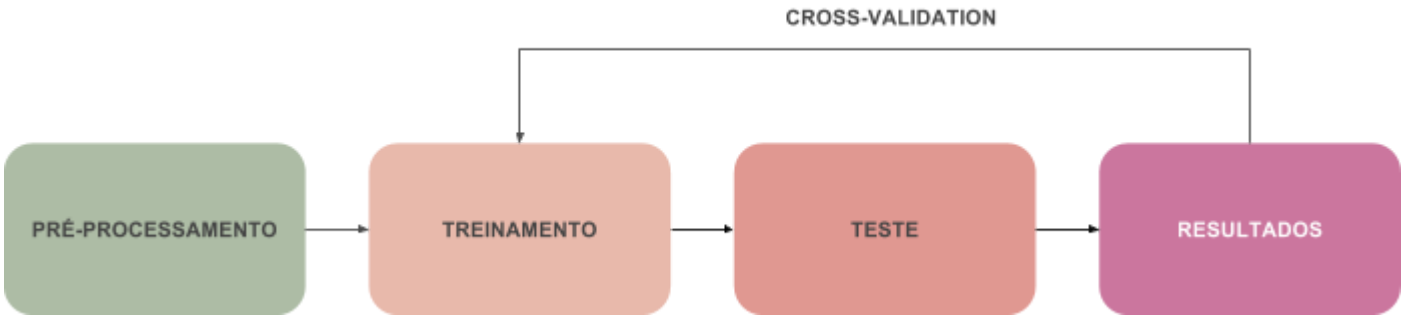
ADO #02 - Multilayer Perceptron (MLP)

VERSÃO 01 - 21/03/2017

Neste trabalho, criaremos a nossa primeira rede neural denominada Multilayer Perceptron (MLP). A MLP é uma rede neural artificial supervisionada e, por isso, necessita das classes corretas de cada instância do conjunto de dados para se ajustar. A fase de ajuste também é chamada de **treinamento**.

Para o treinamento da nossa rede MLP utilizaremos o algoritmo **Backpropagation**. O *Cross-Validation* pode ser aplicado na rede como forma de avaliá-la. A figura abaixo mostra o processo como um todo.



1. OBJETIVO DESTE TRABALHO

Este trabalho como objetivo estudar e avaliar o classificador supervisionado Multilayer Perceptron como parte da Atividade Orientada.

1.1 CONJUNTOS DE DADOS

Na ADO #01 você e sua equipe fizeram a normalização (escala de 0 a 1) dos seguintes conjuntos de dados do UCI Machine Learning Repository:

- Iris - 3 classes
- Adult - 2 classes
- Wine - 3 classes
- Breast Cancer Wisconsin (*o campo Id pode ser removido*) - 2 classes
- Wine Quality - 11 classes
- Abalone - 29 classes

Para evitar um trabalho adicional, vamos usar esses conjuntos de dados neste trabalho.

1.2 PRÉ-PROCESSAMENTO

Você deverá utilizar seu conjunto de dados já pré-processado (isto é, normalizado).

Antes de utilizar os dados em seu classificador, é importante **EMBARALHAR** o conjunto todo. Caso contrário, pode ser que instâncias de uma mesma classe fiquem agrupadas e pode atrapalhar o treinamento.

Esse último detalhe não foi dito na ADO #01, porém uma função **deve ser criada para tal.**

2. A REDE NEURAL MULTILAYER PERCEPTRON (MLP)

A figura abaixo representa uma rede neural MLP com a camada de entrada, 03 camadas ocultas e a camada de saída.

