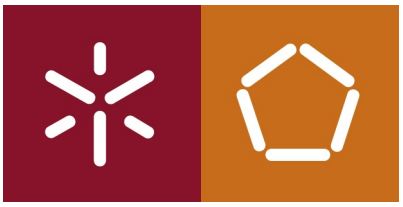


UNIVERSIDADE DO MINHO

OTIMIZAÇÃO EM MACHINE LEARNING

SHALLOW NEURAL NETWORKS

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

GRUPO E

Guilherme Nunes (A80524)
Guilherme Andrade (A80426)
Rui Ribeiro (A80207)
José Costa (A82136)
Rita Pereira (PG41098)
Gabriela Martins (A81987)

7 de Junho de 2020

Conteúdo

Lista de Figuras

Resumo

Quando se ouve falar em redes neuronais, associa-se ao conceito de que estas operam/consistem em múltiplas camadas ocultas. No entanto, existe um tipo de arquitetura que apenas consiste numa camada intermédia, ***Shallow Neural Networks***. Entender este tipo de rede é fundamental para entender o que realmente acontece em redes neuronais profundas, pois com a simplificação oferecida por uma *shallow network*, relativamente a uma *deep neural network*, é possível compreender as operações compostas pelas diversas unidades de cálculo, denominadas por neurónio. Estas unidades determinam a capacidade de generalização de uma rede e, com uma intuição razoável, é possível obter uma melhor interpretação de resultados quando se tratam de redes complexas.

1 Introdução

Shallow Neural Networks é um tipo de arquitetura utilizada nos tempos correntes não só pela sua eficiência, como muitas vezes pela sua efetividade. A utilização de apenas uma camada intermédia permite-nos uma melhor compreensão do funcionamento destes modelos.

Ao longo deste artigo, tenta-se evidenciar o porquê do uso de *shallow neural networks* e a resposta a uma das perguntas mais predominantes na área de *deep learning*, **Do Deep Nets Really Need to be Deep?** [2]

Começa-se por uma explicação teórica de como opera este tipo de rede através de uma vertente matemática complementar.

Após a descrição do modelo, coloca-se este modelo em teste com uma base de dados simples e adequada ao problema, respetivamente o *xor*, devido tanto às propriedades intrínsecas de uma boa *baseline* como ao seu contexto histórico.

Esta operação binária representa o mais simples conjunto linearmente não separável, facilitando um potencial *debugging* do algoritmo e providenciando resultados relativamente simples de analisar.

Após mostrar o propósito das *Shallow Neural Networks*, utilizam-se exemplos mais diversificados e complexos que o *xor* para que se consiga identificar o nível de generalização deste tipo de modelos e o porquê da utilização de redes neuronais profundas.

As *Shallow Neural Networks* apresentam maior simplicidade, interpretabilidade e um menor custo computacional. No entanto, dado que as redes mais profundas conseguem aprender distribuições mais complexas[9], estas apresentam potencialmente melhores resultados, dependendo do problema em causa, apresentando custos computacionais mais elevados.

Nesse sentido, o principal objetivo deste trabalho é a análise e interpretação dos resultados, a nível de dados e capacidade de inferência, de utilização de apenas uma ou várias camadas numa rede neuronal.

2 Shallow Neural Network

Como já referenciado, este tipo de redes consiste, geralmente, em valores de entrada(*input*), uma camada intermédia (*hidden layer*) e uma camada de saída(*output*), e através deste tipo de redes consegue-se entender melhor o que acontece numa rede neuronal profunda tal como com nas suas camadas intermédias.

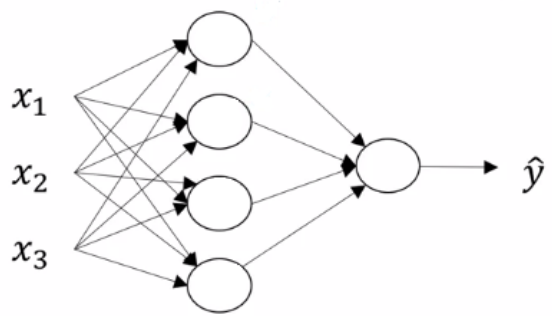


Figura 1: Exemplo da estrutura de uma *shallow neural network*

Começa-se por definir os símbolos matemáticos utilizados para descrever o processo de otimização de uma rede neuronal.

- X representa a matriz de inputs, em que as linhas correspondem ao número de exemplos e as colunas o número de *features*;
- W representa a matriz de pesos;
- b representa o vetor de *biases*;
- Z representa o vetor calculado entre os pesos, *biases* e *inputs*;
- g representa a função de ativação;
- A representa o vetor de ativação obtido após utilizar uma função de ativação em Z;
- Y representa a matriz de resultados;
- lr representa o *learning rate*;

Precisa-se também de referir que funções de ativação [11] foram utilizadas ao longo do trabalho. Estas podem ser representadas tanto matematicamente como graficamente da seguinte forma:

1. *relu* (*Rectified Linear Unit*):

$$f(x) = \begin{cases} x & \text{se x} > 0 \\ 0 & \text{se x} \leq 0 \end{cases} \quad (1)$$

2. *Leaky reLU* (Leaky *Rectified Linear Unit*):

$$f(x) = \begin{cases} x & \text{se x} > 0 \\ 0.01x & \text{se x} \leq 0 \end{cases} \quad (2)$$

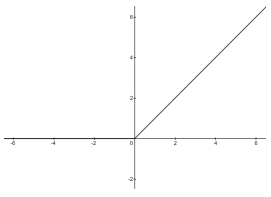


Figura 2: Função de ativação *relu*

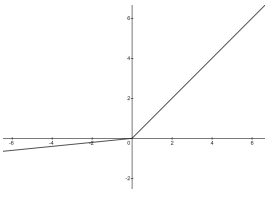


Figura 3: Função de ativação *leaky_ relu*

3. *Sigmoid*:

$$f(x) = \frac{1}{1 + e^{-x}}$$

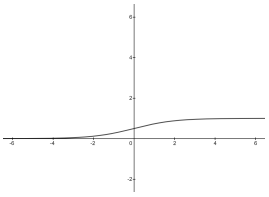


Figura 4: Função de ativação *sigmoid*

4. *Tanh*:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

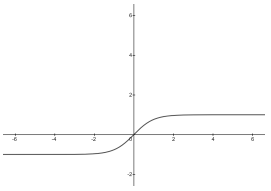


Figura 5: Função de ativação *tanh*

5. *Swish*:

$$f(x) = \frac{x}{1 + e^{-x}}$$

.

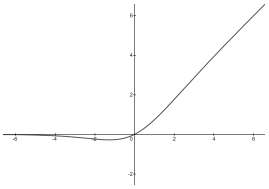


Figura 6: Função de ativação *swish*

2.1 Função de custo

As redes neuronais são treinadas com várias funções de objetivo, sendo que pode se tornar um passo complicado na procura da melhor função, isto porque uma rede neuronal tenta formalizar uma função entre o conjunto de *inputs* para um conjunto de *outputs* através dos pesos presentes na rede.

É muitas vezes impossível ou inviável calcular os pesos "perfeitos"de uma rede neuronal, sendo que existem bastantes variáveis a ter em consideração, tornando-se num problema complexo.

Em vez disso, o objetivo, através do processo de otimização na função objetivo corrente,é navegar pelo espaço de possíveis conjuntos de pesos com a tarefa de maximizar ou minimizar a mesma e e tentar encontrar a melhor solução para o problema [7].

Todos os problemas que serão descritos durante este artigo têm uma classificação binária, sendo que apenas podem tomar dois valores. Dito isto, decidiu-se escolher uma função binária, respetivamente *binary cross-entropy / log loss* [6], como função objetivo dos nossos problemas. Esta função é descrita da seguinte forma(C representa a função objetivo):

$$C = \frac{-1}{N} * \sum Y * log(A) + (1 - Y) * log(1 - A)$$

Uma breve explicação da mesma, é feita da seguinte forma:

Quando o valor esperado de Y é 1, então estamos perante a primeira parte da equação, caso seja 0 estamos na segunda parte da equação, respetivamente $(1 - Y) * log(1 - A)$, sendo que como $Y = 0$, apenas fica-se com $log(1 - A)$, em que A representa a previsão do valor de entrada X associado ao valor de saída Y. Como podemos observar pela imagem 7, quando o valor binário de saída é 0, então queremos que o valor A seja o mais próximo de 0 possível pois $log(1)$ e assim o custo associado a este valor de entrada é 0, sendo que não houve erro. O objetivo é utilizar a mesma lógica para todo o conjunto de dados, daí o somatório no início da função.

