d_{p,k} - b_k(t)\} \\ 0 & \text{caso contrário} \end{cases}$$

onde

$$b_k(t) = \gamma\left(\frac{1}{n} - g_k(t)\right)$$

e

$$g_k(t) = g_k(t-1) + \beta(y_{k,p} - g_k(t-1))$$

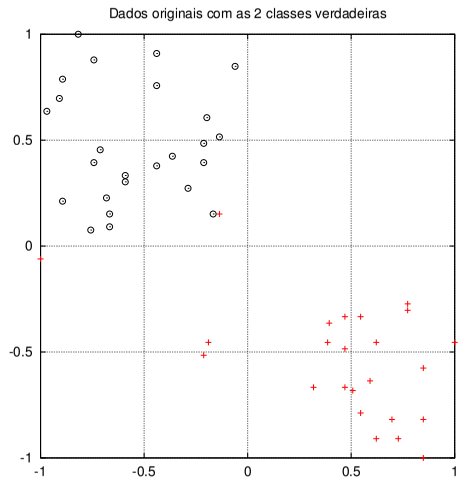
Fator de consciência

- n é o número de características e $g_k(0) = 0$.
- Inicialmente temos então $b_k(0) = \gamma/n$ o que dá a todas as saídas a mesma probabilidade de saírem vencedoras.
- $b_k(t)$ é o fator de consciência para cada saída k.
- Quanto mais vezes a saída k vence, maior se torna o termo $g_k(t)$ o que torna $b_k(t)$ mais negativo.
- Desta forma, um fator $|b_k(t)|$ vai sendo adicionado à distância $d_{p,k}$ tornando possível incluir como saídas vencedoras, saídas a maiores distâncias.
- Valores comuns: $\beta = 10^{-4}$ e $\gamma = 10$.

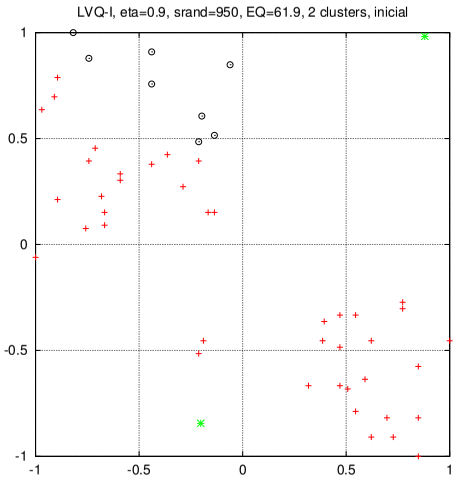
Exemplo

- Consideremos um exemplo para aprendizagem por quantização vetorial.
- Aqui temos as verdadeiras classes dos pontos para podermos comparar com os resultados. No entanto o algoritmo não as usa: é não supervisionado!
- O erro de quantização não é comparável entre problemas com número de clusters diferentes.
- Quais as arquiteturas de rede a usar para cada um dos exemplos que se seguem?

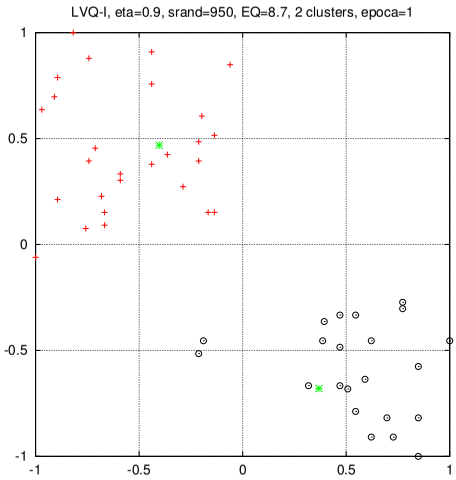
Dados originais com as classes verdadeiras



