Inteligência Computacional

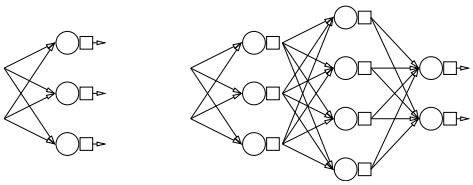
Luís A. Alexandre

UBI

Ano lectivo 2019-20

Conteúdo

Tipos de perceptrões

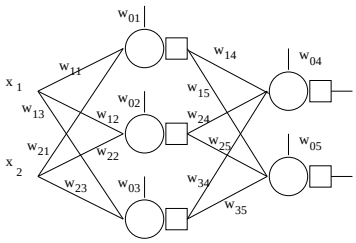


- Chama-se percetrão a um tipo de rede neuronal em que os neurónios estão organizados numa única camada.
- O percetrão multicamada (ou multi-layer perceptron: MLP) é um tipo de rede neuronal em que os neurónios estão organizados em mais de uma camada.

MLP

- As saídas dos neurónios duma camada são as entradas dos neurónios da camada seguinte.
- Isto implica que só existe circulação de informação num sentido na rede: é uma feedforward neural network (FFNN).
- Os MLPs foram desenvolvidos com o objetivo de permitir a aprendizagem de problemas não lineares.
- A sua utilização disparou no final da década de 1980 quando foi proposto um algoritmo que permitia efetuar a aprendizagem com este tipo de redes: a retropropagação do erro (error backpropagation).

Arquitectura do MLP



- O MLP da figura tem apenas uma camada escondida.
- Tem ainda uma camada de entrada (que não tem quaisquer neurónios) e uma camada de saída.
- O fator mais importante na capacidade de aprendizagem duma rede destas é o número de neurónios na camada escondida.
- O número de entradas e de saídas são intrínsecos ao problema a resolver.

Propriedades do MLP

- Para o grande interesse que o MLP despoletou contribuíram sem dúvida as suas propriedades de aproximador universal:
  1. FFNNs com funções de ativação contínuas e diferenciáveis e com uma camada escondida conseguem aproximar qualquer função contínua [4, 6]
  2. FFNNs com funções de ativação contínuas e diferenciáveis e com duas camadas escondidas conseguem aproximar qualquer função [7, 3]
- Embora duas camadas escondidas sejam suficientes, atualmente estudam-se redes com muitas mais camadas na área do deep learning [1] pois podem tornar a aprendizagem mais eficiente (ver mais abaixo).

Retropropagação do erro

- A descida do gradiente usada no neurónio simples evoluiu para retropropagação do erro no caso em que temos um MLP.
- A ideia é a mesma mas o truque está em perceber como fazer para atualizar os pesos das diferentes camadas da rede, visto que apenas a última possui informação direta do erro.
- O algoritmo de retropropagação do erro tem duas fases:
  - passagem para a frente: calcula os valores nas saídas da rede
  - passagem para trás: propaga o erro vindo da camada de saída até à camada de entrada e no processo atualiza os pesos da rede em função deste erro retropropagado

Tipos de aprendizagem com retropropagação

- A aprendizagem supervisionada com retropropagação nas NNs pode dividir-se em três tipos, de acordo com a frequência com que os pesos da rede são atualizados:
  - aprendizagem estocástica ou online: sempre que se mostra um ponto do conjunto de treino à rede, atualizam-se os pesos. Neste caso os pontos são mostrados aleatoriamente.
  - aprendizagem por lotes ou off-line: a atualização dos pesos só é feita quando todos os pontos do conjunto de treino foram mostrados à rede (uma época).
  - aprendizagem por mini-lotes: é um misto das anteriores, pois a atualização dos pesos é feita quando todos os pontos de um mini-lote forem mostrados à rede (o conjunto de treino é dividido em vários mini-lotes).

Tipos de aprendizagem com retropropagação

- A aprendizagem estocástica é mais interessante que a em lotes, embora para um dado número fixo de épocas de treino seja mais demorada.
- Uma vantagem da aprendizagem estocástica prende-se com o facto de ser capaz de escapar mais facilmente a mínimos locais: no treino em lotes o gradiente é em relação a toda a informação no conjunto de treino, enquanto que no estocástico se usam muitas direções diferentes (as relativas a cada ponto individual).
- A abordagem dos mini-lotes acaba por ser a mais usada pois aproxima os bons resultados da estocástica, evitando demorar demasiado tempo.