Por fim, divide-se por $\frac{1}{N}$, para fazer a média. O sinal - é utilizado pois o logaritmo entre $[0, 1]$ é um conjunto de números negativos.

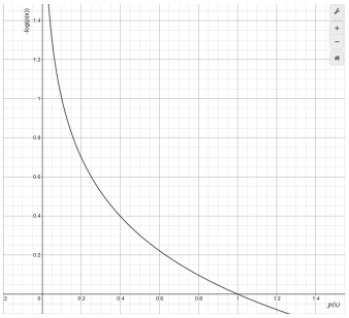


Figura 7: Log Loss para $A = 0$

Todo este processo é semelhante caso $Y = 1$, a apenas temos outra função, respetivamente $-log(A)$, representado na seguinte figura.

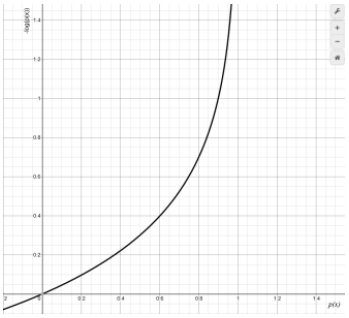


Figura 8: Log Loss para $A = 1$

O processo de treino de uma *Shallow neural network* contêm 3 passos, que serão aqui descritos e explicados.

- *Feedforward* que representa a propagação do *input* pela rede;
- *Backpropagation* que realiza os cálculos do gradiente da função custo relativamente aos pesos e *biases*;
- *Update parameters* que realiza um *update* dos parâmetros calculadas na etapa anterior.

2.2 FeedForward

A parte elementar de uma rede é naturalmente os neurónios. Estes, dado um *input*/X, fazem inicialmente o cálculo de um vetor Z, utilizando os valores recebidos como *input* e os pesos e *biases* já predefinidos.

A predefinição dos pesos passa por um processo de inicialização destes mesmos. É um processo importante pois define um ponto a partir do qual se vai desenvolver o algoritmo.

Tendo isto em conta, para dois modelos com a mesma arquitetura, se inicializados com valores diferentes, poderão atingir resultados diferentes em função da distância e da disposição destes ao potencial ponto de convergência.

Existem várias abordagens desde inicialização aleatória, *Glorot*, *He* [8], entre outros. Neste trabalho optou-se pela inicialização aleatória pois esta abordagem traduz-se como simples e já apresenta condições para quebrar a simetria de pesos[13]. Esta última consiste no fenómeno em que uma inicialização de pesos com todos os valores, por exemplo a 0, tem como sinal para todas as unidades de todas as camadas 0 também, causando que, para qualquer *input*, independentemente da classe, o resultado da ativação será sempre o mesmo até à(s) unidade(s) de *output*. Consequentemente, devido ao funcionamento da *backpropagation* e à forma como se determina o erro proporcional ao valor dos pesos, todas as unidades irão receber a mesma intensidade de sinal e a

atualização será igual para todos os pesos, impedindo a rede de aprender quaisquer *features*. Por outro lado, existe ainda os problemas de *vanishing e exploding gradients*, sendo estes problemas mais associados a redes profundas.

O primeiro traduz uma perda ao longo da cadeia de camadas, na *backpropagation*, em que os valores dos gradientes vão sendo cada vez mais pequenos até chegar a uma situação de *underflow*, ou seja, o sistema deixa de ter capacidade de representar um valor demasiado pequeno. O mesmo pode acontecer para *exploding gradients* mas no sentido inverso, em que durante o processo do cálculo do gradiente, os valores tomam um crescimento exponencial atingindo *overflow*, ou seja, o sistema deixa de conseguir representar valores demasiado grandes. Quanto estes problemas de representação ocorrem, são usados *NaNs, not a number*, causando instabilidade numérica.

A inicialização pode definir a possibilidade(ou impossibilidade) de aprendizagem assim como a velocidade e capacidade de convergência, logo deve ser um processo considerado importante e devidamente deliberado.

Em seguida ao cálculo de Z, é usada uma função de ativação que usa o valor calculado anteriormente e passa-o para a camada seguinte. Estas funções de ativação têm como objetivo adicionar uma não linearidade à rede, tornando-as potencialmente capazes de modelar funções mais complexas, caso contrário, o valor passado para os neurónios da próxima camada seriam apenas uma combinação linear dos pesos e do *biases* e uma rede seria apenas uma combinação linear de todas as camadas.

Este processo é definido da seguinte forma, em que *super script* L(que só serve para representação) representa o último *layer*, e $L-1$ representa *layer* anterior e A^{L-1} corresponde ao *input*/X na primeira layer.

$$Z^L = A^{L-1} * W^L + b^L$$
$$A^L = g^L(Z^L)$$

2.3 Backpropagation

Após a propagação dos inputs pela rede, obteve-se as previsões associadas aos mesmos no vetor A^L. Através deste e dos valores de saída, utilizamos a função de custo definida anteriormente para calcular o erro associado aos parâmetros utilizados. Estes parâmetros iniciais definidos aleatoriamente apenas são um ponto de partida para uma potencial solução.

De modo a obter qual a direção que cada peso tem que seguir, é necessário realizar as derivadas parciais de cada peso associadas à função objetivo.

Dado este tipo de arquitetura presente na imagem 1, consegue-se observar que apenas temos os valores de entrada e duas camadas, a camada intermédia e camada de saída.

Sendo assim tem-se dois vetores referentes aos *biases* e duas matrizes referentes aos pesos:

- $W1$, uma matriz com o número de linhas equivalentes ao número de entradas, e o número de colunas respetivo ao número de neurónios presentes na camada intermédia e o seu respetivo vetor de *biases* $b1$, neste caso 1 *biases* por cada neurónio na camada intermédia.
- $W2$, uma matriz constituída pelo número de neurónios na camada intermédia por 1, pois é um problema binário e o seu respetivo *biases* $b2$.

Este processo é definido da seguinte forma em que C representa a função custo e N representa o tamanho do conjunto.

$$C = \frac{-1}{N} * \sum Y * log(A^L) + (1-Y) * log(1-A^L)$$

$$A^L = g^L(Z^L)$$

$$Z^L = W^L * A^{L-1} + b^L$$

Primeiramente tem-se que calcular as derivadas parciais para cada peso e *biases* utilizados na rede, começando pelo último *layer* tem-se:

$$\frac{\partial C}{\partial W^L} = \frac{\frac{\partial C}{\partial A^L} * \frac{\partial A^L}{\partial Z^L} * \frac{\partial Z^L}{\partial W^L}}{N}$$

Decompondo todas estas derivadas parciais obtém-se:

$$\frac{\partial C}{\partial A^L} = -(\frac{Y}{A} + \frac{1-Y}{A-1}) = -(\frac{Y(A-1)*(1-Y)*A}{A(A-1)}) = -(\frac{-Y+A}{A*(A-1)}) = -(\frac{-(Y-A)}{-A*(1-A)}) = \frac{A-Y}{A*(1-A)}$$

$$\frac{\partial A^L}{\partial Z^L} = g'^L(Z^L)$$

em que g^L representa a função de ativação utilizada no segundo *layer*. Como neste artigo utiliza-se *logistic classifier* por norma no segundo *layer*, obtém-se a seguinte equação.

$$\frac{\partial A^L}{\partial Z^L} = sigmoid(1 - sigmoid(Z^L)$$

visto que $A^L = sigmoid(Z^L)$

$$\frac{\partial A^L}{\partial Z^L} = A(1 - A(Z^L))$$

$$\frac{\partial Z^L}{\partial W^L} = A^{L-1}$$

Voltando à derivada parcial de W^L temos que:

$$\frac{\partial C}{\partial W^L} = \frac{\frac{\partial C}{\partial A^L} * \frac{\partial A^L}{\partial Z^L} * \frac{\partial Z^L}{\partial W^L}}{N}$$

$$\frac{\partial C}{\partial W^L} = \frac{\frac{A-Y}{A(1-A)} * A(1-A) * (A^{L-1})}{N}$$

$$\frac{\partial C}{\partial W^L} = \frac{(A-Y)*(A^{L-1})}{N}$$

Todo este processo é semelhante para b^L, obtendo a seguinte equação:

$$\frac{\partial C}{\partial b^L} = \frac{A-Y}{N}$$

Em relação às derivadas parciais da primeira camada, é de notar que a ativação utilizada em $L-1$ não está presente na função objetivo diretamente, pois esta só necessita dos resultados de saída presentes em A^L, sendo assim temos que:

$$\frac{\partial C}{\partial W^{L-1}} = \frac{\frac{\partial C}{\partial A^{L-1}} * \frac{\partial A^{L-1}}{\partial Z^{L-1}} * \frac{\partial Z^{L-1}}{\partial W^{L-1}}}{N}$$

$$\frac{\partial A^{L-1}}{\partial Z^{L-1}} = g'^{L-1}(Z^{L-1})$$

Sendo que g^{L-1} representa a função de ativação presente no primeiro *layer*, podendo esta ter qualquer formato desde que fosse passado como parâmetro a função de derivação associado à função de ativação.

