Implementação de Rede Neuronal

FCUP
Inteligência Artificial (CC2006) 2018/2019
Trabalho de Grupo AK
Eduardo Morgado – 201706894 – MIERSI
Simão Cardoso – 201604595 – MIERSI
Sónia Rocha – 201704679 – MIERSI

1. Introdução:

1.1. Um Perceptron:

O objetivo principal de redes neuronais é resolver problemas de forma orgânica[1], como tal, *perceptrons* e redes neuronais são baseadas nos nossos neurónios, ou seja, tal como os nossos neurónios, *perceptrons* e redes neuronais recebem sinais/*input*, transmitem esses sinais para um núcleo de processamento que decide como tratar essa informação **[3]**. Sendo assim um *perceptron* pode ser representado pela *Figura 1*, onde (1) é a camada de nós/unidades de input para o *perceptron* (as unidades que que fornecem os dados), (2) a zona de conexões a partir das quais são passados os dados[2], ambas estas partes não números que podem ser tanto negativos como positivos **[1]** (3), a zona de processamento de dados, nesta fase é somado o produto de todos os pesos multiplicados pelo seu valor de *input* e depois somamos ainda um valor *bias* **[1,3]**, (4) é a **activation function**[3], para um único *perceptron*, a *activation function* é uma **função linear**, pelo que não consegue trabalhar/estudar problemas mais complexos, independentemente da complexidade da unidade de processamento do *perceptron*, problemas do dia-a-dia, problemas estes que são **não-lineares**, como tal esses problemas têm que ser processados por uma função não linear que consegue determinar certas caraterísticas do problema, estas funções devem também ser contínuas e deriváveis para poderem classificar os dados em intervalos de valores **[3]**.

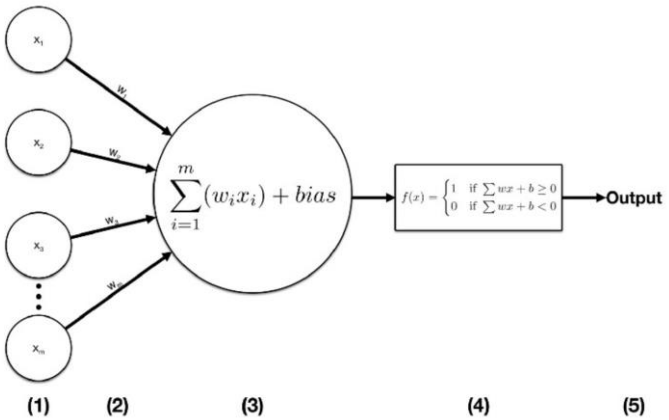


*Figura 1- Um Perceptron (imagem retirada de **[3]**)*

[1] Orgânica no sentido de tentar resolver os problemas de uma forma mais natural, da mesma forma como o ser humano aprende por tentativa-erro as redes neuronais também o fazem.
[2] Cada conexão tem um certo peso, uma dada influência para o problema, **quanto maior o peso, mais influente será a conexão [3]**.
[3] Uma função utilizada para dividir/organizar o *output* de (3) **[1,3]**.

Durante o processo de treinamento, uma vez que, tanto os *perceptrons* como as redes neuronais são métodos de **aprendizagem supervisionada**[4], após utilizar a nossa célula neuronal podemos determinar o erro desta em relação ao problema: $erro = (valor\ esperado - valor\ obtido)$, dessa forma podemos depois calibrar (modificar o valor) os pesos das conexões de forma a, num próximo treino, o resultado ser possivelmente melhor **[3]**. Dessa forma, o ajuste de pesos numa fase de treinamento é **crucial no processo de aprendizagem**.

Tal como vimos anteriormente, uma das maiores limitações para os *perceptrons* é a sua linearidade, impossibilitando o estudo de problemas mais complexos, dessa forma surge um novo método, as redes neuronais.