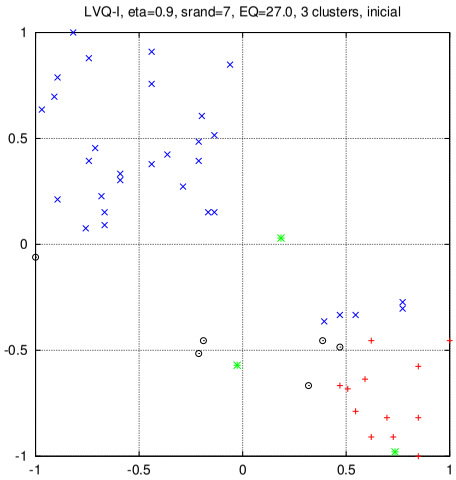
Exemplo 1



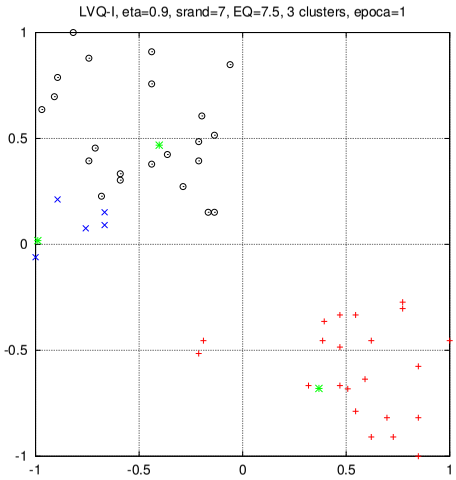
Exemplo 1



Exemplo 2



Exemplo 2



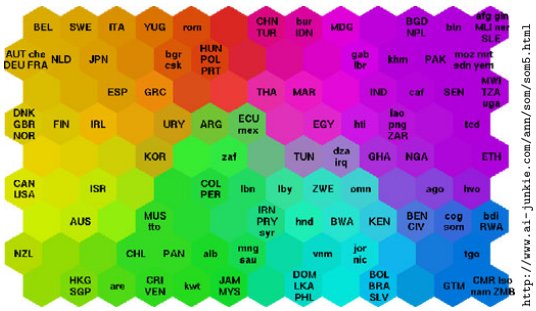
Self Organizing Maps (SOM)

SOM

- Uma rede do tipo SOM é uma rede não supervisionada em que todos os neurónios estão ligados a todas as entradas e além disso encontram-se organizados espacialmente numa malha (o "mapa").
- Estas redes foram inventadas por Kohonen em 1982. Desde então apareceram várias variantes.
- Conseguem adaptar-se aos dados de tal forma que dão uma visão topológica da forma como os dados se encontram organizados.
- A estrutura básica destas redes é normalmente uma malha bidimensional de neurónios que é ajustada ao longo da aprendizagem aos dados.

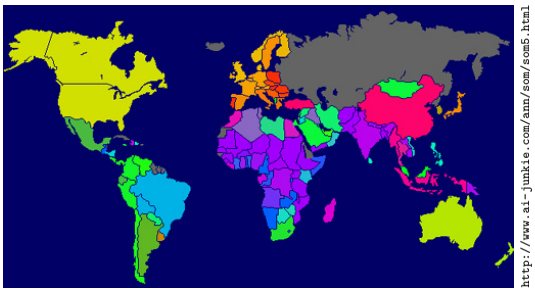
SOM: exemplo

- Um exemplo da aplicação da SOM é o que se mostra abaixo: a aplicação aos dados da riqueza de diversos países.



SOM: exemplo

- A informação em termos de cor pode ser reposta num mapa mundo para ficarmos com uma ideia mais clara da riqueza relativa das nações.



SOM: aprendizagem

- A aprendizagem nos SOM é baseada numa estratégia competitiva.
- Consideremos de novo os vetores de dados como sendo representados por $x_{,p}$, sendo a sua coordenada i dada por $x_{i,p}$, sendo p o índice do vetor no conjunto de dados.
- A dimensão do espaço de entrada onde se encontram estes vetores é m.
- O primeiro passo na aprendizagem é a definição da grelha de neurónios (o mapa) que é normalmente bidimensional e de forma retangular ou quadrada.
- O número de neurónios é menor que o número de vetores de dados.
- O vetor de pesos m-dimensional de cada neurónio do mapa é associado ao centro dum cluster.
- Duas formas de inicializar os vetores de pesos são:
  - de forma aleatória
  - usando os valores de pontos de dados escolhidos aleatoriamente

SOM: aprendizagem

- A forma de treino mais habitual é a estocástica: os pesos são ajustados após a exibição de cada ponto do conjunto de dados.
- O ajuste dos pesos é feito usando:

$$w_{kj}(t+1) = w_{kj}(t) + \eta(t) h_{qn,kj}(t)(x_{,p}(t) - w_{kj}(t))$$

onde qn representam a linha e a coluna do neurónio vencedor e $\eta(t)$ é a taxa de aprendizagem.

- O neurónio vencedor é aquele cujo vetor de pesos possui a menor distância ao vetor de entrada $x_{,p}$.
- A função $h_{qn,kj}$ é a chamada função de vizinhança e é usada para determinar quais os neurónios de coordenadas kj terão os seus pesos ajustados, quando o neurónio vencedor é o que se encontra na posição qn.

SOM: aprendizagem

- A função de vizinhança pode ter diferentes formas embora seja comum usar a que se segue:

$$h_{qn,kj}(t) = \exp\left(-\frac{||c_{qn} - c_{jk}||}{2\sigma(t)^2}\right)$$

onde c_{qn} e c_{jk} representam as coordenadas dos neurónios em termos do mapa e $\sigma(t)$ uma função que regula o tamanho da vizinhança a considerar.