$$\frac{\partial Z^{L-1}}{\partial W^{L-1}} = A^{L-2}$$

Visto que a ativação antes do primeiro *layer* são os inputs então obtém-se:

$$\frac{\partial Z^{L-1}}{\partial W^{L-1}} = X$$

Agora apenas ficou-se com $\frac{\partial C}{\partial A^{L-1}}$. Para resolver este problema utiliza-se a regra da cadeia visto que A^{L-2} não está diretamente presente na função custo e obtemos:

$$\frac{\partial C}{\partial A^{L-1}} = \frac{\partial C}{\partial A^{L}} * \frac{\partial A^{L}}{\partial Z^{L}} * \frac{\partial Z^{L}}{\partial A^{L-1}}$$

$$\frac{\partial C}{\partial A^{L}} * \frac{\partial A^{L}}{\partial Z^{L}} = A^{L} - Y^{L}$$

$$\frac{\partial Z^{L}}{\partial A^{L-1}} = W^{L}$$

logo obtém-se,

$$\frac{\partial C}{\partial W^{L-1}} = g'^{(Z^{L-1})} * X * A^{L} - Y^{L} * W^{L}$$

Todo este processo é semelhante para b^{L-1}, obtendo a seguinte equação:

$$\frac{\partial C}{\partial b^{L-1}} = g'^{(Z^{L-1})} * A^{L} - Y^{L} * W^{L}$$

2.4 Udpate parameters

Após ter as derivadas parciais de cada peso e *biases* procede-se o *udpate* dos mesmos, através do algoritmo do gradiente descendente[12]. Este processo é descrito da seguinte forma:

$$W1 = W1 - lr * dW1$$

$$W2 = W2 - lr * dW2$$

$$b1 = b1 - lr * db1$$

$$b2 = b2 - lr * db2$$

Este processo é dos, se não o mais importante processo de otimização. O hiper-parâmetro *lr* (learning rate ou taxa de aprendizagem) pode ser interpretado como o salto que os parâmetros dão no caminho dos parâmetros ideais.

Uma taxa de aprendizagem muito pequena implica que as alterações nos pesos sejam mais pequenas e o processo de otimização mais longo(entre outros fatores)[4], e, caso a taxa seja demasiada grande, pode dar saltos indesejados, tal saltos que podem impedir este processo de otimização de obter uma solução satisfatória, como se pode observar pela figura 9.

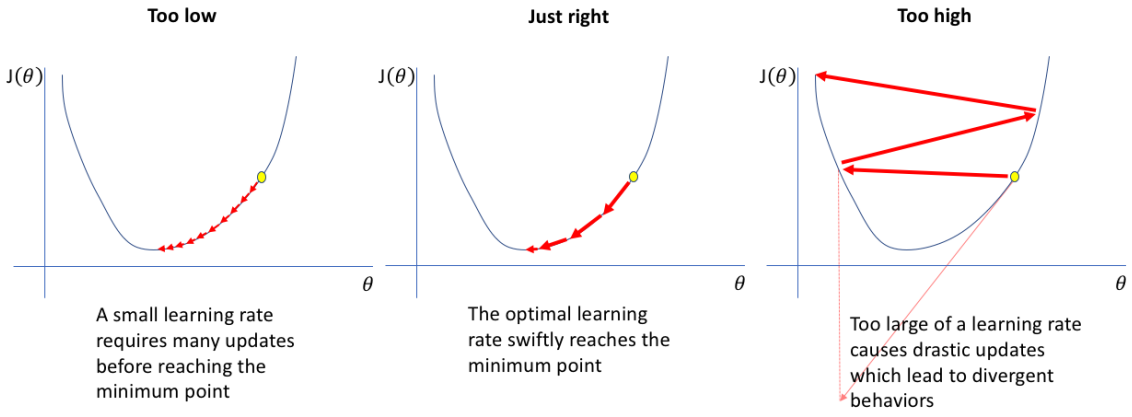


Figura 9: Taxa de aprendizagem

Quando o *learning rate* é demasiado elevado, verifica-se um cenário de divergência na função objetivo. A justificação para tal baseia-se no facto de que ao dar saltos demasiado grandes, pode-se ultrapassar as potenciais soluções ótimas (ou semi ótimas).

Se o salto originar pesos onde o gradiente apresenta um maior valor absoluto de declive que o conjunto de pesos anterior, o próximo *update* poderá originar um salto ainda maior originando um fenómeno de *overshoot* ainda mais agravado que o salto anterior, provocando um afastamento dos valores dos pesos atuais aos valores dos pesos ótimos (ou semi ótimos) relativamente à *loss*, tendo impacto direto no desempenho do modelo.

3 Introdução ao *Xor*

Depois de se ter explicado, em parte, como funciona a implementação de uma *shallow neural network* e também os princípios teóricos que a fundamentam, precisa-se de testar, na prática, se é possível que o modelo consiga avaliar corretamente uma base de dados linearmente não separável.

A maneira mais fácil de o comprovar, é utilizando a problemática do *xor* [15]. Como se sabe, o *xor* não é linearmente separável uma vez que, com uma reta, não é possível separar os *outputs* 0 e 1, como apresentado nas imagens a seguir.

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

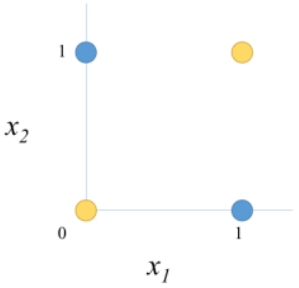


Figura 10: Representação tabelar e gráfica da base de dados *xor*

Primeiro, começou-se por testar com um neurónio na camada intermédia para que se possa justificar a utilização de mais neurónios nessa mesma camada.

É de realçar que para as experiências seguintes foi usada a função de ativação *relu*, no entanto, estes resultados também se aplicam às outras funções de ativação.

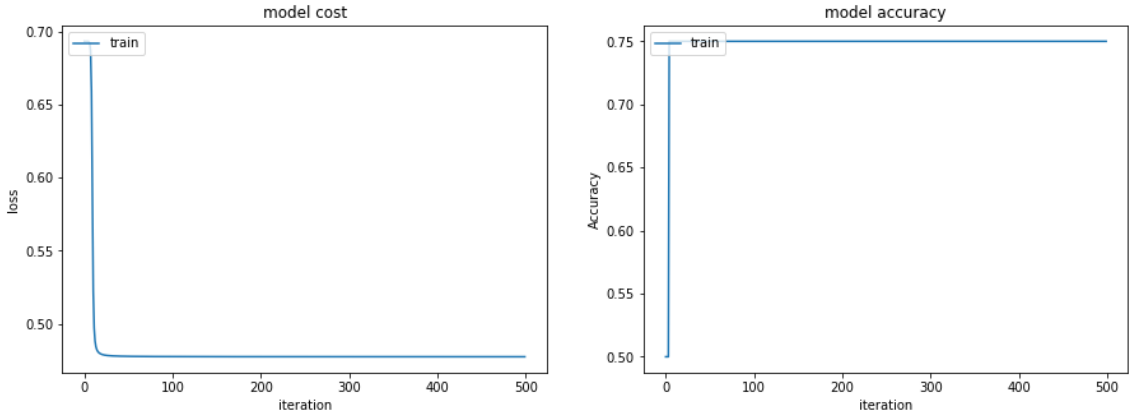


Figura 11: Gráficos de aprendizagem do *xor* com 1 neurónio na camada intermédia de 100 em 100 *epochs*

Como era de esperar, mesmo após 50 mil *epochs* (o custo/*loss* é avaliado a cada 100 *epochs*) a precisão nunca alcança os 100%. Então, aquilo que resta fazer, a seguir, é aumentar o número de

neurónios na camada intermédia e ver que efeito tem na separação dos valores da base de dados *xor*. Em seguida, passou-se o número de neurónios para 2 e obteve-se resultados bastante positivos.