1.2. Um Perceptron, Dois Perceptrons…Uma Rede Neuronal:

Vimos já anteriormente que os *perceptrons* estão limitados ao tipo de problemas, à sua linearidade, como tal, é necessário encontrar uma solução para resolver este problema, a solução é tornar o sistema mais complexo, para isso adicionamos mais neurónios ao nosso sistema, formando assim uma rede neuronal, permitindo introduzir conceitos como **backpropagation**[5] e ***hidden layers***[6] **[4]** o que por sua vez permite analisar problemas não lineares, sendo possível utilizar então uma função não linear e criar gradientes de valores para o problema. A *Figura 2* apresenta uma rede neuronal com uma camada intermédia com 4 unidades.

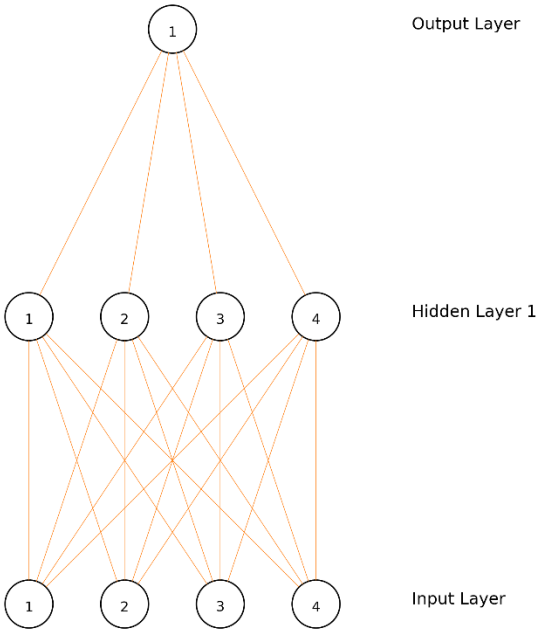


Figura 2- Uma rede neuronal de 3 camadas (input,hidden e output) com 4 unidades nas camadas de input e hidden e uma unidade para a camada de output

[4] Para que o método aprenda o modelo do problema é fornecido um conjunto de observações do problema com a respetiva solução

[5] Ajuste contínuo de todos os pesos de forma a minimizar o erro da rede neuronal.

[6] Neurónios entre camadas de *input* e *output*, permitem o estudo de problemas com caraterísticas mais complexas

2. Como é que uma Rede Neuronal Pensa:

A forma geral de funcionamento(treinamento) de uma rede neuronal são dois passos, inicialmente, após a rede estar criada, passamos os inputs por todos os nós da rede, num processo de ***feedforward***, neste processo, a camada de *output* retorna valores possíveis de solução para os *inputs* fornecidos, uma vez que a aprendizagem é supervisionada podemos comparar esses valores com os esperados e calcular um erro, este erro pode ser utilizado com utilização de uma função (iremos ver qual numa próxima secção) para calcular a modificação dos pesos na camada imediatamente anterior, o processo volta a ser repetido para a camada anterior até percorrer todas as camadas *hidden*, este processo de calcular um erro de *output* e ir modificando os pesos anteriores e ir passando o erro para trás na rede neuronal é um processo chamado de ***backpropagation***.

2.1. Feedforward:

Nesta etapa iremos considerar uma camada e a sua próxima camada, sejam as camadas I e H, para cada uma destas camadas iremos ter nós/unidades sendo cada nó I_i para $i = 1, \dots, número\ de\ nós\ na\ camada\ I(n)$ e H_j para $j = 1, \dots, número\ de\ nós\ na\ camada\ H(m)$.

Para cada nó $H_j\ j = 1(1)m$ temos as seguintes ligações com a camada I: $(I_1, H_j), (I_2, H_j), \dots, (I_n, H_j)$, cada uma dessas ligações tem associado um peso $w_{i,j}\ i = 1(1)n, j = 1(i)m$ **[4]**.