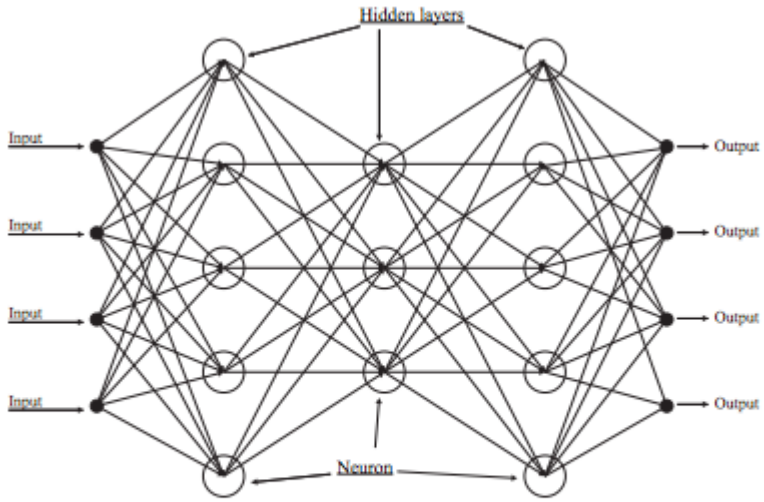


Figure 1: Description of the Multilayer perceptron. The MLP includes 13 neurons, 3 hidden layers and 5 inputs and 5 outputs.

Fonte: http://web-ext.u-aizu.ac.jp/labs/is-se/conference_proceedings/iwait-15/41.pdf

Nessa última figura não está ilustrada, mas para cada camada de neurônios na rede colocaremos uma dimensão adicional, chamada de *bias* ou viés. O aumento de dimensão facilita o cálculo de funções discriminantes lineares, como visto nos slides de "Funções Discriminantes Lineares".

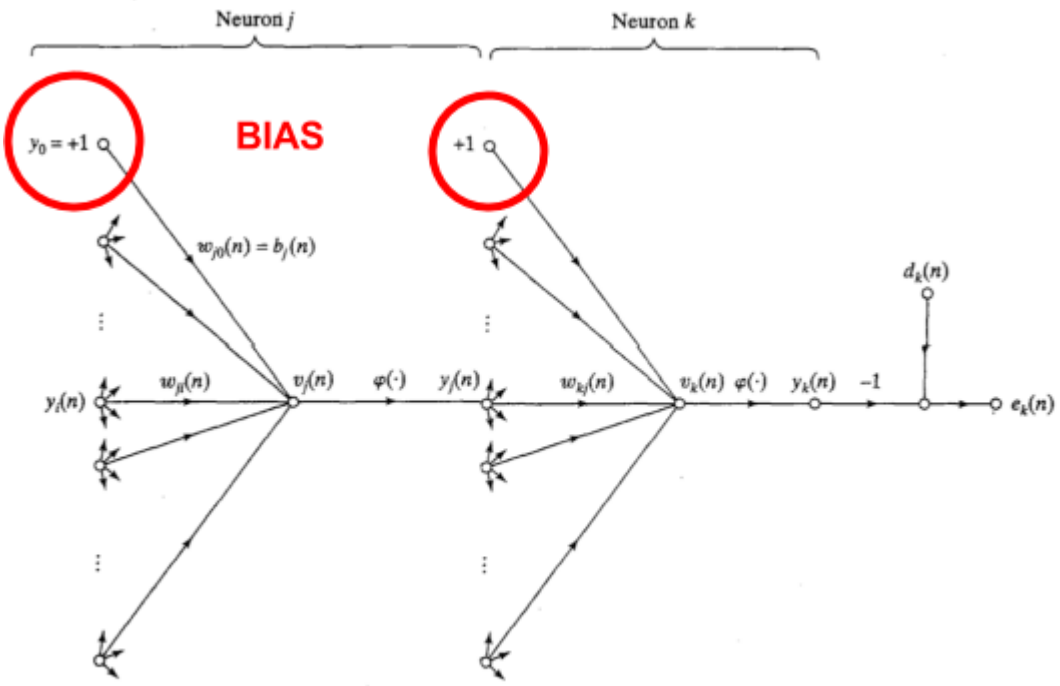


FIGURE 4.4 Signal-flow graph highlighting the details of output neuron *k* connected to hidden neuron *j*.

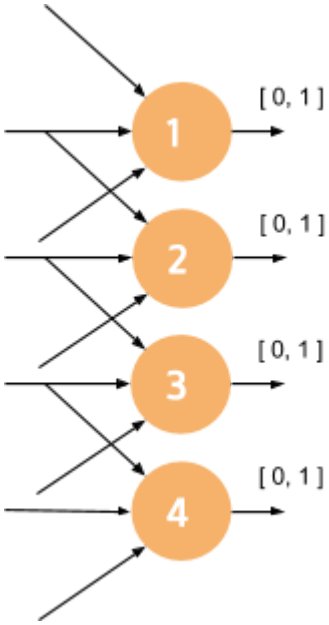
2.1 QUANTIDADE DE NEURÔNIOS NA CAMADA DE SAÍDA

Dependendo do problema, poderão haver diferentes quantidades neurônios na camada de saída. Neste trabalho iremos considerar a abordagem de **01 neurônio para cada classe**. No apêndice, você poderá ver a abordagem de considerar apenas 01 neurônio na camada de saída, mas não a utilizaremos neste trabalho.