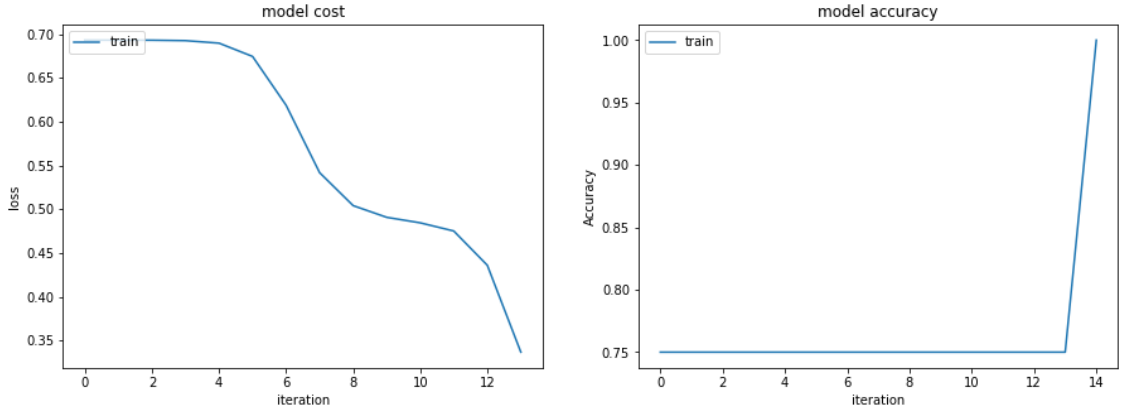


Figura 12: Gráficos de aprendizagem do *xor* com 2 neurónios na camada intermédia de 100 em 100 *epochs*

Pelo que se consegue observar dos gráficos em cima, após 1400 *epochs*, o modelo consegue alcançar precisão 100% pelo que se percebe que conter um número de neurónios maior que 1 na camada intermédia resolve o problema das bases de dados não linearmente separáveis. Para termos mais um parâmetro de comparação, testou-se também um modelo com 4 neurónios na camada intermédia.

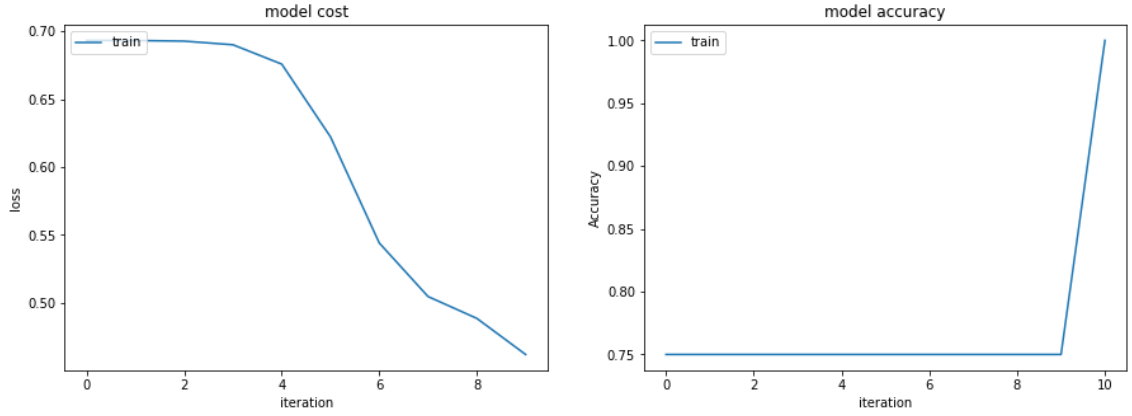


Figura 13: Gráficos de aprendizagem do *xor* com 4 neurónios na camada intermédia de 100 em 100 *epochs*

Mais uma vez, através dos gráficos, consegue-se provar que ter vários neurónios intermédios fazem a diferença na resolução desta problemática, tornando, por vezes, a convergência mais rápida.

4 Metodologia

Após a verificação do funcionamento do algoritmo para as *shallow neural networks*, procedeu-se à uniformização de uma série de processos que permitissem uma análise sistemática para vários casos de estudo. Esta abordagem apresenta-se como uma mais valia no sentido de ter bases pelas quais se pode tirar conclusões e encarar, com espírito crítico, a realidade que estas impõem.

Este processo começa pela definição de uma arquitetura de rede base, a partir da qual se constrói todo um raciocínio à medida que se vai aumentando a complexidade do modelo (se for o caso).

A rede base vai ser constituída por uma camada intermédia com apenas uma unidade e uma camada de *output* com uma unidade responsável pela classificação binária.

Como processo de análise vão ser alteradas as ativações das unidades da rede de modo a estabelecer comparações de tempos de execução assim como de convergência dos dados. Esta análise providenciará uma maior intuição para as consequências que a escolha da função de ativação tem para a precisão do modelo e para os tempos de computação exigidos para cada uma em termos de operações e de *epochs* necessárias para atingir valores de *loss* satisfatórios. As funções de ativação utilizadas serão a *sigmoid, relu, leaky relu, tanh* e *swish*. Como complementação à análise, vão ser feitas *learning curves* e uma matriz de confusão de forma a ter uma interpretação mais precisa no que toca à relação entre a classificação prevista e a classificação real dos dados. Por fim, é feito uma análise visual dos pesos obtidos de cada modelo, de forma a inferir a necessidade de manter ou diminuir um certo número de unidades na camada intermédia. Esta parte poderá ser mais subjetiva e a decisão tomada poderá variar consoante o que o contexto atual exige.

Este mesmo processo vai ser executado para redes progressivamente mais complexas formadas a partir da rede base. As redes mais complexas vão ser constituídas por 2, 3, 40 e 100 unidades na camada intermédia. A escolha destes números deriva da necessidade de verificar o comportamento em termos de escalabilidade das funções de ativação assim como a capacidade de generalização duma *shallow neural network*.

5 Casos de Estudo

5.1 Descrição do caso de estudo número 1

O primeiro *dataset* é constituído por 1500 imagens, com 12x12 pixeis, sendo metade destas constituídas por círculos e outras por quadrados, como podemos observar na figura 14. Também se acrescenta que é um problema de classificação binária e que este *dataset* é linearmente separável.

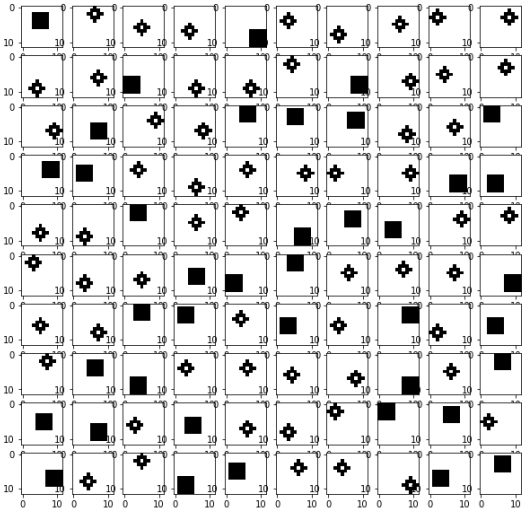
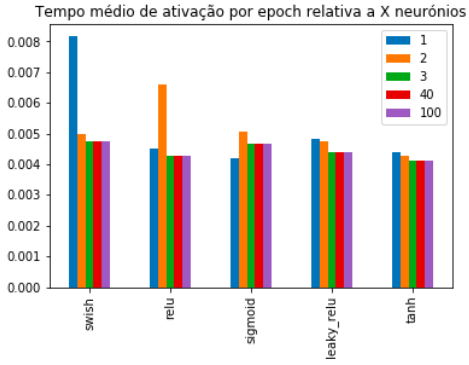


