**Relatório Individual Redes Neurais**

**Aluno: Clésio de Araújo Gonçalves**

O XOR foi o problema inicial utilizado na implementação da rede neural multicamada, com o intuito de entender os conceitos e os procedimentos para realizar o atualização dos pesos durante o treinamento da rede neural utilizando apenas funções do numpy.

Para aplicação de redes neurais em uma base de dados, foi escolhida a base **Breast Cancer Wisconsin (Diagnostic) Data Set** (disponível em: <https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>). Essa base de dados tem o objetivo de **prever se a paciente possui ou não câncer de mama baseado em alguns atributos**. A base de dados possui 30 atributos previsores como entrada e 569 instâncias no total. O target (alvo) possui valor 0 para tumor de câncer maligno e 1 para tumor benigno.

**Atributos previsores da base de dados (em inglês)**: [' radius\_mean', ' texture\_mean', 'perimeter\_mean', 'area\_mean', 'smoothness\_mean', 'compactness\_mean', 'concavity\_mean', 'concave\_points\_mean', ' symmetry\_mean', ' fractal\_dimension\_mean', 'radius\_se', ' texture\_se', ' perimeter\_se', ' area\_se', ' smoothness\_se', 'compactness\_se', 'concavity\_se', ' concave\_points\_se', ' symmetry\_se', ' fractal\_dimension\_se', 'radius\_worst', 'texture\_worst', ' perimeter\_worst', ' area\_worst', ' smoothness\_worst', ' compactness\_worst', 'concavity\_worst', 'concave\_points\_worst', 'symmetry\_worst', ' fractal\_dimension\_worst']

Duas implementações de redes neurais foram aplicadas na base de dados:

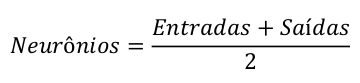
1. **implementação do algoritmo backpropagation com funções do numpy**
2. **implementação do modelo sequencial em camadas Densas do keras**

Essas duas implementações serão abordadas neste relatório.

1. **Implementação do algoritmo backpropagation com funções do numpy (breast-cancer-numpy.ipynb)**

O algoritmo implementado em python possui **variáveis de configuração** que podem ser facilmente alteradas, como: quantidade de épocas, taxa de aprendizagem, limiar do erro médio quadrado, quantidade de neurônios nas camadas de entrada, saída e camadas ocultas. Além da variável momento (utilizada na atualização dos pesos) e que ajuda o algoritmo de rede neural a encontrar o mínimo global mais rápido.

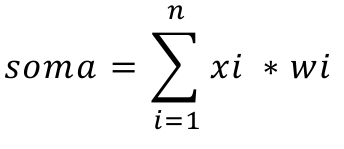
**Para a estrutura da rede neural**, recomenda-se utilizar 30 neurônios na camada de entrada (pois a base possui 30 atributos previstos) e 1 neurônio na camada de saída, pois o target possui apenas duas classes (0 e 1), ou seja, trata-se de um problema de classificação binária. Por padrão, a rede neural utiliza duas camadas ocultas. A inicialização da quantidade de neurônios das camadas ocultas, utilizou a seguinte fórmula:



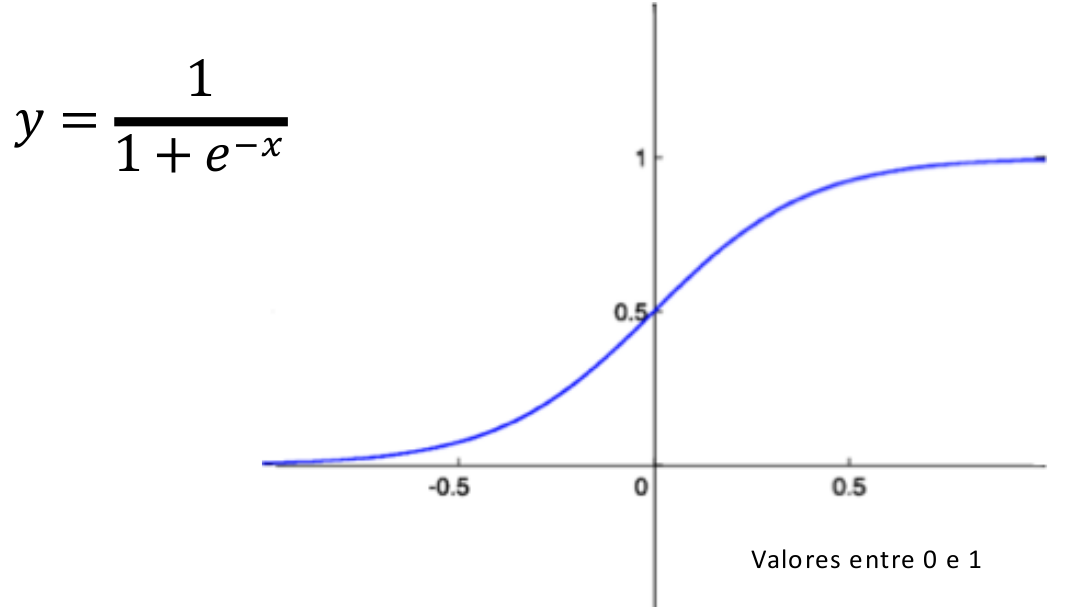
onde (30 + 1) / 2, possui valor arredondado 16.