Para explicar esse o funcionamento, vamos supor um exemplo com 5 instâncias, 3 atributos (todos normalizados) e 4 classes.

INSTÂNCIA	ATRIBUTO 1	ATRIBUTO 2	ATRIBUTO 3	ATRIBUTO 4	CLASSE
1	0,01	0,32	0,78	0,46	1
2	0,83	0,91	0,96	0,56	2
3	0,13	0,64	0,07	0,55	3
4	0,53	0,73	0,99	0,16	4
5	0,33	0,27	0,08	0,14	2

Como temos 04 classes nesse conjunto, devemos criar uma rede MLP com **4 neurônios** de saída, como mostrado na figura abaixo.



Agora precisamos **garantir** que:

- se a instância for da classe 1, somente o **neurônio 1** seja ativado (ou seja, tenha valor 1), e os outros neurônios tenham valores iguais a **zero**.
- se a instância for da classe 2, somente o neurônio 2 seja ativado (ou seja, tenha valor 1), e os outros neurônios tenham valores iguais a **zero**.
- se a instância for da classe 3, somente o neurônio 3 seja ativado (ou seja, tenha valor 1), e os outros neurônios tenham valores iguais a **zero**.
- se a instância for da classe 4, somente o neurônio 4 seja ativado (ou seja, tenha valor 1), e os outros neurônios tenham valores iguais a **zero**.

Por isso, precisamos modificar o conjunto de dados para se adequar às 4 saídas esperadas de cada neurônio.

INSTÂNCIA	ATRIBUTO 1	ATRIBUTO 2	ATRIBUTO 3	ATRIBUTO 4	SAÍDA NEURÔNIO 1	SAÍDA NEURÔNIO 2	SAÍDA NEURÔNIO 3	SAÍDA NEURÔNIO 4
1	0,01	0,32	0,78	0,46	1	0	0	0
2	0,83	0,91	0,96	0,56	0	1	0	0
3	0,13	0,64	0,07	0,55	0	0	1	0
4	0,53	0,73	0,99	0,16	0	0	0	1
5	0,33	0,27	0,08	0,14	0	1	0	0

Com essa adaptação, ao término da fase de Forward, é possível calcular o erro *e(n)* de cada neurônio, comparando essas saídas esperadas *d(n)* com a saída do neurônio *y(n).*

2.2 SIMULAÇÃO DO ALGORITMO BACKPROPAGATION

ETAPA FORWARD - INICIALIZAÇÃO

1. Ajustar as instâncias de entrada como vistos no item 2.1.
2. Inicializar os todos pesos *w* da rede com valores reais aleatórios no intervalo `[ -1, 1 ]` (sim, os pesos podem ser negativos);
3. Inicializar o contador de iterações `count = 1`.

ETAPA FORWARD - EXECUÇÃO

1. Considerar a próxima instância x do conjunto de treinamento.
2. Adicionar uma dimensão a mais na primeira camada oculta ($x_0 = 1$). Chamamos isso de bias, ou viés.
3. Calcular $v_j(n)$ para todo neurônio j da primeira camada oculta.
4. Calcular a $y_j(n) = \varphi(v_j(n))$ para todo neurônio *j* da primeira camada oculta. Lembre-se que a função $\varphi(x)$ é uma função sigmóide.
5. Adicionar uma dimensão a mais na entrada da L-ésima camada oculta ($x^L_0 = 1$);
6. Calcular $v^L_k(n)$ para todo neurônio *k* da L-ésima camada oculta.
7. Calcular a $y^L_k(n) = \varphi(v^L_k(n))$ para todo neurônio *k* da L-ésima camada oculta.
8. Repetir os passos 5 a 7 para todas as camadas ocultas.