Nesta etapa, o algoritmo para cada nó H_j, calcula a soma de todos os pesos multiplicados pelos nós origem ou seja, $W^j = w_{1,j} * I_1 + \dots + w_{n,j} * I_n$ para $i = 1(1)n\ e\ j = 1(1)m$, a *Figura 3* apresenta uma representação deste processo **[4]**. Após realizar a soma ainda é necessário somar o valor de *bias* e, ao resultado final (valor escalar) fazê-lo passar por uma função de ativação não linear, a função que utilizamos foi a função *sigmoid* $\sigma(x) = \frac{1}{1+e^{-x}}$, sendo assim, $W^j = \sigma(W^j + bias)$ **[4]**.

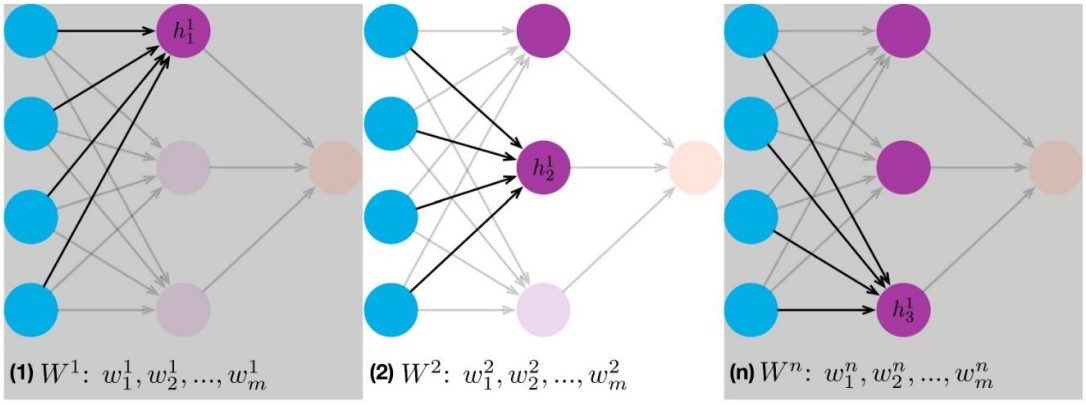


Figura 3-Processo de feedforward

No final a matriz gerada $\begin{bmatrix} W^1 \\ \dots \\ W^m \end{bmatrix}$ apresenta, para cada linha, o valor do nó H_j para $j = 1(1)m$, este processo repete-se até ser calculado o valor de todos os nós da camada *output*.

Este processo pode ser visto de um ponto algébrico como o produto de matrizes, esta interpretação foi a utilizada na realização do programa, uma vez que, todos os nós das camadas *hidden* têm uma conexão com todos os nós da camada anterior, os pesos das conexões podem ser vistos da seguinte forma:

$W^k = \begin{bmatrix} w_{1,1} & \cdots & w_{1,m} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \dots & w_{n,m} \end{bmatrix}$, onde k representa a camada a considerar, $K = H^1, \dots, H^{última\ camada\ hidden}, O(camada\ output)$, n é o número de nós na camada $k-1$ e $m\ o\ número\ máximo\ de\ nós\ na\ camada\ k$.

Se agora considerarmos os nós/valores da cama $k-1$ como $C^{k-1} = \begin{bmatrix} c_1^{k-1} \\ \vdots \\ c_n^{k-1} \end{bmatrix}$, a camada C^k, os valores do seu nó podem ser calculados por: $C^k = \sigma(W^k \cdot C^{k-1} + bias)$

2.2. Backpropagation:

Após a etapa anterior, os nós da camada *output* já têm valores, sendo assim podemos agora calcular o $erro_j = val_j - val\_esperado_j$, $j = 1(1)n$ onde n é o número de nós da camada *output* **[2,4]**.