**Para a inicialização dos pesos**, foi utilizada uma função randômica que gera números aleatórios entre 1 e -1.

O cálculo se inicia com o **somatório da multiplicação das entradas pelos respectivos pesos**, conforme fórmula abaixo:

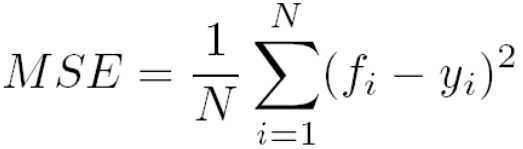


**A função de ativação** utilizada foi a sigmoide, tanto nas camadas ocultas quanto na camada de saída. A função sigmoide recebe como parâmetro de entrada X o valor do somatório dado na equação acima (soma). Para qualquer somatório como entrada, a função sigmoide retorna retorna valores entre 0 e 1 (não retorna valores negativos).



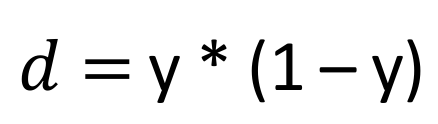
O cálculo do somatório e função de ativação se sucede até chegar na camada de saída, onde é calculado o valor do erro, para posterior ajuste dos pesos com o algoritmo de backpropagation.

**O cálculo de erro na camada de saída**, utilizou um algoritmo mais simples, onde: erro = respostaCorreta - respostaCalculada. Ao final, foi calculado o erro médio quadrado (elevando o erro ao quadrado e calculando a média em seguida). Esse cálculo de média penaliza erros maiores, tendo maior proporção do que erros menores, pois eleva o erro ao quadrado, conforme equação abaixo.



**A atualização dos pesos** na rede neural leva em consideração alguns procedimentos: descida do gradiente, cálculo da derivada sigmoide, cálculo do delta, backpropagation, taxa de aprendizagem e momento.

**O objetivo da descida do gradiente** é encontrar a combinação de pesos que o erro seja o menor possível (mínimo global), então o algoritmo calcula a derivada parcial para mover para a direção do menor valor possível do erro. O cálculo da derivada recebe como parâmetro Y o valor de cada neurônio (representado pela saída da função de ativação sigmoide), conforme fórmula abaixo:



Após o cálculo da derivada parcial, é realizado o **cálculo do delta da camada de saída**, multiplicado o erro pela derivada sigmoide, representado pela fórmula abaixo:

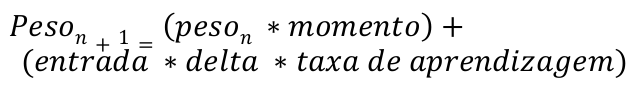


O cálculo do delta da segunda camada oculta é representado pela seguinte fórmula:

****

A diferença do cálculo do delta da segunda camada oculta para a primeira camada oculta é que **na primeira camada oculta recebe o delta da segunda camada oculta** ao invés do delta de saída.

O ajuste dos pesos do algoritmo de backpropagation é retrocedido da camada de saída até a camada de entrada, baseado na seguinte fórmula:



onde a **taxa de aprendizagem** define o quão rápido o algoritmo vai aprender e o **momento** é utilizado para acelerar a descida do gradiente para encontrar o mínimo global, escapando de mínimos locais. Por padrão, o momento possui valor 1, mas pode ser customizado pelo usuário.

Cabe destacar que o cálculo de descida de gradiente utilizado no algoritmo foi o **Batch gradient descent,** onde calcula o erro para todos os registros e em seguida atualiza os pesos para todos os registros.