9. Adicionar uma dimensão a mais na entrada da camada de saída valendo 01;
10. Calcular $v_m(n)$ para todo neurônio m da camada de saída.
11. Calcular a $y_m(n) = \varphi(v_m(n))$ para todo neurônio m da camada de saída.
12. Calcular o erro $e_m(n) = d_m(n) - y_m(n)$ todo neurônio m da camada de saída, onde $d_m(n)$ representa a saída esperada (ou classe) da instância.
13. Calcular o erro quadrático de todos os C neurônios da camada de saída:

$$\mathscr{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

14. Se essa for a última instância do conjunto de treinamento, adicionar 01 ao contador de iterações `count` e calcular o erro quadrático médio:

$$\mathscr{E}_{\text{av}} = \frac{1}{N} \sum_{n=1}^{N} \mathscr{E}(n)$$

    a. Se a ***razão*** entre esse erro quadrático médio e o seu valor anterior (ou seja, taxa de mudança - rate of change) for menor do que um limiar θ (constante definida no início do algoritmo), então o deve-se parar o algoritmo.
    b. Caso contrário, deve-se ir continuar a sua execução, ou seja, executar a etapa Backward dessa última instância e depois voltar para a primeira instância de treinamento.

ETAPA BACKWARD

A partir de agora, o processamento é realizado da camada de saída em **direção à camada de entrada**.

15. Deve-se calcular o valor de $\delta_j(n)$ para todos os j neurônios da camada de saída.

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$$

16. Atualizar os pesos $w_{ji}^{(l)}$ entre a camada de saída e a última camada oculta. Na fórmula abaixo, n representa o tempo, j representa os neurônios da camada de saída (L) e i representa os neurônios da camada oculta (L-1).

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n)$$

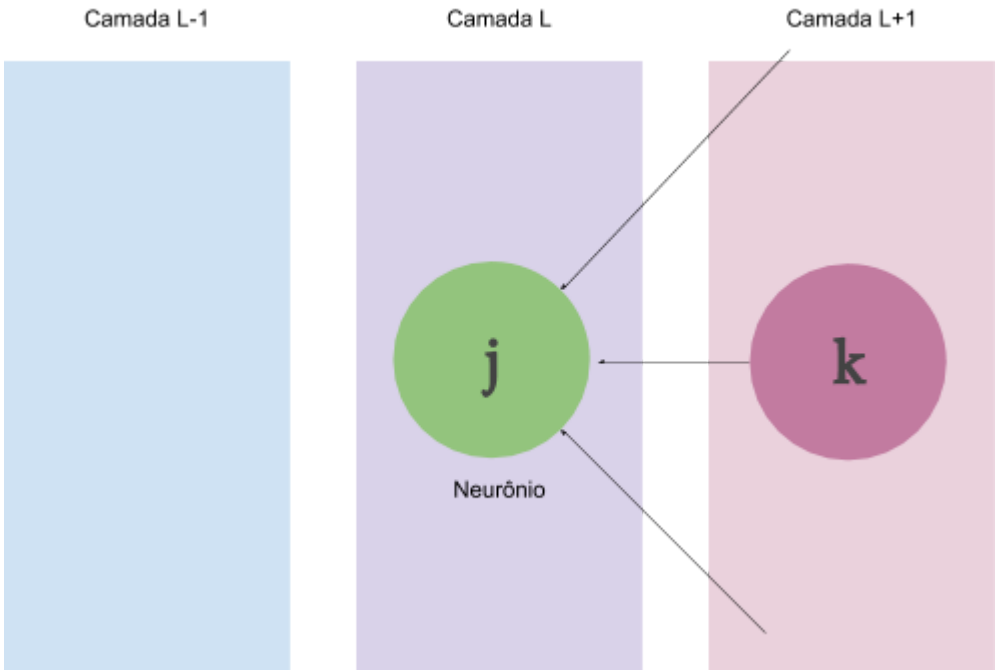
17. Calcular o valor de $\delta_j(n)$ para todos os j neurônios da *L-ésima camada oculta* da maneira abaixo. Nessa fórmula, k representa os neurônios da camada à direita da rede.