Depois de ser calculado o erro, temos que o corrigir, para isso devemos, aos pesos das ligações que conectam o nó atual modifica-las, mas não podemos parar aí pois, apesar de corrigirmos os valores para um nó, ainda devemos corrigir para os nós das camadas anteriores, daí a utilização do algoritmo de *backpropagation* **[2,4]**.

Considerando duas camadas, H e O, sendo H a primeira camada *hidden* antes da camada *output* O, seja $E^O = \begin{bmatrix} e_1 \\ \vdots \\ e_n \end{bmatrix}$ como o erro da cama *output*, onde n é o número de nós dessa camada, consideremos α como o *learning rate*[7] para a rede neuronal, a correção da rede neuronal, baseia-se no calculo da perda/desperdício de valores de um nó e acrescentá-lo aos seus pesos diminuindo

[7] A quantidade de modificações que os pesos sofrem durante o processo de recalibração/correção de erro **[6]**, é um valor positivo pequeno, entre 0 e 1, controla a velocidade de adaptação da rede neuronal ao problema, *learning rates* mais pequenos requerem mais iterações/treinos/*epochs* da rede neuronal, pois alteram os pesos mais lentamente **[6]**. É importante referir que *learning rates* muito grande levam a soluções não ótimas pois convergem muito rapidamente, *learning rates* muito pequenos podem parar o processo **[6]**.

a perda de informação com o avanço da rede, aos valores de perda da rede neuronal chamamos de **deltas** Δ **[5]**.

Dessa forma o valor de perda de pesos e do *bias* entre a última camada *hidden* H e a camada de *output* O é calculado da seguinte forma:

$$\Delta w_{i,j} = \alpha \cdot E^O \cdot \sigma'(O) \cdot H^T$$

$$\Delta bias_j = \alpha \cdot E^O \cdot \sigma'(O)$$

Onde H^T é a matriz transposta dos valores dos nós da camada H, $i = 1(1)n$ e $j = 1(1)m$ onde n e m são respetivamente o número de nós em H e O **[5]**. Após serem calculados estes deltas, os seus valores são somados aos seus pesos correspondentes e ao *bias* para a cama j, estando os pesos entre H e O corrigidos, o processo repete-se para as camadas anteriores até os pesos entre a camada *input* e a primeira *hidden* terem sido corrigidos **[5]**.

3. Resultados:

Para esta implementação de uma rede neuronal foi utilizado como linguagem de implementação Python dada a facilidade de representação da rede. Para o programa executamos *Main.py* e selecionamos os ficheiros em *./Data* de input e de teste, para os resultados apresentados esses ficheiros foram *parity4.txt* e *parity4_predictions.txt*, as imagens que iremos apresentar pode ser encontradas na pasta *./Images* do ficheiro após a execução/criação da rede.

Para os testes o conjunto de dados de input está representado na *Figura 4* e os testes, juntamente com o resultado da rede estão representados na *Figura 5*.

Input(x)				**Expected(y)**
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
0	1	0	1	0
0	0	0	0	0
0	1	1	0	0
1	1	0	0	0
0	0	1	1	0
1	1	1	1	0
0	1	1	0	0

Figura 4-Conjunto de dados de treino

A tabela da *Figura 5*, os valores de output foram obtidos para um total de 1000 *epochs* para a rede neuronal, para cada *learning rate*

Entrada				Previsão de rede
1	0	0	0	0.98087617
0	1	0	1	0.1610888
1	1	1	1	0.1610933
0	0	0	0	0.16170989
1	0	1	1	0.70480722
0	0	1	1	0.16109541
0	1	0	1	0.16108883

Figura 5-Tabela de previsões da rede neuronal para valores de teste

As *Figuras 6* e *7* apresentam os gráficos da relação de erro médio e *epochs* e *learning rate* respetivamente.