O algoritmo é executado repetidas vezes até atingir uma das **duas condições de saída: atingir o limiar do erro médio quadrado ou o alcançar o número máximo de épocas (ambas informados pelo usuário).**

**Resultados obtidos antes e depois do treinamento da rede: (os gráficos encontram-se no final do relatório)**

| **Coeficiente de Aprendizado** | **Épocas** | **MSE inicial (%)** | **MSE final (%)** | **Gráfico** |
| --- | --- | --- | --- | --- |
| **0.001** | **5** | **24.94** | **23.96** | **01** |
| **0.001** | **15** | **30.18** | **22.48** | **02** |
| **0.001** | **50** | **29.51** | **20.63** | **03** |
| **0.001** | **500** | **35.36** | **19.20** | **04** |
| **0.001** | **10.000** | **42.14** | **18.07** | **05** |
| **0.1** | **5** | **56.17** | **36.73** | **06** |
| **0.1** | **15** | **26.91** | **23.95** | **07** |
| **0.1** | **50** | **25.20** | **21.77** | **08** |
| **0.1** | **500** | **24.05** | **21.33** | **09** |
| **0.5** | **5** | **51.19** | **37.26** | **10** |
| **0.5** | **15** | **43.94** | **37.26** | **11** |
| **0.5** | **50** | **31.82** | **37.26** | **12** |
| **0.5** | **500** | **26.83** | **37.26** | **13** |
| **0.0001** | **10.000** | **30.95** | **14.92** | **14** |
| **0.0001** | **100.000** | **27.76** | **15.69** | **15** |
| **0.0001** | **10.000** | **25.38** | **14.98 (8.959)** | **16** |
| **0.0002** | **10.000** | **31.30** | **17.57** | **17** |
| **0.00001** | **10.000** | **26.26** | **21.88** | **18** |

**Cabe destacar que para todas as iterações utilizou-se o limiar de erro médio quadrado 0.15, ou seja, 15% de erro.**

Utilizando o coeficiente de aprendizado com valor **0.001**, observou-se que à medida que aumenta a quantidade de épocas, o erro médio quadrado diminui, mas não atinge o limiar [01] [02] [03] [04] [05]. Esse valor é considerado baixo, ou seja, o aprendizado da rede neural será mais baixo e lento, mas tem mais chances de chegar no mínimo global.

Utilizando o coeficiente de aprendizado **0.1**, a rede não obteve um bom aprendizado, apesar de diminuir o erro médio quadrado à medida que aumenta as iterações no algoritmo [06] [07] [08] [09].

Ao aumentar o valor da taxa de aprendizado para **0.5**, acontece algo ainda pior: o valor do erro fica preso em um mínimo local de **37.26%**, mesmo alterando a quantidade de épocas [10] [11] [12] [13]. Isso se deve ao fato dessa taxa de aprendizagem ser um valor alto para a base, chegando a convergência mais rapidamente e perdendo o mínimo global. Casos curiosos acontecem nos gráficos [12] e [13], onde o erro inicial é menor do que o mínimo local convergido ao final das iterações. Isso se deve ao fato da taxa de aprendizado ser considerada alta e ao descer o gradiente convergiu em um mínimo local incorreto.

Assim, diminuiu-se ainda mais o valor da taxa de aprendizado da rede, alterando o valor para **0.0001**. Utilizando esse valor para a taxa de aprendizado, a rede neural obteve melhor aprendizado, pois obteve-se os melhores resultados de erro médio quadrado, **abaixo de 15% de erro médio quadrado** [14] [16]. Percebeu-se que ao aumentar excessivamente o número de épocas, o erro médio quadrado não caiu na mesma proporção, obtendo uma convergência em torno de 15% de erro [15].

É importante destacar que ao perceber o comportamento do erro médio quadrado com o valor de taxa de aprendizado **0.0001**, definiu-se o limiar de 0.15 (ou 15%) para o erro médio quadrado como condição de saída do algoritmo. Utilizando 10.000 épocas, o algoritmo conseguiu atingir o limiar na 8.959ª iteração [16].

Realizou-se outros dois novos testes com novos valores de taxa de aprendizado, ambas com 10.000 épocas de treinamento, o primeiro foi com a taxa de aprendizado de valor **0.0002** [17] e o segundo de valor **0.00001** [18]. **No primeiro teste, ao aumentar o valor da taxa de aprendizado de 0.0001 para 0.0002**, o algoritmo obteve um bom resultado de erro médio quadrado (17.57%), mas não tão bom quanto o resultado com valor 0.0001 de taxa de aprendizado; **no segundo teste, ao diminuir o valor da taxa de aprendizado de 0.0001 para 0.00001,** o algoritmo não obteve um bom aprendizado quando comparado aos resultados com valor 0.0001 de taxa de aprendizado [14] [16], devido a diminuir excessivamente o valor da taxa de aprendizado.