    Por exemplo, se estamos calculando $\delta_j(n)$ da última camada oculta, então k se refere aos

neurônios da **camada de saída**. Se não for a última camada oculta, então *k* se referiria aos neurônios da camada oculta à direita.

$$\delta_j^{(l)}(n) = \varphi_j'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n)$$

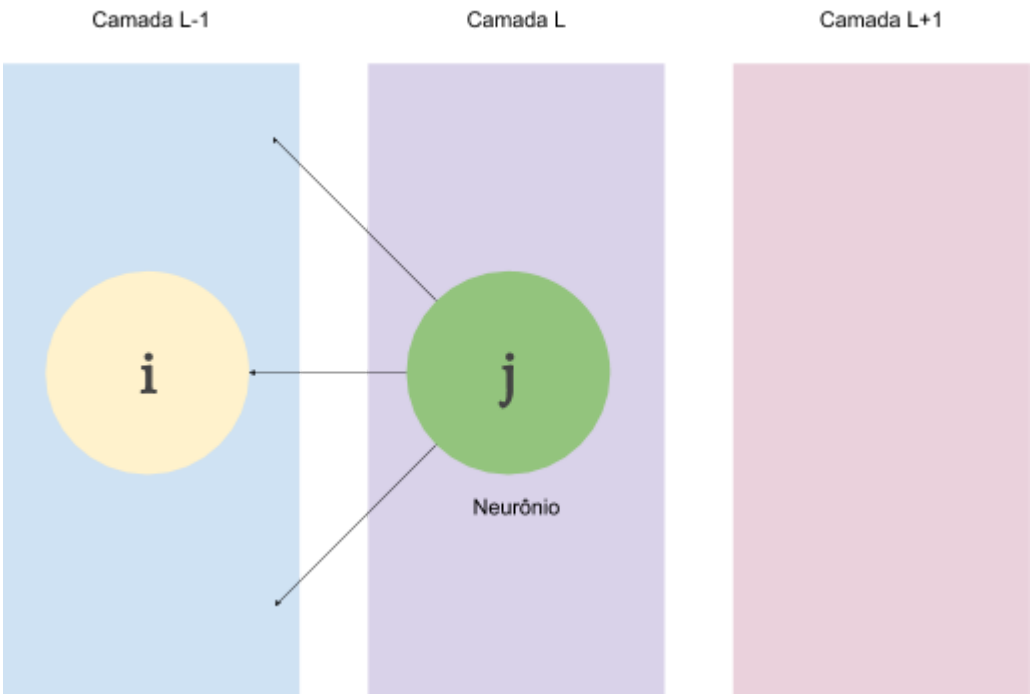
Essa figura ilustra a localização do neurônio da camada oculta L.



18. Atualizar os pesos $w_{ji}^{(l)}$ entre a *L*-ésima oculta e a *(L-1)*-ésima camada oculta. Na fórmula abaixo, *n* representa o tempo, *j* representa os neurônios da *L*-ésima camada oculta e *i* representa os neurônios da (L-1)-ésima camada oculta.
    Por exemplo, se *L* for a última camada oculta, então *(L-1)* é a penúltima camada oculta. Porém, se *L* for a primeira camada oculta, então (L-1) corresponde à entrada X (ou camada de entrada)

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n) y_i^{(l-1)}(n)$$

A figura abaixo ilustra a localização dos neurônios nas camadas L e L-1.



19. Repetir os passos 17 e 18 para todas as camadas ocultas.

20. Ir para o passo 01 da etapa FORWARD - EXECUÇÃO.

3. VALIDAÇÃO CRUZADA

A Validação Cruzada (ou também conhecido como *Cross Validation*) é uma técnica para avaliação de classificadores. Para esse trabalho, considere que:

- Utilizaremos o *10*-fold Cross Validation, que consiste em dividir a amostra em 10 partes de tamanhos (aproximadamente) iguais.
- Repetir 10 rodadas de treinamento, deixando alternadamente uma das partes para teste em cada rodada.

4. EXECUÇÕES DO ALGORITMO

Para avaliarmos o desempenho do algoritmo MLP sobre os conjuntos de dados, iremos modificar os valores de seus parâmetros a fim de encontrar os mais adequados aos dados em questão.

4.1 PARÂMETROS DA REDE NEURAL

Abaixo, estão todos os parâmetros existentes em nossa rede MLP.