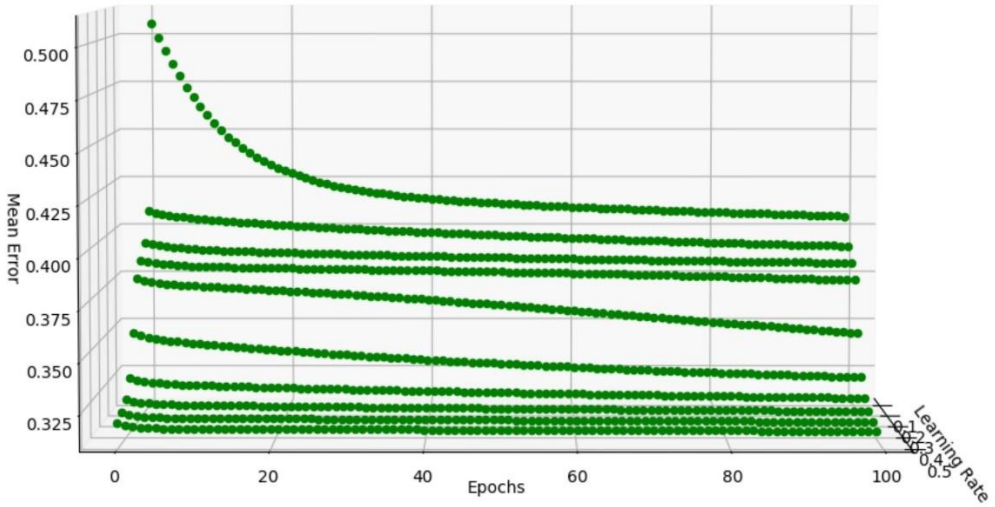


Figura 6-Relação entre erro média e epochs

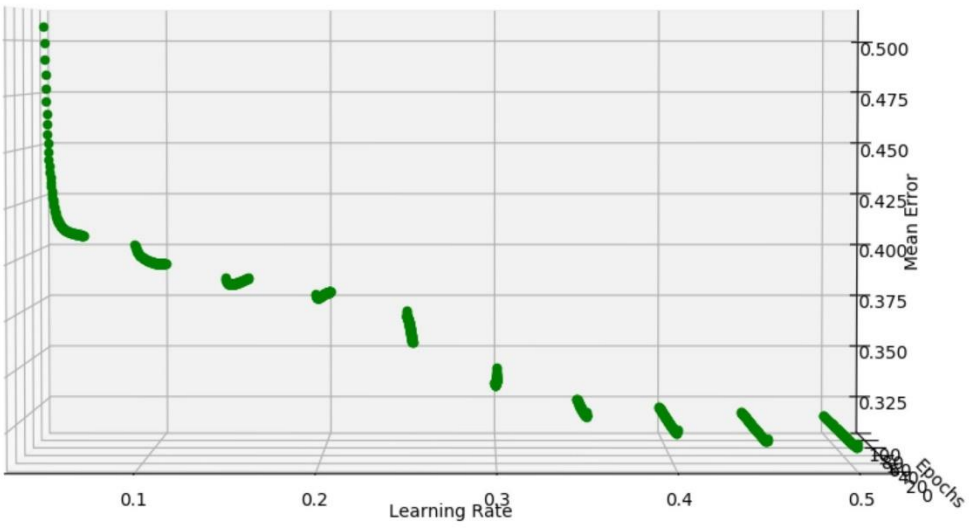
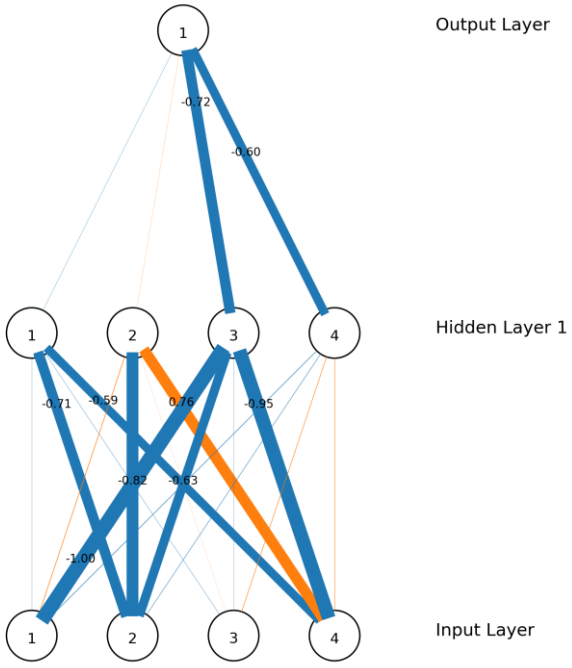