Retropropagação do erro

- Na descrição do algoritmo que se segue, vamos considerar o seguinte:
  - w_{ij} é o peso entre o neurónio i duma camada e o neurónio j da camada seguinte
  - $f(\cdot)$ é a função de ativação
  - s_j é a soma pesada das entradas do neurónio j
  - y_j é a saída do neurónio j
  - e_j é o erro no neurónio j
  - x é um ponto ou padrão de entrada na rede
  - $c(d(x), z(x))$ é a função de custo: diz como penalizar um erro dado uma saída $z(x)$ quando o que se pretendia era que a rede desse como saída o valor $d(x)$, quando se apresenta o ponto x nas entradas da rede
  - N é o número de padrões do conjunto de treino X
- A derivação detalhada é morosa podendo ser consultada no livro recomendado pag. 38 a 42 (com notação algo diferente da aqui apresentada).

Retropropagação do erro

Algoritmo da retropropagação do erro

1. **Inicialização dos pesos** Inicializar todos os pesos da rede com valores entre -0.1 e 0.1.
2. **Cálculo das saídas e dos erros** Para cada padrão do conjunto de treino, achar os seguintes valores:

$$s_j = w_{0j} + \sum_i w_{ij} x_i \qquad \text{primeira camada} \qquad (1)$$

$$s_j = w_{0j} + \sum_{i \in \text{camada anterior}} w_{ij} y_i \qquad \text{restantes camadas} \qquad (2)$$

$$y_j = f(s_j) \qquad (3)$$

$$e_j = f'(s_j) \frac{\partial c}{\partial y_j} \qquad \text{camada de saída} \qquad (4)$$

$$e_j = f'(s_j) \sum_{p \in \text{camada seguinte}} w_{jp} e_p \qquad \text{restantes camadas} \qquad (5)$$

Retropropagação do erro

3. **Atualização dos pesos** Atualizar os pesos de acordo com as seguintes expressões:

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \qquad (6)$$

$$\Delta w_{ij} = -\frac{\eta}{N} \sum_{x_i \in X} x_i e_j \qquad \text{primeira camada} \qquad (7)$$

$$\Delta w_{ij} = -\frac{\eta}{N} \sum_{y_i \in \text{camada anterior}} y_i e_j \qquad \text{restantes camadas} \qquad (8)$$

4. **Verificar condições de paragem** Se as condições de paragem não forem verificadas, voltar ao ponto 2.

Retropropagação do erro: condições de paragem

- Existem várias possibilidades a considerar em termos de condições de paragem da aprendizagem.
- A mais simples consiste em impor um número fixo de épocas.
- Outra mantém o algoritmo em funcionamento enquanto não for atingido um valor pré-definido de erro da função de custo (loss).
- Pode usar-se o erro de classificação no conjunto de treino, como critério de paragem.
- Veremos mais à frente que se deve usar como critério de paragem o erro num conjunto de validação.
- Finalmente, muitas vezes o que se faz é usar uma combinação das várias abordagens referidas, por exemplo, treinar até se obter um dado valor de erro ou então até se atingir um número máximo de épocas.

Retropropagação do erro

- O algoritmo que acabámos de ver é para o treino em lotes: a atualização dos pesos só é feita após todos os pontos do conjunto de treino terem sido mostrados à rede e ter sido calculado o erro associado a cada um.
- Para o treino estocástico, (atualização dos pesos após o processamento de cada ponto do conjunto de treino) é necessário alterar as expressões (7) e (8), bastando retirar o somatório e a divisão por N e passar a realizar a atualização dentro do ciclo do ponto 2, para cada padrão, em vez de apenas uma vez após o fim do ciclo.
- E para o treino com mini-lotes?
- De notar que no algoritmo descrito não foi especificada a função de custo $c(\cdot,\cdot)$ nem a função de ativação. Assim, para a sua utilização há que escolher uma função de custo e uma função de ativação e achar os termos $\frac{\partial c}{\partial y_j}$ e f', respetivamente.

Retropropagação do erro

- Para termos o algoritmo da retropropagação do erro completamente definido temos de escolher as funções f e c.
- Vejamos duas das escolhas mais comuns.
- Vamos escolher para f a função sigmóide. Desta forma, $f'(x) = f(x).(1 - f(x))$, como já vimos quando estudámos o neurónio simples e a descida do gradiente (para $\lambda = 1$).
- Uma escolha comum para a função de custo é o **erro quadrático médio** (MSE - mean squared error), onde m é o número de saídas da rede (usada para regressão):