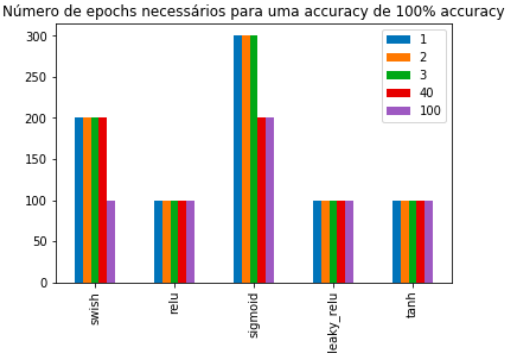
Figura 14: Primeiro caso de estudo

5.1.1 Análise de resultados

Como descrito anteriormente, este dataset é linearmente separável pelo que com apenas um neurónio consegue-se obter uma classificação ótima. Testou-se várias combinações de neurónios na camada intermédia e várias funções de ativação e os resultados podem ser observados na seguinte figura:



(a) Tempo médio por *epoch*



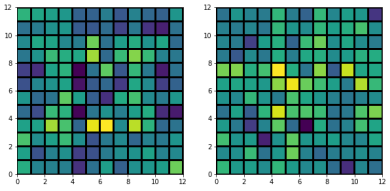
(b) Número de *epochs* para *accuracy* de 100%

Os tempos descritos em cima são apenas para o leitor ter uma noção do custo de cada tipo de ativação no ambiente descrito. Sobre os resultados, como se consegue observar, todas as respetivas funções de ativação para os respetivos neurónios conseguiram obter uma *accuracy* satisfatória. Isto era de esperar, pois o primeiro caso de estudo foi especificamente escolhido dado a sua simplicidade.

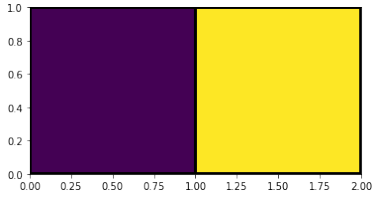
Sobre o número de *epochs* necessário referentes a cada função de ativação na sua convergência, todas estas conseguem ter um *score* perfeito e não se consegue fazer muitas observações visto que todas elas convergem em 100 *epochs*. Dito isto, não foram feitas as *learning curves*.

A função de ativação que necessita de mais *epochs* para atingir os 100% de *accuracy*, segundo a imagem apresentada acima, é a *sigmoid*.

Como a função de ativação *tanh* apresenta, no geral, melhores resultados, tanto no tempo médio por *epoch* para vários valores de neurónios na camada de ativação, como para número de *epochs* necessárias para uma *accuracy* de 100%, foi a função utilizada para apresentar as próximas conclusões.

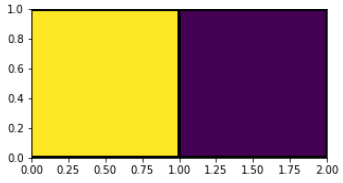


(a) *First Layer*

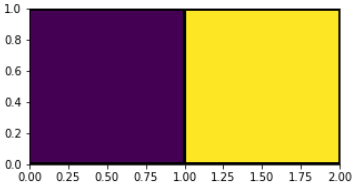


(b) *Second Layer*

Figura 16: Pesos da camada intermédia com 2 unidades



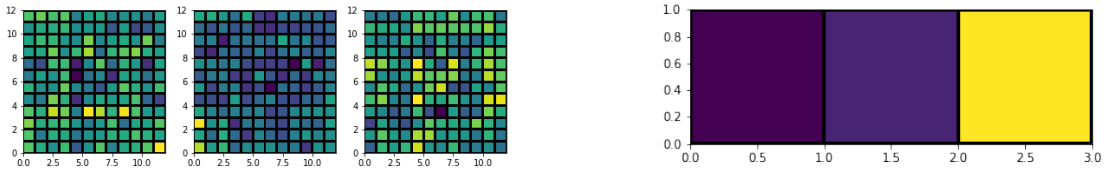
(a) Camada intermédia da *Label* 0



(b) Camada intermédia da *Label* 1

Pela figura, analisando as imagens relativas aos valores resultantes das somas pesadas efetuadas

pelas unidades da camada intermédia, ou seja, antes da ativação não linear, verifica-se que, para instâncias de dados com classificações reais opostas, a cor da direita de uma instância troca com a da esquerda da outra e vice-versa. Isto pode ser visto e comparado à utilização de dois *bits* para a representação de dois valores apenas, ou seja, da mesma forma que se pode usar apenas um bit para representar dois valores possíveis (natureza binária), também se pode utilizar apenas uma unidade para representar o valor associado àquelas somas pesadas. Conclui-se assim que para este caso, não seriam necessárias duas unidades intermédias e que uma apenas chegaria para atingir valores de *accuracy* e *loss* satisfatórios.



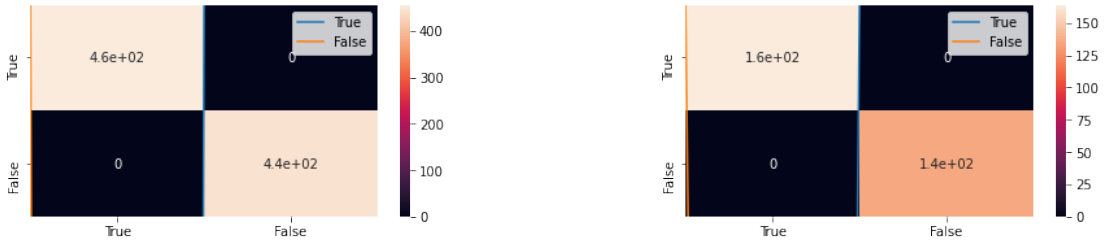
(a) *First Layer* (b) *Second Layer*

Figura 18: Pesos da camada intermédia com 3 unidades

(a) Camada intermédia da *Label* 0 (b) Camada intermédia da *Label* 1

As figuras acima mostram as representações, para cada uma das *labels*, das camadas intermédias com 3 unidades. Tal como tinha sido concluído acima, já com 2 unidades na camada intermédia é possível responder ao problema.

Mais neurónios na camada intermédia significa que a rede será mais específica às nuances dos dados. No entanto, aumentar o número de unidades para 3 na camada intermédia quando é possível usar apenas 2 unidades para responder ao problema, resulta numa ineficiência de recursos, dado que o tempo computacional não justifica este aumento.



(a) Matriz de Confusão do *dataset* de treino (b) Matriz de Confusão do *dataset* de test

Figura 20: Matriz de Confusão para a rede com função de ativação *tanh* e 150.000 *epochs*

Como é possível observar pela figura 20, bem como pelas conclusões tiradas no início desta secção de Análise de Resultados, a rede classifica corretamente 100% dos casos, quer de treino ou de teste, resultando em 0 *false negatives* e 0 *false positives* [3].

Para qualquer função de ativação e número de neurónios na camada de ativação, é possível chegar aos mesmos resultados em 30 mil*epochs* (os epochs são registados de 100 em 100) ou menos.

5.2 Descrição do caso de estudo número 2

O segundo caso de estudo refere-se a um conjunto de dados mais complicado, sendo que este não é linearmente separável e é constituída por 600 imagens, com 8 * 8 pixeis. Metade destes são constituídos por retângulos verticais e outra metade por retângulos horizontais como podemos observar na figura 21. Também se consegue observar que este é um problema de classificação binária.

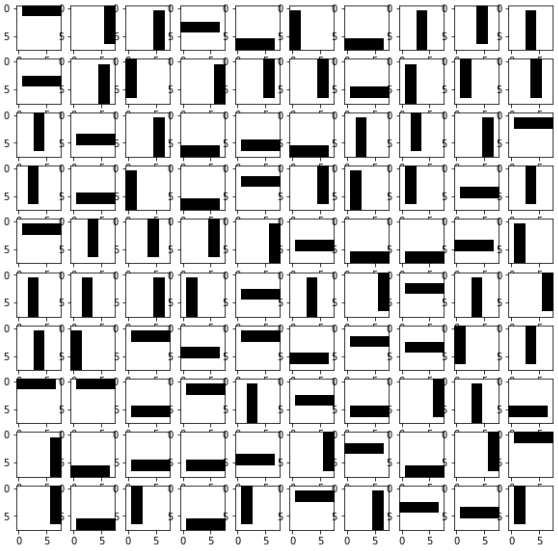


Figura 21: Segundo caso de estudo

5.2.1 Análise de resultados

Este *dataset*, como se apresenta como um caso consideravelmente mais complexo, espera-se que apresente resultados com maior conteúdo em termos de interpretação.

Para a verificação disto, procedeu-se novamente ao teste de várias combinações entre o número de unidade da camada intermédia e funções de ativação.