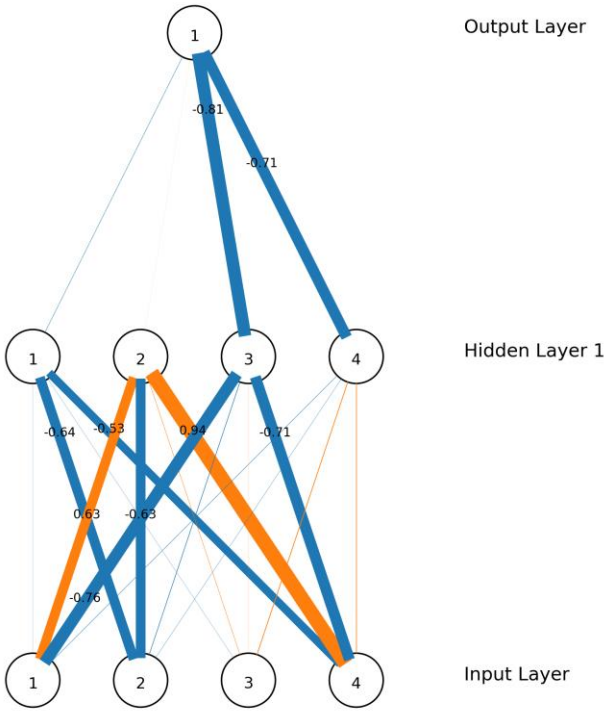
Figura 7-Relação entre erro médio e learning rate

A seguir apresentamos alguns dos gráficos da nossa rede neuronal com os pesos, por uma questão de facilitação de observação, a rede apresentada foi treinada para 10 iterações por cada *learning rate*.

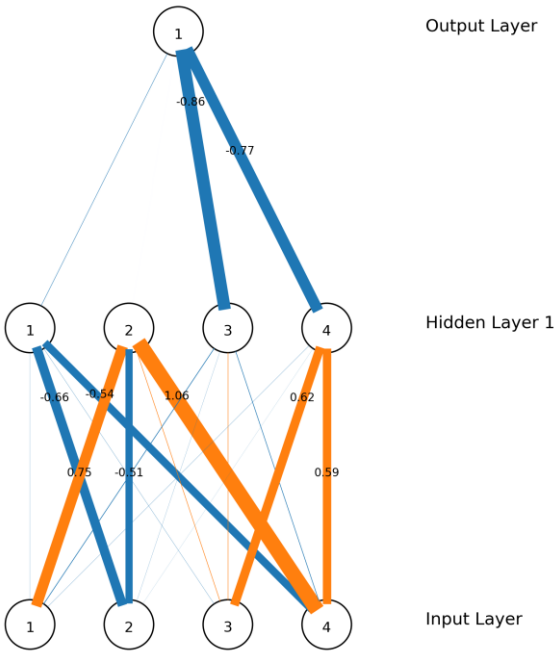
Neural Network architecture (Initial Network)