$$c(d(x), y(x)) = \frac{1}{m}\sum_{i=1}^{m}(d_i - y_i)^2$$

- Logo,

$$\frac{\partial c}{\partial y_j} = -\frac{2}{m}(d_j - y_j)$$

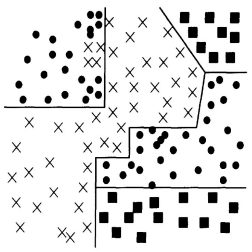
O que é a classificação

- A classificação é um tipo de problemas em que pretendemos fazer corresponder a cada ponto de entrada uma etiqueta chamada classe.
- O papel da rede neste caso é o de aprender, a partir do conjunto de treino com exemplos de vários pontos de cada classe, a distinguir os pontos de cada classe para que se possa classificar corretamente pontos não pertencentes ao conjunto de treino (generalizar).
- Muitos problemas do dia-a-dia podem ser vistos como problemas de classificação:
  - ao olharmos pela janela de manhã, classificamos o dia como bom, mau, chuvoso, etc.;
  - ao cruzarmo-nos com pessoas na rua tentamos reconhecê-las: colocá-las na classe dos conhecidos ou dos desconhecidos;
  - podemos também fazer outro tipo de classificação das pessoas com quem nos cruzamos: homens/mulheres, crianças/adultos, bonitos/feios, altos/baixos, etc.

Como usar um MLP na classificação

- A ideia é que o MLP crie fronteiras de decisão entre as diferentes classes do problema.
- A cada linha nas fronteiras de decisão corresponde um neurónio da camada escondida.

- Quantos neurónios são precisos na camada escondida para implementar as fronteiras da figura ao lado?
- Quantos seriam precisos para classificar corretamente todos os pontos da figura?

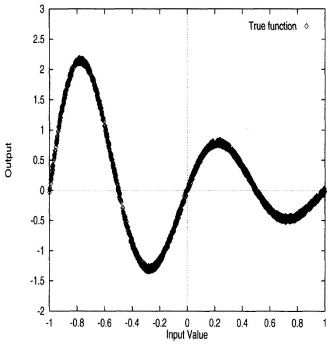


(fig.: Engelbrecht, p.51)

O que é a regressão

- A regressão é um tipo de problema em que se efetua aproximação de funções: dado um ponto na entrada da rede pretendemos que a saída seja a mais próxima possível dum valor dado por uma função.
- A função a aproximar é muitas vezes desconhecida em termos analíticos, tudo o que se possui são valores de amostras em determinados pontos.
- Os neurónios da camada escondida neste caso servem para lidar com os pontos de inflexão da função a aproximar: são precisos tantos quantos os pontos de inflexão mais um.

Como usar um MLP na regressão



(fig.: Engelbrecht, p.51)

- Quantos neurónios na camada escondida são necessários para o exemplo acima?
- Que arquitetura de rede deveria ser usada?

Regularização

- A **regularização** é uma alteração feita à função de custo com o objetivo de limitar a gama de soluções que se podem obter com o modelo.
- Esta limitação, vai no sentido de simplificar o espaço de pesquisa e conseguirmos obter uma solução rapidamente e mais adequada ao problema concreto que estamos a resolver.
- Por vezes estas alterações são feitas de forma a que seja incluído conhecimento a priori no treino do modelo.

Decaimento dos pesos

- O tipo de regularização mais simples é o chamado **decaimento dos pesos** (**weight decay**) que consiste em modificar a função de custo de forma a penalizar valores de peso elevados, p.ex.:

$$E = MSE + \lambda \mathbf{w}^T \mathbf{w} = MSE + \lambda \sum_i w_i^2 \tag{9}$$

  onde λ é uma constante positiva que controla a importância dada ao termo de regularização e no vetor $\mathbf{w}$ colocámos todos os pesos do modelo.
- Ao treinarmos a nossa rede com esta função de custo estamos a exigir que a rede aproxime os dados (através do termo MSE) e ao mesmo tempo que os pesos tenham valores pequenos, através do termo de decaimento.
- Este regularizador representa na realidade a norma L_2 do vetor de pesos e é chamado também de regularizador de Tikhonov.

Decaimento dos pesos

- Vejamos o efeito do decaimento dos pesos sobre a atualização dos pesos.
- Quando usamos a descida do gradiente para atualizar os pesos, fazemos:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E \tag{10}$$

  onde η é a taxa de aprendizagem (learning rate) e E representa a função de custo (sem regularização).
- Quando introduzimos o decaimento dos pesos, a expressão passa a ser

$$\mathbf{w} \leftarrow \mathbf{w} - \eta (\nabla_{\mathbf{w}} E + \lambda \mathbf{w}) = \tag{11}$$
$$\mathbf{w} \leftarrow (1 - \eta\lambda)\mathbf{w} - \eta \nabla_{\mathbf{w}} E \tag{12}$$

- Vemos que aparece um termo que faz com que se dê menos importância ao valor anterior dos pesos, em cada iteração.