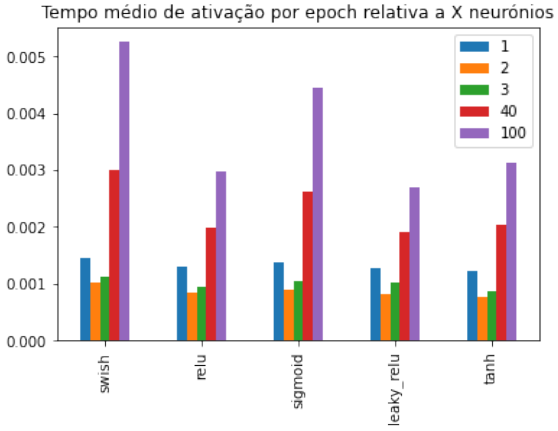


Figura 22: Tempo médio por *epoch*

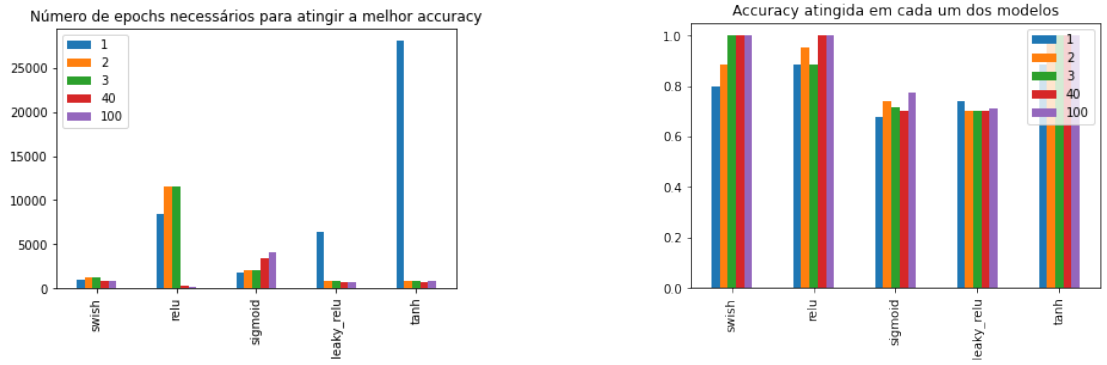
A figura 22 apresenta tempos médios de execução maiores que os do *dataset* anterior.

Isto pode dever-se à complexidade intrínseca a cada um. Este conjunto de dados, de uma forma visual, poderia-se concluir que este apresenta um maior volume de *pixels* relevantes, o que aumentaria o número de neurónios ativados por cada instância. Por sua vez, esta maior taxa de

ativação teria consequências no tempo de execução da *backpropagation*, que, no cálculo dos diversos gradientes dos diversos parâmetros, um gradiente nulo poderá fazer a diferença entre um cálculo relativamente rápido e um relativamente lento.[1]

A questão aqui passa pelo facto de os tempos não manterem as proporções entre si, relativamente à variação do número de unidades, relativamente ao gráfico representado na figura 15a.

Esta variação foi associada ao *CPU*[14] em si, tendo em conta que os modelos foram treinados localmente. A atividade no processador (outras aplicações), na altura em que se treinava um ou outro modelo, poderia influenciar ligeiramente os tempos e produzir impurezas na amostra temporal demonstrada. Por outro lado, existe a possibilidade do aproveitamento do cálculo vetorial efetuado não ser otimizado para uma unidade computacional e ser otimizado para duas e três. Isto poderia-se dever à otimização construída na arquitetura dos processadores centrais.



(a) Número de *epochs* da melhor *accuracy*

(b) *accuracy* de cada modelo

Figura 23: Análise dos modelos

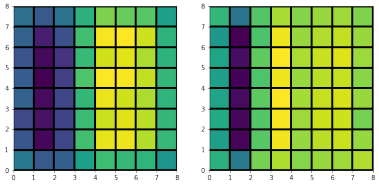
A figura 23 apresenta-se como uma medida de desempenho mais direta, tendo em conta o número de *epochs* necessárias para atingir o melhor valor possível de *accuracy*, dentro de um limite máximo definido e também o correspondente melhor valor possível mediante a ativação e número de unidades.

Conjugando a informação das duas figuras, o primeiro aspeto importante passa pela noção de que com apenas uma unidade na camada intermédia, não foi possível atingir uma *accuracy* de 100%, indicando que este conjunto de dados apresenta uma complexidade demasiado grande para um modelo composto apenas por um neurónio na camada oculta, visto ser um conjunto de dados linearmente não separável.

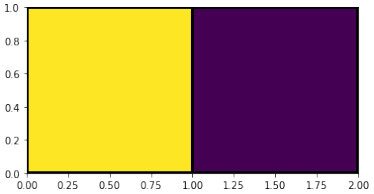
A partir da utilização de duas unidades, já foi possível chegar a uma *accuracy* satisfatória.

Considerando a amostra suficientemente demonstrativa, identifica-se novamente que a tangente hiperbólica apresenta menores tempos de computação à exceção de uma unidade que traduz por um pico na figura 23a, que poderá estar associada a pequenos progressos na *loss* durante o treino, mas não suficientes tendo em conta o panorama geral. Menciona-se ainda a *swish* que apresenta valores satisfatórios a partir de duas unidades, e a relu que apresenta os mesmo a partir das 40 unidades apesar de quebrar um pouco nos tempos de desempenho.

Como a função de ativação *tanh* foi a única função que conseguiu, com 2 unidades na camada de ativação e 30 mil *epochs* predefinidos, chegar a uma *accuracy* de 100% e a função *swish* conseguiu uma *accuracy* na ordem dos 80%, procedemos à análise das diferenças, em termos de visualização da camada intermédia, dos resultados destas funções.

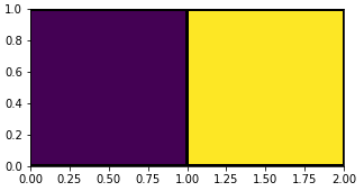


(a) *First Layer*

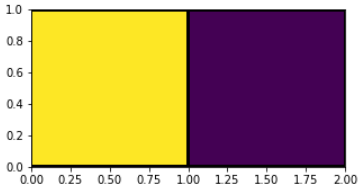


(b) *Second Layer*

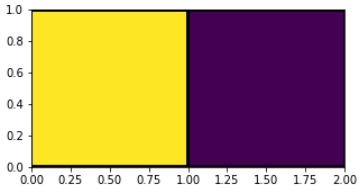
Figura 24: Pesos da camada intermédia com 2 unidades e função de ativação *tanh*



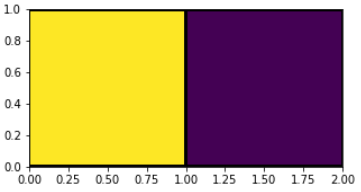
(a) Camada intermédia da *Label* 0



(b) Camada intermédia da *Label* 1



(c) Camada intermédia da *Label* 0



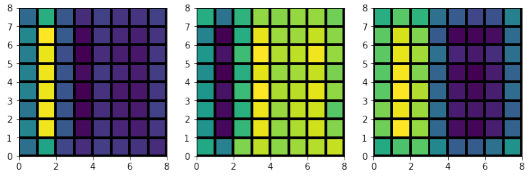
(d) Camada intermédia da *Label* 1

Figura 25: Camada intermédia da *swish* (a e b) e da tangente hiperbólica (c e d)

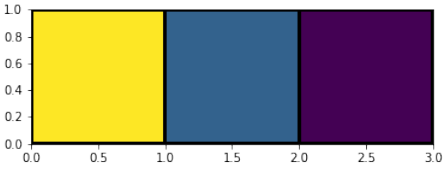
A figura 25 permite comparar, para cada função de ativação identificada, a representação da camada intermédia para cada uma das *labels* do problema. Enquanto que na função *swish* existe uma troca entre as cores das instâncias, em cada unidade, na função *tanh* não existe diferença visual.

Considerando o valor de *accuracy* de cada uma das funções e os aspectos visuais ilustrados anteriormente, conclui-se que a função *tanh* com esta representação, com apenas dois neurónios na camada intermédia consegue representar todas as características distintas de cada *label*, enquanto que a função *swish* necessita de mais unidades (utilizando as mesmas *epochs*) ou mais *epochs* para o conseguir fazer.