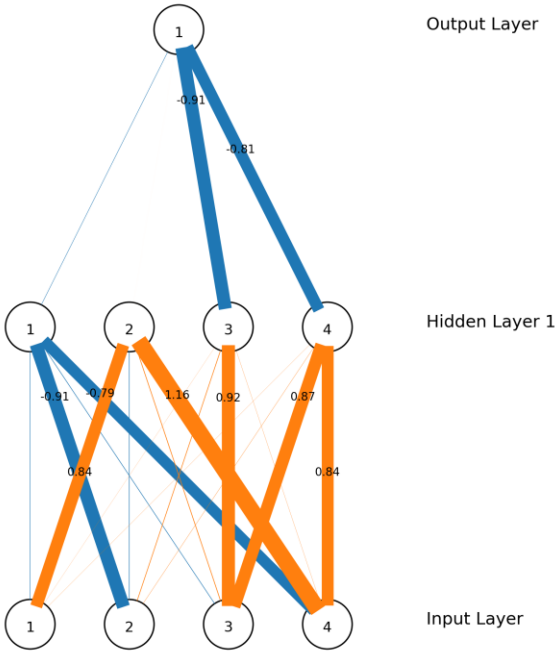
Neural Network architecture (Network learning rate 0.05)



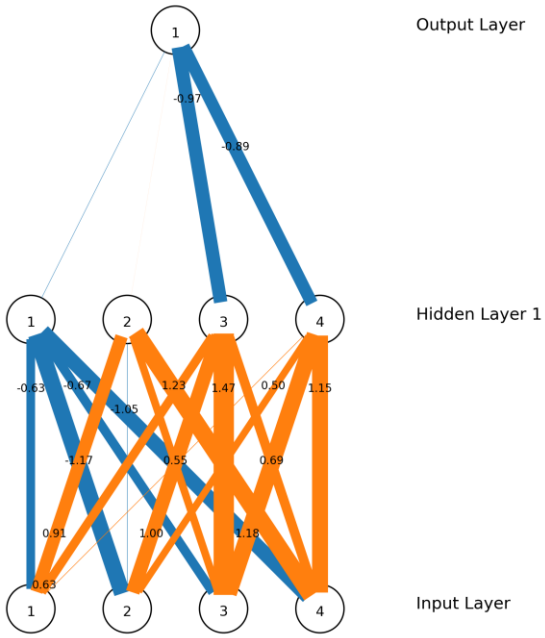
Neural Network architecture (Network learning rate 0.1)



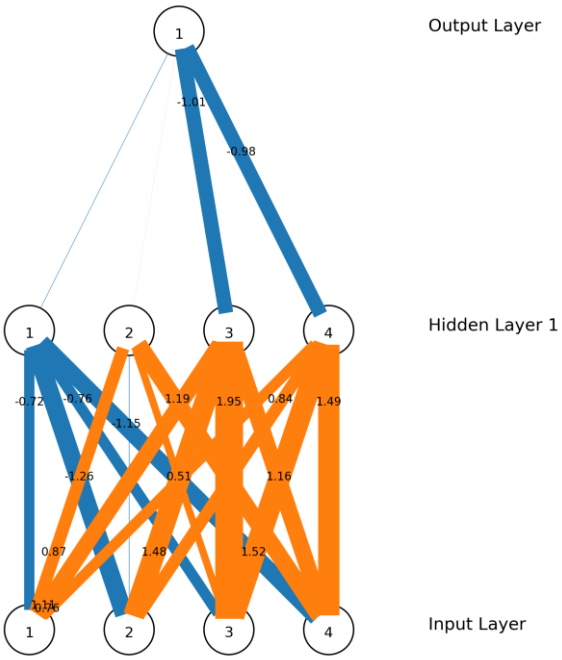
Neural Network architecture (Network learning rate 0.2)