Norma L_1

- Um outro regularizador comum é a norma L_1 dos pesos

$$E = MSE + \lambda \sum_i |w_i| \tag{13}$$

- O uso deste regularizador faz com que os pesos sejam **esparsos**, ou seja, muitos tenham valores zero e poucos sejam os diferentes de zero.
- Isto pode ser visto como uma forma de escolha de determinadas características (feature selection), pois algumas entradas ficam associadas a um peso zero e são ignoradas.

Conjuntos de redes

- Conjuntos (ou ensembles) de redes podem melhorar os resultados.
- Um conjunto é normalmente composto por várias redes idênticas, com exceção de serem inicializadas com pesos diferentes e treinadas independentemente umas das outras.
- A ideia é que estas redes, embora muito semelhantes, irão produzir resultados diferentes mesmo tendo sido treinadas e testadas nos mesmos dados, pois os pesos que usam são distintos umas das outras.
- Assim, pretendemos beneficiar do desempenho de cada uma destas redes, e para tal vamos combinar as saídas das redes de acordo com determinadas abordagens.

Conjuntos de redes: combinar saídas

- Consideremos que temos n redes treinadas para resolverem o mesmo problema.
- Se tivermos um problema de classificação, com m classes, podemos considerar que a saída do ensemble corresponde, por exemplo, à classe mais votada pelas n redes.
- Se tivermos um problema de regressão, podemos considerar que a saída do ensemble corresponde:
  - usar a média das saídas: $y_m = \frac{1}{m} \sum_{i=1}^{n} y_i$;
  - usar uma média pesada das saídas de cada rede: $y_{mp} = \sum_{i=1}^{n} w_i y_i$, onde w_i é o peso a atribuir à saída da rede i;
- Finalmente, para ambos os tipos de problema, podemos sempre usar apenas a rede que apresentou melhores resultados, ignorando as restantes.

Conjuntos de redes

- Apenas faz sentido combinar a informação de várias redes se elas discordarem umas das outras.
- Outra forma de construir um conjunto de redes é treinar cada rede em diferentes versões dos dados.
- Uma forma de obter diferentes versões dos dados é simplesmente partir o conjunto em sub-conjuntos diferentes.
- Uma outra forma, chamada bagging [2], consiste em treinar cada rede com um conjunto gerado por amostragem com repetição do conjunto de treino original, mantendo nestes novos conjuntos o mesmo número de pontos do conjunto de treino original.
- O resultado das redes é depois agrupado usando votação (para a classificação) ou faz-se uma média das saídas (para a regressão).

Conjuntos de redes

- Outras formas de obter redes diferentes passam por alterar qualquer elemento constituinte da rede: mudar a topologia; mudar o algoritmo de treino; mudar a taxa de aprendizagem; mudar as funções de ativação; etc.
- Finalmente, uma outra forma de obter melhores resultados passa por fazer as redes trocarem informação durante o treino, em vez de serem treinadas de forma independente.

Boosting

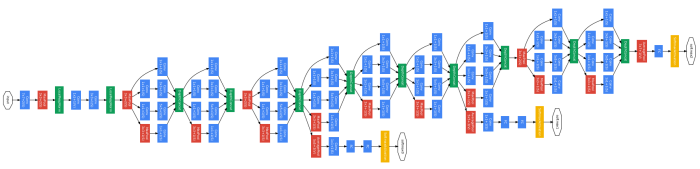
- Um método que usa troca de informação entre as redes durante o treino é o boosting:
  - as redes são treinadas sequencialmente;
  - as primeiras tratam os padrões mais fáceis e deixam os mais difíceis para as seguintes;
  - a estes padrões mais difíceis é associado um custo de erro superior, ficando estas redes mais aptas a lidarem com os padrões mais difíceis.
- Para mais detalhes, consultar, p.ex., `https://en.wikipedia.org/wiki/AdaBoost`.

Redes Profundas

- As redes profundas (que levam ao deep learning) são redes que contêm muitas camadas, tipicamente dezenas, embora existam redes com mais de 1000 camadas.
- Primeiro recordamos que é suficiente uma rede com 2 camadas escondidas para termos um aproximador universal. Então porquê usar mais camadas?
  - Existiam evidências empíricas que mostravam que certo tipo de redes com várias camadas, tinha resultados muito bons a processar imagens (redes de convolução, ver abaixo).
  - O próprio cérebro humano, no seu cortex, exibe cerca de 6 camadas;
  - Mais recentemente [5] ficou provado que existem vantagens computacionais em usar muitas camadas.