A estrutura fixa apresentada pela camada intermédia com a função *tanh*, sendo que esta foi a única que conseguiu apresentar *acuuracy* de 100% e, portanto, responder totalmente ao problema, indica que a mesma não conseguiria apresentar uma solução com *accuracy* considerável com apenas 1 unidade na camada intermédia, indo em conta com o facto do problema ser linearmente não separável.

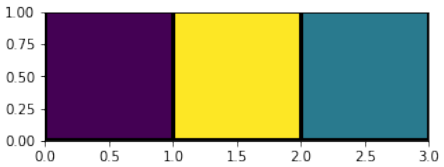


(a) *First Layer*

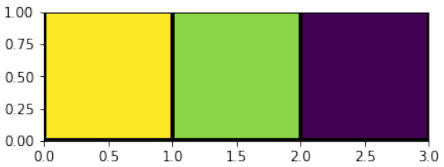


(b) *Second Layer*

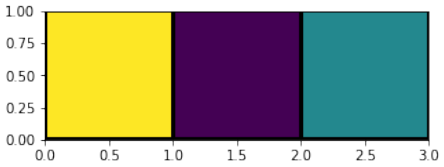
Figura 26: Pesos da camada intermédia com 3 unidades e função de ativação *tanh*



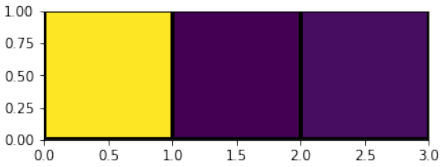
(a) Camada intermédia da *Label* 0



(b) Camada intermédia da *Label* 1



(c) Camada intermédia da *Label* 0



(d) Camada intermédia da *Label* 1

Figura 27: Camada intermédia da função de ativação *swish* (a e b) e da tangente hiperbólica (c e d)

Procede-se agora a uma mesma análise com 3 neurónios na camada intermédia.

Como podemos observar pela figura 16b consegue-se analisar que com 3 unidades obtêm-se uma *accuracy* de 100% para a função de ativação *swish*, que para duas unidades e 30 mil *epochs* não tinha conseguido.

Na figura 27 estão ilustradas as camadas intermédias das diferentes funções de ativação. Mesmo observando uma diferença de cores entre todas as unidades do *swish* para os *label* 0 e 1, afirma-se que estas cores nem sempre divergem.

Foram testados vários casos(não apresentados aqui dado a dimensão do relatório) em que apenas duas cores mudavam, enquanto que uma unidade na camada intermédia permanecia inalterável.

Isto permite concluir que, com um número de *epochs* mais elevado e apenas duas unidades na camada intermédia, esta função de ativação potencialmente atingiria os 100% de *accuracy*.

No caso da função de ativação *tanh*, que já apresentava uma estrutura que respondia ao problema com uma *accuracy* de 100%, ao haver um acrescento de uma nova unidade, apenas resultou na adição de mais um conjunto de variáveis, resultando num acréscimo do final da estrutura inicial apresentada na figura 25. Como já tinha sido explicado na secção anterior, como a rede já responde ao problema com apenas 2 unidades, apenas teríamos vantagens na utilização de 3 ou mais unidades caso o processo seja computacionalmente mais rápido (de acordo com os tempos de ativação verificados anteriormente), mas como referido e evidenciado na figura 23b, isto não acontece.

As *learning curves* do modelo que apresentou, em geral, melhores resultados de *performance* (*tanh*) estão ilustradas na figura 28 e atingiu resultados excelentes em apenas 700 epochs.

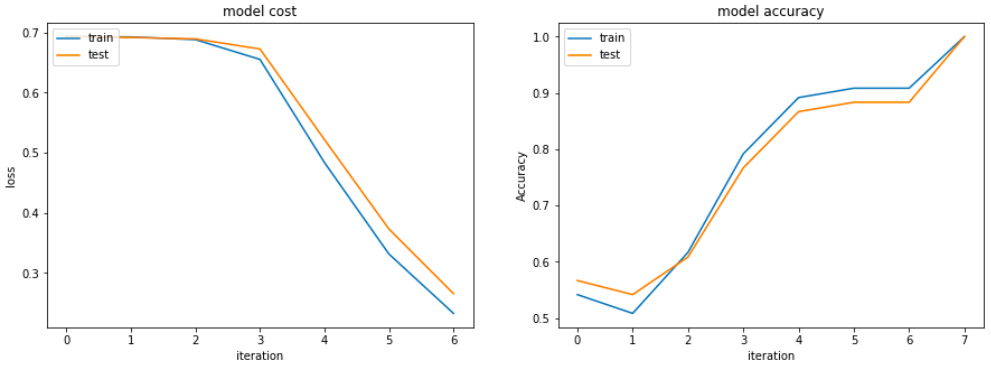
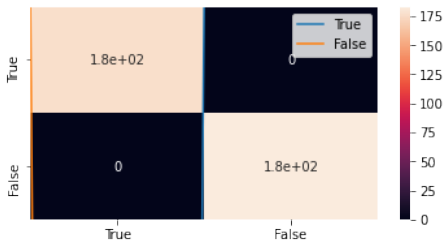
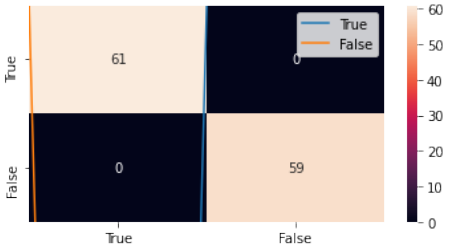


Figura 28: *Learning curves*

Como é possível observar pela figura 29, bem como pelas conclusões tiradas no início desta secção de Análise de Resultados, a rede classifica corretamente 100% dos casos, quer de treino ou de teste, resultando em 0 *false negatives* e 0 *false positives*.



(a) Matriz de Confusão do *dataset* de treino



(b) Matriz de Confusão do *dataset* de test

Figura 29: Matriz de Confusão para a rede com função de ativação *tanh* e 2 unidades na camada intermédia

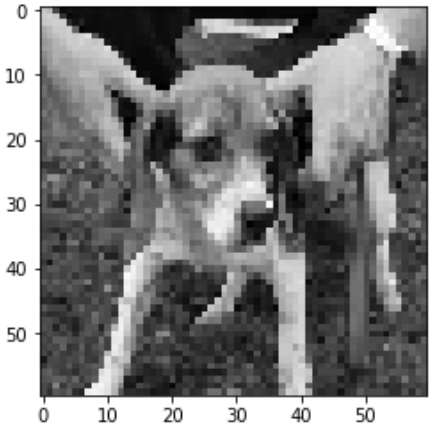
5.3 Descrição do caso de estudo número 3

Para este caso de estudo, decidiu-se utilizar um exemplo mais complexo, respetivamente, um problema de classificação binária em que o objetivo é conseguir distinguir entre gatos e cães. O conjunto de dados foi fornecido pelo *kaggle* [5], em que o conjunto é constituído por 25000 imagens com uma dimensão de (300,300,3).

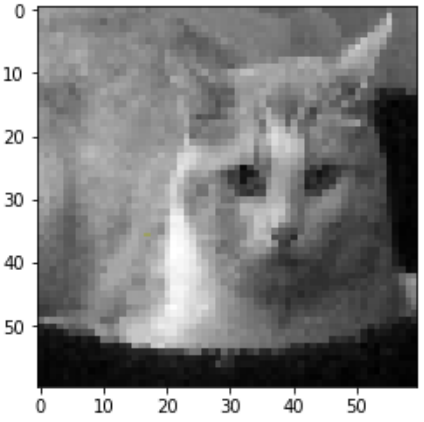
(a)

(b)

Dada a arquitetura dos modelos e complexidade das imagens, decidiu-se transformar estas para *grayscale* e reduzir o tamanho para 60, ficando com uma dimensão de (60,60,1), como ilustrado na seguinte figura:



(a)



(b)

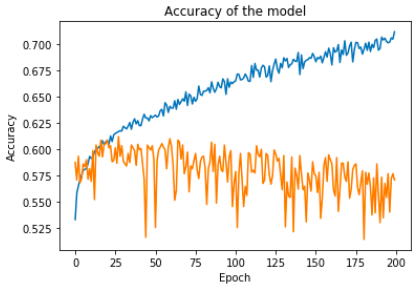
Durante esta secção utiliza-se dois modelos, uma *Shallow Neural Network* com 400 neurónios e uma *Deep Neural Network* com 3 camadas intermédias de 500,300,100 neurónios respetivamente.