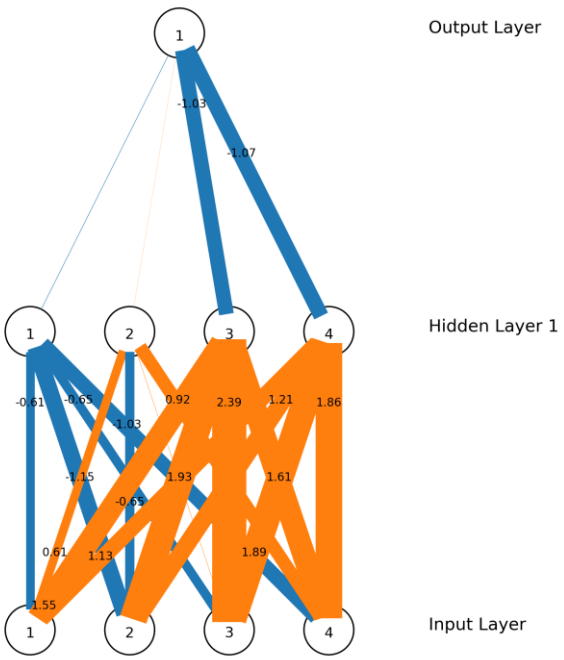
Neural Network architecture (Network learning rate 0.3)



Neural Network architecture (Network learning rate 0.4)



Neural Network architecture (Final Network)



4. Comentários Finais:

Por observação dos dados das *Figuras 6* e *7* podemos concluir que, quanto maior os *epochs* de treino, menor o erro médio da rede neuronal, o mesmo pode ser concluído para os *learning* rates, quanto maior o valor do *learning rate*, menor o erro cometido, o que por sua vez implica que a rede convirja mais rapidamente para a solução mais correta.

Por observação dos gráficos da rede neuronal, podemos observar que ao longo do processo de treino, a rede vai "incentivar" certas conexões a melhorarem de forma a conduzir a rede a melhores soluções.

Para melhorar o *output* de uma rede neuronal, podemos acrescentar mais camadas entre a *input* e *output* bem como mais nós para cada camada, podemos aumentar o conjunto de dados fornecidos para treino e ainda modificar o *learning rate*.

Nota Adicional: Este relatório, bem como o programa utilizado para obtenção dos dados, pode ser acedido no repositório da equipa, https://github.com/thejoblessducks/A-Neural-Network-That-Could.git, ou em https://github.com/eamorgado/NeuralNet.git

7. Referências:

[1] A Medium Corporation 26 de junho de 2016, Harsh Pokharna, *For Dummies-The Introduction to Neural Networks we all need!(Part1)*,consultado pela última vez a 10 de maio de 2019, <https://medium.com/technologymadeeasy/for-dummies-the-introduction-to-neural-networks-we-all-need-c50f6012d5eb>

[2] A Medium Corporation 27 de junho de 2016, Harsh Pokharna, *For Dummies-The Introduction to Neural Networks we all need!(Part2)*,consultado pela última vez a 10 de maio de 2019, <https://medium.com/technologymadeeasy/for-dummies-the-introduction-to-neural-networks-we-all-need-part-2-1218d5dc043>

[3] Towards Data Science 18 de junho de 2017, *Introducing Deep Learning and Neural Networks—Deep Learning for Rookies(1)*, Nahua Kang, consultado pela última vez a 10 de maio de 2019, <https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-for-rookies-1-bd68f9cf5883>

[4] Towards Data Science 27 de junho de 2017, *Multi-Layer Neural Networks with Sigmoid Function— Deep Learning for Rookies(2)*, Nahua Kang, consultado pela última vez a 10 de maio de 2019, <https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f>

[5] Towards Data Science 30 janeiro de 2017, *How Does Back-Propagation in Artificial Neural Networks Work?*, Anas AI-Masri, consultado pela última vez a 11 de maio de 2019, <https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7>

[6] Machine Learning Mastery 25 de janeiro de 2019, *Understand the Impact of Learning Rate on Model Performance With Deep Learning Neural Networks*, Jason Brownlee, consultado pela última vez a 12 de maio de 2019, <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>