- Normalmente faz-se decrescer $\sigma(t)$ com a evolução do algoritmo: quanto menor é o seu valor, menor é o tamanho da vizinhança a considerar. Pode-se usar a seguinte função:

$$\sigma(t) = \sigma(0)\exp(-t/\tau_1)$$

onde $\sigma(0)$ é um valor inicial elevado, e τ_1 um valor positivo pequeno.

SOM: aprendizagem

- Relativamente a $\eta(t)$ que é a taxa de aprendizagem, é também uma função decrescente ao longo do tempo. É frequente o uso da seguinte função (uma forma semelhante à do σ)

$$\eta(t) = \eta(0)\exp(-t/\tau_2)$$

onde $\eta(0)$ é um valor inicial elevado, e τ_2 um valor positivo pequeno.

- A aprendizagem termina quando não se verificam mudanças significativas nos pesos dos neurónios entre duas iterações consecutivas.

SOM versus LVQ

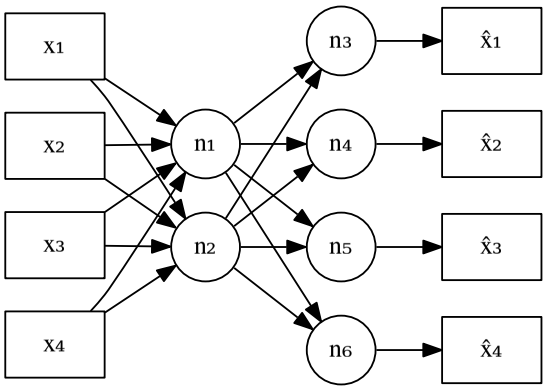
- Olhando com atenção para as expressão de atualização dos pesos vemos que a do LVQ e a do SOM são semelhantes.
- Em que diferem então estes algoritmos?
  - O SOM tem os neurónios organizados como uma malha o que permitem obter uma visão espacial dos dados;
  - O SOM mantém a estrutura topológica do espaço de entrada: 2 vetores de dados próximos no espaço de entrada, serão mapeados para neurónios próximos (ou para o mesmo) no espaço de saída;
  - O LVQ não muda a posição dos dados, escolhe apenas um cluster para cada ponto do espaço de entrada ao passo que o SOM faz corresponder a cada ponto do espaço de entrada uma posição (um neurónio) no espaço de saída.

Autoencoders

Autoencoders

- Consideremos o caso em que temos dados sem informação das classes, ou seja, apenas os vetores de entrada x_j sem os valores d_j respetivos.
- O autoencoder é uma rede não supervisionada que permite aprender usando a retropropagação, e que se assemelha a um MLP, com as diferenças referidas abaixo.
- Como é que podemos aprender usando a retropropagação se não temos valores desejados? Como achar o sinal de erro?
- O truque é usarmos como valores desejados os mesmos valores que são inseridos na entrada da rede.

Autoencoders: exemplo



Os retângulos representam valores numéricos e os círculos neurónios.

Autoencoders

- Para treinar a rede, são usados os valores de entrada como as etiquetas desejadas à saída, ou seja, $d_i = x_i$.
- Os valores $\hat{x}_i$ são as estimativas que a rede produz em relação às respetivas entradas x_i.
- Quanto melhor a rede estiver treinada, mais próximos ficam os valores $\hat{x}_i$ de x_i.
- Que sentido faz termos uma rede para aprender os dados de entrada?
  - A ideia é que ao aprender os dados de entrada e simultaneamente estar a ser alvo de uma redução no número de neurónios na camada escondida, estamos a forçar a rede a fazer **compressão de informação**.
- Qual o interesse ou o objetivo a alcançar com esta abordagem?
  - O AE pode ser usado para criar **novas representações dos dados** (tipicamente mais compactas) que podem depois ser usadas em vários contextos.

Autoencoders

- O autoencoder serviu de base para uma das abordagens que permitiu o treino eficiente de redes profundas.
- Tal foi conseguido com stacked autoencoders (autoencoders empilhados).
- Nesta abordagem cada autoencoder é treinado para aproximar as saídas do anterior.
- Isto permite construir uma rede arbitrariamente profunda fazendo sempre um treino local: só se treina um autoencoder de cada vez e esse treino é independente do treino de todos os outros.
- Por vezes, no final, afina-se o treino, fazendo um processo de backpropagation com todos os autoencoders empilhados, sendo que para isso se colocam um ou mais percetrões (um MLP) no final. Desta forma criamos um classificador para **problemas supervisionados**.
- Existe uma grande variedade de stacked autoencoders: denoising, contractive, sparse, etc.. (Para mais detalhes, ver por exemplo [1]).

Leitura recomendada

- Engelbrecht, sec. 4.1, 4.2, 4.4 e 4.5.1

[1] T. Amaral, J.M. Sá, L.M. Silva, Luís A. Alexandre, and J. Santos. Using different cost functions to train stacked auto-encoders. In *12th Mexican International Conference on Artificial Intelligence*, Mexico City, Mexico, November 2013. IEEE.