Dado a complexidade do caso de estudo, esta secção não segue a metodologia descrita anteriormente, mas tenta responder à pergunta referenciada inicialmente *Do Deep Nets Really Need to be Deep?*.

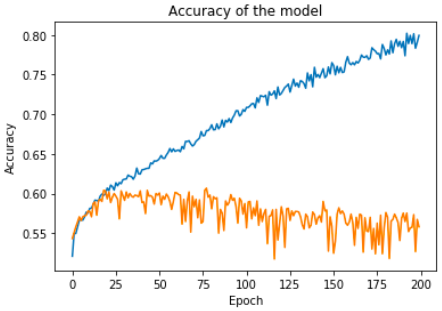
Usamos a mesma função de otimização, o mesmo número de *epochs* e a mesma taxa de aprendizagem em cada um dos modelos, respetivamente gradiente descendente, 200 epochs e uma taxa de aprendizagem fixa. Por fins computacionais, dado que cada *epoch* num *GPU* fornecido pelo *google colab* demora aproximadamente um minuto, não conseguimos aumentar muito mais o número de *epochs* nem testar várias topologias.

5.3.1 Análise de Resultados

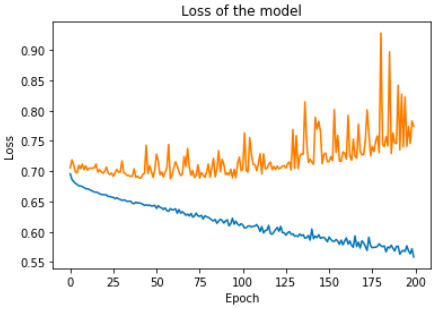
Os resultados de aprendizagem e perda estão ilustrados nas seguintes figuras, onde a linha amarela representa o caso de treino e a linha azul para o caso de teste:



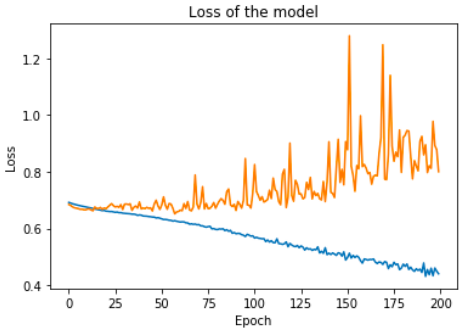
(a) *Shallow Neural Network accuracy*



(b) *Deep Neural Network accuracy*



(c) *Shallow Neural Network loss*



(d) *Deep Neural Network loss*

Pelas figuras, pode-se comparar os resultados de *accuracy* e *loss* entre uma *shallow network* e uma *deep network*. Verifica-se que a rede profunda, para as dadas circunstâncias, consegue demonstrar uma melhor capacidade de aprender *features* para os casos de treino. Isto justifica-se pelo maior número de unidades e camadas que traduz uma maior complexidade nesta rede, permitindo a geração de atributos mais complexos ao longo da hierarquia. Para os casos de teste, tanto para uma rede como para a outra, observa-se que não é conseguida uma grande capacidade de generalização, pois existe um grande diferença entre os valores conseguidos para o treino e para o teste, designando-se este fenómeno, neste caso, tendo em conta o problema e a sua complexidade, por *overfitting*[10]. Isto deve-se a uma complexidade demasiado elevada, em função do número de unidades computacionais, em relação ao *dataset*, providenciando a capacidade de mapear *inputs* a *outputs* de forma direta, impedindo a generalização para casos que a rede nunca viu durante o treino.

Do ponto de vista individual, entende-se que a rede profunda apresenta, para este caso, um maior potencial para desenvolver soluções satisfatórias (que poderiam ser conseguidas com um processo de *hyperparameter tunning*, alteração de hiperparâmetros consoante os resultados observados e complexidade do problema).

6 Discussão

Shallow Neural Networks são modelos poderosos e continuam a ser usados atualmente, o seu potencial para a superação de problemas que implicam bases de dados linearmente não separáveis foram um avanço importante no estudo de redes neuronais.

O desenvolvimento deste tipo de redes levou a que se pudesse aprofundar o que realmente está a acontecer na rede, o que acontece aos pesos e de que maneira a não linearidade, ou seja, as funções de ativação não lineares, ajudam a superar este desafio.

De seguida, entrou a possibilidade de aprofundar a rede. Isto permite um aumento da complexidade da função não linear aproximada pelo processo de treino e consequentemente apresentar melhores resultados para casos que apresentem atributos mais complexos. É necessário ter em conta que uma rede com maior número de unidades computacionais também se torna difícil de controlar no sentido de que podem ser captados e aprendidos atributos e tendências indesejáveis. Isto é um caso prático, muitas vezes, verificado no campo de NLP (*Natural Language Processing*) em relação a estereótipos e preconceitos.

Poderia-se, no entanto, considerar o desenvolvimento de uma *shallow network* se o contexto apresentasse exigências, tal como limitações no poder computacional, velocidade de inferência, facilidade de manutenção (devido a tempos de treino, tendo em conta que o desempenho de uma rede neuronal tende a piorar com o tempo devido à entropia intrínseca à natureza da informação) ou prazos de desenvolvimento a respeitar.

Este projeto centrou-se no porquê da utilização de *Shallow Neural Networks* e se realmente as redes necessitam de ser profundas.

Tendo em conta o trabalho aqui demonstrado, obtém-se uma melhor intuição da necessidade e funcionamento de *Shallow Neural Networks* e dos seus componentes individuais, tal como fundamentos de redes neuronais profundas. Obteve-se uma boa percepção em relação às potencialidades que algoritmos deste género apresentam para a resolução dos mais variados problemas. Tudo depende das circunstâncias do tipo do problema a resolver, sendo que existem diferentes tipos de arquiteturas para diferentes tipos de problemas.

Para o futuro, espera-se um maior foco no desenvolvimento de redes profundas, de forma a melhor compreender o papel que estas poderão desempenhar na criação de aplicações e obtenção de resultados que ajudem a responder a questões importantes para a sociedade dos dias de amanhã.

Referências

[1] Adeodato, P., Vasconcelos, G., Arnaud, A., Santos, R., Cunha, R. e Monteiro, D. *Neural Networks vs Logistic Regression: a Comparative Study on a Large Data Set*. 2004.

[2] Ba, L. e Caruana, R. *Do Deep Nets Really Need to be Deep?* Acedido a 15 de maio de 2020. URL: http://datascienceassn.org/sites/default/files/Do%20Deep%20Nets%20Really%20Need%20to%20be%20Deep.pdf.

[3] Bindal, A. *Measuring just Accuracy is not enough in machine learning, A better technique is required.*. Acedido a 22 de maio de 2020. URL: https://medium.com/techspace-usict/measuring-just-accuracy-is-not-enough-in-machine-learning-a-better-technique-is-required-e7199ac36856.

[4] Brownlee, J. *Understanding the dynamics of Learning rate*. Acedido a 29 de maio de 2020. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks.

[5] *Dataset "Dogs vs. Cats"*. Acedido a 28 de Maio de 2020. URL: https://www.kaggle.com/c/dogs-vs-cats.

[6] Godoy, D. *Binary Cross Entropy*. Acedido a 15 de Maio de 2020. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[7] Goodfellow, I., Bengio, Y. e Courville, A. *Deep Learning*. Acedido a 25 de Maio de 2020.

[8] Kakaraparthi, V. *Xavier and he initializer*. Acedido a 14 de Maio de 2020. URL: https://medium.com/@prateekvishnu/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528.

[9] Kawaguchi, K., Kaelbling, L. e Bengio, Y. *Generalization in Deep Learning*. 2019.

[10] Koehrsen, W. *overfitting-vs-underfitting*. Acedido a 30 de maio de 2020. URL: https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765.

[11] *Neural Network Activation Functions*. Acedido a 12 de maio de 2020. URL: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right.

[12] *Stochastic Gradient Descent*. Acedido a 12 de maio de 2020. URL: https://scikit-learn.org/stable/modules/sgd.html.

[13] *Symmetry Breaking*. Acedido a 2 de Junho de 2020. URL: https://en.wikipedia.org/wiki/Symmetry_breaking.

[14] Wang, Y., Wei, G.-Y. e Brooks, D. *Benchmarking TPU, GPU, and CPU Platforms for Deep Learning*. 2019.

[15] Yanling, Z., Bimin, D. e Zhanrong, W. *Analysis and study of perceptron to solve XOR problem*. Acedido a 1 de Junho de 2020.