Pode-se concluir que os melhores resultados do algoritmo de rede neural foi ao utilizar a taxa de aprendizado com valor 0.0001 e 10.000 épocas, obtendo um erro médio quadrado abaixo de 15% [14] [16].

1. **Implementação do modelo sequencial em camadas Densas do keras**

A segunda implementação aplicada a base de dados utilizou o modelo sequencial em camadas densas do keras. Essa implementação tem o objetivo de comparar os resultados obtidos com a implementação da rede neural utilizando apenas funções do numpy.

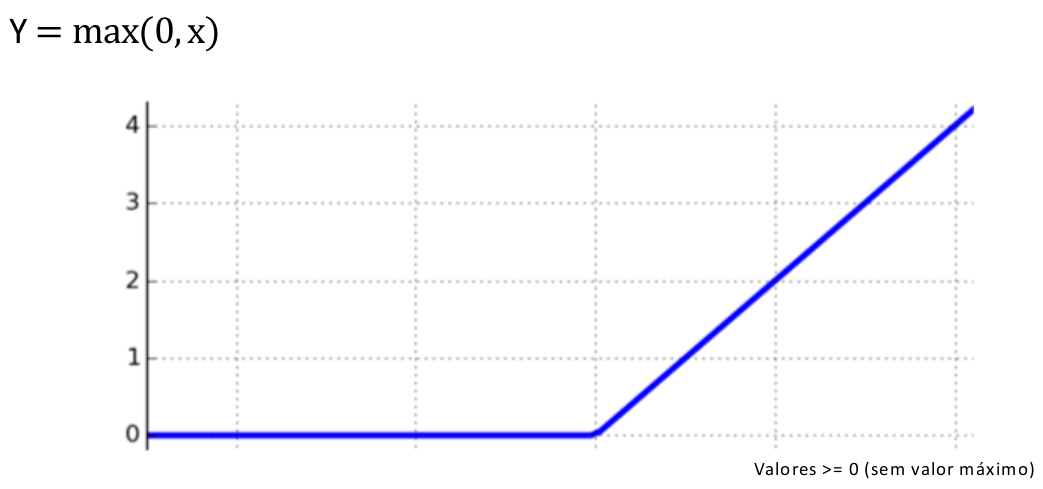
Foram aplicadas várias técnicas utilizando o keras com o intuito de melhorar os resultados obtidos. A seguir, são enumeradas as técnicas utilizadas.

**2.1 Brest Cancer Simples (breast-cancer-simples.ipynb)**

A base de dados é inicialmente dividida em treino e teste, definindo 25% da base para dados que serão utilizados no teste da rede neural.

A quantidade de camadas (entrada, ocultas e saída) e neurônios por camada segue o mesmo padrão implementado utilizado apenas funções do numpy, ou seja, são utilizadas 30 neurônios na camada de entrada, duas camadas ocultas (cada uma com 16 neurônios) e uma camada de saída com 1 neurônio.

A função de ativação utilizada nas camadas ocultas é a função ReLU (Rectified Linear Units) e a função de ativação utilizada na camada de saída é a função sigmoide (retornando valores entre 0 e 1). Na função ReLU, caso o somatório da multiplicação das entradas pelos respectivos pesos seja maior do que zero, a função retorna a soma; caso contrário, retorna 0. A imagem a seguir mostra a equação utilizada na função relu:



Função ReLU

A inicialização dos pesos dos pesos é realizada com uma distribuição uniforme (parâmetro kernel\_initializer = 'random\_uniform').

Foi utilizado **o otimizador Adam para atualizar os pesos** da rede de forma iterativa com base nos dados de treinamento, ou seja, atualiza os pesos para o erro de cada entrada. Assim, ele não espera calcular o erro de todas as entradas para realizar a atualização dos pesos.

A taxa de aprendizagem utilizada foi 0.001. O decaimento da taxa de aprendizagem foi calculado após a atualização dos pesos, o valor do decaimento utilizado foi 0.0001. Além disso, utilizou-se do atributo clipvalue = 0.5 para prender o valor da taxa de aprendizagem para que ele não fique muito disperso dentro do gradiente.

Utilizou-se também uma função de perda binary\_crossentropy, pois há apenas duas classes (0 ou 1) para o problema de classificação utilizado. A função de perda define a forma como irá fazer o cálculo do erro.

O algoritmo é executado durante 100 épocas de treinamento. Durante o treinamento da rede neural, o cálculo do erro é feito em lotes de 10 registros, ou seja, calcula o erro para 10 registros e depois atualiza os pesos.