Parâmetro	Descrição
a	Parâmetro da função sigmóide.
η	Taxa de aprendizado para atualização dos pesos. Em alguns artigos, usa-se uma função de decaimento com relação ao número de iterações.
α	*Momentum* (constante utilizada para considerar os valores anteriores dos pesos na sua atualização)
θ	Limiar constante que define o critério de parada. Se a taxa de mudança no erro quadrático médio a cada iteração for menor do que θ, o treinamento deve ser finalizado.
M	Quantidade de camadas ocultas
Q_m	Quantidade de neurônios na camada oculta *m* tal que $1 \leq m \leq M$.

4.2 CONSTANTES DA REDE NEURAL

A partir dos parâmetros mencionados, iremos executar nosso algoritmo. Para todas as execuções, os parâmetros listados abaixo devem ser **constantes**. Isso por dois motivos: (1) a literatura os define como padrões e (2) reduzir a quantidade de testes em nosso trabalho.

Todavia, em trabalhos acadêmicos para publicações de artigos em congressos recomenda-se que seja encontrada a melhor taxa por meio de testes em cima do conjunto de dados adotado.

Parâmetro	Valor
a	1.0
η	0.01
θ	0.01

4.3 QUAL A MELHOR TOPOLOGIA DA REDE?

O objetivo aqui é descobrir qual a **quantidade de camadas ocultas** e **quantidade de neurônios por camada** possui a melhor taxa de acerto utilizando o Cross-Validation.

Para cada conjunto de dados, a rede neural será treinada e testada com **diferentes configurações** na quantidade de camadas ocultas e quantidade de neurônios em cada uma delas. Portanto, o ideal é que esses itens sejam parametrizáveis em seu algoritmo.

Abaixo, segue uma tabela com os valores mínimos e máximos de cada parâmetro.

Parâmetro	Descrição	Valor mínimo	Valor máximo
M	Quantidade de camadas ocultas	1	3
Q_m	Quantidade de neurônios na camada oculta *m*	0	**2 *MAIOR(INPUT, OUTPUT)** Onde: INPUT - quantidade de neurônios na camada de entrada. OUTPUT - quantidade de neurônios na camada de saída. MAIOR - função que calcula o maior valor entre os dois parâmetros

Você e sua equipe deverão testar os conjuntos de dados com, **pelo menos, 10 configurações** e apresentar os resultados (veja como apresentar os resultados na seção "RESULTADOS").

4.4 QUAL O EFEITO DO MOMENTUM?

O momentum α é uma constante utilizada para considerar os valores anteriores dos pesos na sua atualização. Em nosso experimento, trabalharemos com $\alpha = 0$ para a maioria dos casos.

No entanto, para estudar o efeito do Momentum sobre o treinamento dos dados, faremos o seguinte processo:

1. Para um determinado conjunto de dados, encontrar a topologia de rede neural (subseção 4.3) com o melhor percentual de acertos nos testes do Cross Validation.
2. Testar essa topologia com diferentes valores de α, agora demonstrando **quantas iterações foram necessárias para atingir o limiar θ**. Os seguintes valores de α devem ser utilizados:
    - $\alpha = 0.0$;
    - $\alpha = 0.25$;
    - $\alpha = 0.50$;
    - $\alpha = 0.75$;
    - $\alpha = 0.90$;

4. RESULTADOS

Essa seção descreve como os resultados deverão ser apresentados em um relatório para ser entregue no Blackboard.

4.1. Comparação dos modelos de MLP

Durante a execução das topologias de MLPs, uma tabela semelhante à mostrada abaixo deve ser apresentada.

TABELA 1 - Comparação de número de camadas ocultas e neurônios de uma MLP.

Number of neurons	Correctly Classified Instances (%)	Mean absolute error	Root mean squared error
(3, 3, 0)	75.3906	0.3041	0.4192
(5, 5, 0)	75.1302	0.2983	0.4281
(3, 3, 3)	76.1719	0.4463	0.3078
(3, 3, 5)	75.7813	0.3092	0.4162
(3, 5, 5)	76.0417	0.3129	0.4192
(5, 5, 5)	73.8281	0.3125	0.4302
(5, 5, 10)	75.0000	0.3051	0.4220
(10, 10, 10)	74.4792	0.3036	0.4355
(20, 20, 20)	74.4792	0.2978	0.4274

Na tabela constam as seguintes informações:

- Número de neurônios: informa a quantidade de neurônios utilizada em cada camada no seguinte formato (X, Y, Z), onde X, Y, Z representam respectivamente a primeira, segunda e terceira camada oculta da rede. Se o valor de algum deles for igual a zero, então significa que aquele camada não foi utilizada.
- **% Instâncias corretamente classificadas**: porcentagem de instâncias classificadas corretamente pela rede neural durante a fase de testes do Cross-Validation (feito depois do treinamento da rede);
- Erro absoluto médio: é o erro calculado nos testes do Cross-Validation e compara a saída esperada com a saída do neurônio. Sua fórmula pode ser vista aqui: https://en.wikipedia.org/wiki/Mean_absolute_error
- Raiz do erro quadrático médio: representa o desvio padrão simples das diferenças entre os valores e suas observações. Sua fórmula pode ser vista aqui: https://en.wikipedia.org/wiki/Root-mean-square_deviation

Além dessas 4 informações, você deverá apresentar a coluna:

- **Número de iterações**: média da quantidade de iterações necessárias até a sua convergência, ou seja, até atingir o limiar θ. Lembre-se que 01 iteração é quando todos os elementos do conjunto de teste são apresentados à rede.

Como gerar essa tabela? Os resultados dessa tabela são compostos a partir dos testes do Cross-Validation.

Por exemplo, para a configuração (3, 3, 0) você deverá executar o Cross-Validation dividindo o conjunto de dados em 10 partes iguais [p_1, p_2, …, p_{10}]. Logo teremos 10 execuções da seguinte maneira:

EXECUÇÃO 01 (E-01):

1. Separar p_1 e treinar a rede com o restante [p_2 até p_{10}].
2. Testar a rede treinada com p_1. Aqui você obterá a quantidade de instâncias corretamente classificadas, o erro absoluto médio, raiz do erro quadrático médio e o número de iterações necessárias para convergência do treinamento.

EXECUÇÃO 02 (E-02):

3. Separar p_2 e treinar a rede com o restante [p_1, p_3, …, p_{10}].
4. Testar a rede treinada com p_2. Aqui você obterá a quantidade de instâncias corretamente classificadas, o erro absoluto médio, raiz do erro quadrático médio e o número de iterações necessárias para convergência do treinamento.

Fazer o mesmo para as execuções 03 até 10.

Ao final, você teremos na tabela:

- % Instâncias corretamente classificadas: somar todas as quantidades de instâncias corretamente classificadas de E-01 até E-10 e dividir pela quantidade total de instâncias testadas.
- Erro absoluto médio: somar todos os erros absolutos médios obtidos de E-01 até E-10 e dividir por 10.
- Raiz do erro quadrático médio: somar todas os raízes do erro quadrático médio obtidos de E-01 até E-10 e dividir por 10.
- **Número de iterações**: média aritmética do número de iterações obtidos entre E-01 até E-10.

4.2. Qual melhor Momentum?

A partir do melhor resultado obtido em 4.1, usar a topologia para preencher a seguinte tabela abaixo:

Valor de α	Número de iterações (média das 10 execuções Cross Validation)
0.0	
0.25	
0.50	
0.75	
0.90	

Veja que aqui as 10 execuções do Cross-Validation serão feitas, porém somente a etapa de treinamento será utilizada nesse resultado (até a rede convergir).

4.3. Matriz de confusão

Aproveitando a execução do item 4.2, você deverá montar a matriz de confusão multi-nível para a melhor topologia encontrada em 4.1.

Caso não se lembre, consiste em uma matriz N x N, sendo N a quantidade de classes, que apresenta a quantidade de elementos X classificados como Y, sendo X e Y duas classes iguais ou diferentes.

Um exemplo da matriz de confusão aparece abaixo para o conjunto de dados Iris. As linhas representam as classes corretas (esperadas) e as colunas são as que seu modelo gerou ou respondeu.

Total		Predicted		
		Iris-setosa	*Iris-versicolor*	*Iris-virginica*
Actual	*Iris-setosa*	12	1	1
	Iris-versicolor	0	16	0
	Iris-virginica	0	1	16

Fonte: https://docs.wso2.com/display/ML100/Model+Evaluation+Measures

Então teremos 01 matriz de confusão para cada conjunto de dados.