Redes Profundas: exemplo

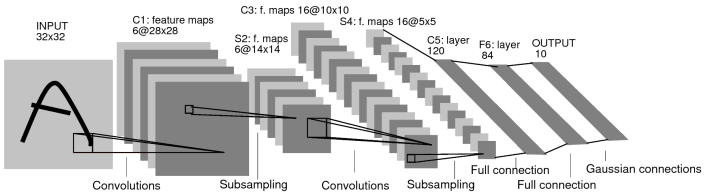
Googlenet [10]:



Redes de Convolução: CNNs

- As redes profundas mais comuns são provavelmente as redes de convolução (convolutional neural networks, ou CNNs).
- Estas redes têm a sua origem nos trabalhos do Fukushima na década de 1970, com o Neocognitrão.
- A ideia original era a de simular o funcionamento do sistema visual humano.

LeNet5 [8]:



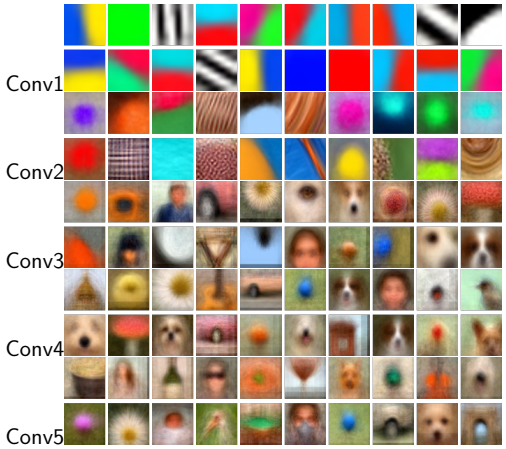
Redes de Convolução: CNNs

- São redes compostas por uma primeira parte em que é feita extração de características (features) e uma segunda parte onde são usadas essas características para classificar a imagem de entrada.
- A parte que faz extração de características tem tipicamente dois tipos de camadas: as de **convolução** e as de **sub-amostragem** (pooling).
- As camadas de convolução aplicam um filtro aos dados que recebem e geram os chamados mapas de características (feature maps).
- As camadas de sub-amostragem fazem simplesmente uma redução no tamanho desses mapas.

Redes de Convolução: CNNs

- Na prática os filtros aplicados servem para realçar determinadas características das imagens, como sejam arestas ou cores.
- A sub-amostragem torna irrelevante a zona da imagem em que uma dada característica é detetada, gerando assim uma espécie de **invariância** na deteção destas características: não interessa o local exato da imagem onde são detetadas, apenas interessa que estão presentes na imagem.
- A figura da página seguinte, tirada do artigo [9], ilustra alguns dos filtros presentes nas camadas de convolução e como eles representam conceitos mais complexos com a profundidade da respetiva camada.

Redes de Convolução: filtros



Leitura recomendada

- Engelbrecht, sec. 3.1.1, 3.2.1 a 3.2.3, 3.3, 3.4.
- Jorge Salvador Marques, "Reconhecimento de Padrões", IST Press, 1999; ver sec. 5.4.2 para a derivação do algoritmo de retropropagação do erro usando a notação aqui apresentada.

[1] Y. Bengio.
Learning deep architectures for AI.
Foundations and Trends in Machine Learning, 2(1):1–127, 2009.

[2] L. Breiman.
Bagging predictors.
Machine Learning, 26(2):123–140, 1996.

[3] G. Cybenko.
Continuous valued neural networks with two hidden layers are sufficient.
Technical report, Dep. of Computer Science, Tufts University, Medford, MA, 1988.

[4] G. Cybenko.
Approximation by superposition of a sigmoidal function.
Mathematics of Control, Signal and Systems, 2:303–314, 1989.

[5] Ronen Eldan and Ohad Shamir.
The power of depth for feedforward neural networks.

CoRR, abs/1512.03965, 2015.

[6] K. Hornik.
Some new results on neural network approximation.
Neural Networks, 6:1060–1072, 1993.

[7] A. Lapedes and R. Farber.
How neural nets work.
In Anderson, editor, *Neural Information Processing Systems*, pages 442–456. New York:American Institute of Physics, 1987.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278 –2324, nov 1998.

[9] Ivet Rafegas, Maria Vanrell, Luís A. Alexandre, and Guillem Arias.
Understanding trained CNNs by indexing neuron selectivity.
Pattern Recognition Letters, 2019.

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich.
Going deeper with convolutions.
CoRR, abs/1409.4842, 2014.