4.4. Formato do relatório

Para cada conjunto de dados devem haver:

1. Nome do conjunto de dados
    a. Comparação dos modelos de MLP
    b. Tabela de comparação do Momentum
    c. Matriz de confusão
    d. Conclusões da equipe sobre os resultados desse conjunto de dados

VÍDEO EXPLICATIVO

Você deverá elaborar um vídeo da tela explicando:

1. **Algoritmo:** você deverá mostrar como está estruturado e como funciona cada parte do seu código-fonte.

2. **Execução:** você deverá escolher um conjunto de dados (por exemplo, IRIS) e uma versão (por exemplo, a rede neural com 2 camadas ocultas e 5 neurônios em cada camada) e mostrar a sua execução, explicando o que está acontecendo.
   Não precisa mostrar a execução de outro conjunto e de outra versão do MLP pois o vídeo ficará muito extenso.

Nessa parte, você deverá utilizar um software gravador de tela. Os sistemas Windows 10 e macOS possuem softwares gravadores de tela nativos.

ENTREGA E ENVIO NO BLACKBOARD

- O projeto pode ser feito em equipes de 01 a 03 integrantes.
- Os arquivos pré-processados utilizados devem ser compactados em enviados no Blackboard.
- Os código-fontes devem ser compactados em enviados no Blackboard.
- O vídeo pode ficar com um tamanho um tanto grande. Por isso, você pode fazer o upload no YouTube como "não listado" e enviar o link via Blackboard.
- O relatório deve ser entregue em PDF (preferencialmente feito em LaTeX).

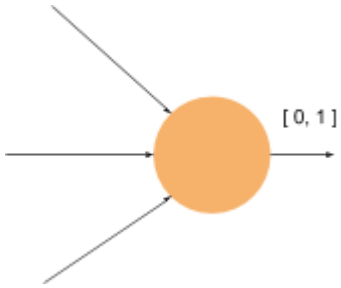
As dúvidas podem ser enviadas no e-mail willian.yhonda@sp.senac.br

BOM TRABALHO!

APÊNDICE

1. CONSIDERAR APENAS 01 NEURÔNIO NA CAMADA DE SAÍDA

Uma outra abordagem para lidar com a saída da rede seria considerarmos apenas 1 neurônio na camada de saída. Lembre-se que, devido à função de ativação $\varphi(x)$, a saída da rede será um valor entre 0 e 1. A figura abaixo ilustra esse neurônio.



Em casos onde há apenas duas classes, o resultado seria 0 para uma classe e 1 para outra classe.

Em problemas multiclasses, teremos que particionar o intervalo [0, 1] na quantidade de classes existentes. Em um problema de, por exemplo, 03 classes, as saídas seriam 0.0, 0.5 e 1.0 respectivamente para as classes 01, 02 e 03.

Como outro exemplo, vamos considerar o conjunto abaixo com 04 classes.

INSTÂNCIA	ATRIBUTO 1	ATRIBUTO 2	ATRIBUTO 3	ATRIBUTO 4	CLASSE
1	0,01	0,32	0,78	0,46	1
2	0,83	0,91	0,96	0,56	2
3	0,13	0,64	0,07	0,55	3
4	0,53	0,73	0,99	0,16	4
5	0,33	0,27	0,08	0,14	2

Adaptando as classes para as saídas esperadas do neurônio, teríamos os valores 0.0, 0.33, 0.66, 1.0 para respectivamente as classes 1, 2, 3 e 4. Logo, o conjunto seria modificado para os valores da tabela mostrada abaixo.

INSTÂNCIA	ATRIBUTO 1	ATRIBUTO 2	ATRIBUTO 3	ATRIBUTO 4	CLASSE

1	0,01	0,32	0,78	0,46	0.0
2	0,83	0,91	0,96	0,56	0.33
3	0,13	0,64	0,07	0,55	0.66
4	0,53	0,73	0,99	0,16	1.0
5	0,33	0,27	0,08	0,14	0.33

Somente durante a fase de testes, onde você gostaria de avaliar o desempenho da rede, o ideal seria aplicar $y_k(n) = ROUND(\varphi(v_k(n)))$. A função matemática ROUND arredonda um número decimal para o valor de classe mais próximo. Portanto, uma saída 0.11 de $\varphi(v_k(n))$ seria arredondada para 0.0.

Essa abordagem começa a não ser interessante quando temos um grande conjunto de classes, uma vez que temos que