Para a **predição, é utilizado um limiar de 0.5 para a classificação**, sendo valor 1 caso o neurônio da camada de saída tenha probabilidade maior que 0.5.

As métricas utilizadas para fazer a avaliação da classificação da base foram a acurácia e o índice kappa. A tabela a seguir apresenta os resultados da avaliação:

| **Base de dados** | **Acurácia** | **Função de perda** | **kappa** |
| --- | --- | --- | --- |
| Treino | 0.8527 | 0.6215 | ---- |
| Teste (25%) | 0.8462 | 0.9993 | 0 1  0 [35, 18]  1 [ 4, 86] |

Com base no índice kappa, a rede neural classificou corretamente 35 amostras para a classe 0 e 86 amostras para a classe 1. Além disso, classificou incorretamente 18 amostras que pertencem a classe 0 como sendo da classe 1; e classificou 4 amostras que pertencem a classe 1 como sendo da classe 0.

**2.2 Brest Cancer Cruzada (breast-cancer-cruzada.ipynb)**

Esse algoritmo apresenta uma melhoria do algoritmo descrito no item 2.1.

**Duas novas técnicas foram inseridas nesse novo algoritmo: Dropout e Validação Cruzada (corss-validation)**.

O Dropout é uma técnica utilizada para evitar o overfitting no treinamento da base, ele zera o valor de um percentual de neurônios da camada de entrada (ou oculta). Esses valores zerados não terão influência sobre o resultado final. Porém, ao inserir um valor muito alto no dropout, ocorrerá underfitting e a rede neural não vai conseguir aprender. O valor utilizado no Dropout da camada de entrada e nas camadas ocultas é 0.2, ou seja, zera o valor de 20% dos neurônios da camada de entrada e das camadas ocultas.

A validação cruzada define um número de vezes que base de dados será dividida entre treino e teste e não apenas uma única vez como no algoritmo do item 2.1. Para cada vez que realizar a divisão da base de dados, a rede é treinada durante 100 épocas. Ao final das 100 épocas, o algoritmo guarda a acurácia obtida para cada treinamento. Foi definido o valor 10 para a validação cruzada. Assim, para cada validação cruzada o treinamento é realizado durante 100 épocas, totalizando 1.000 épocas de treinamento na rede.

| **Validação cruzada** | **Épocas** | **Média Acurácia** | **Média geral acurácia** | **Desvio padrão** |
| --- | --- | --- | --- | --- |
| 1 | 100 | 0.737 | 0.877 | 0.072 |
| 2 | 100 | 0.877 |
| 3 | 100 | 0.912 |
| 4 | 100 | 0.842 |
| 5 | 100 | 0.930 |
| 6 | 100 | 0.930 |
| 7 | 100 | 0.860 |
| 8 | 100 | 0.947 |
| 9 | 100 | 0.772 |
| 10 | 100 | 0.964 |

Pode-se observar que o valor da acurácia varia entre 77% e 96%, devido a interferência que o conjunto de dados das amostras de treino e teste foram divididos.

**2.3 Brest Cancer Tuning (breast-cancer-tuning.ipynb)**

Esse algoritmo apresenta uma melhoria do algoritmo descrito no item 2.2.

**A pesquisa em grade foi inserida nesse algoritmo para buscar os melhores parâmetros a serem utilizados na rede neural.**

Para a seleção dos melhores valores de parâmetros, foi passado um dicionário contendo as opções para cada parâmetro, conforme a tabela abaixo:

| **Parâmetro** | **Descrição** | **Opções de valor** |
| --- | --- | --- |
| **batch\_size** | Quantidade de registros que tem q passar para calcular o ajuste de pesos | [10, 30] |
| **epochs** | Quantidade de épocas | [50, 100] |
| **optimizer** | Otimizador utilizado no treinamento. adam tem a tendência de dar resultados melhores do que o sgd | ['adam', 'sgd'] |
| **loos** | Função de perda. 'binary\_crossentropy' é o padrão (com resultados melhores); ‘hinge’ é outra função que pode ser utilizada para problemas de classificação binária | ['binary\_crossentropy', 'hinge'] |
| **kernel\_initializer** | Inicialização dos pesos. A valor ‘normal’ usa uma distribuição estatística normal | ['random\_uniform', 'normal'] |
| **activation** | Função de ativação. ‘tanh’ é a tangente hiperbólica que retorna valores entre -1 e 1 | ['relu', 'tanh'] |
| **neurons** | Quantidade de neurônios da camada escondida | [16, 8] |

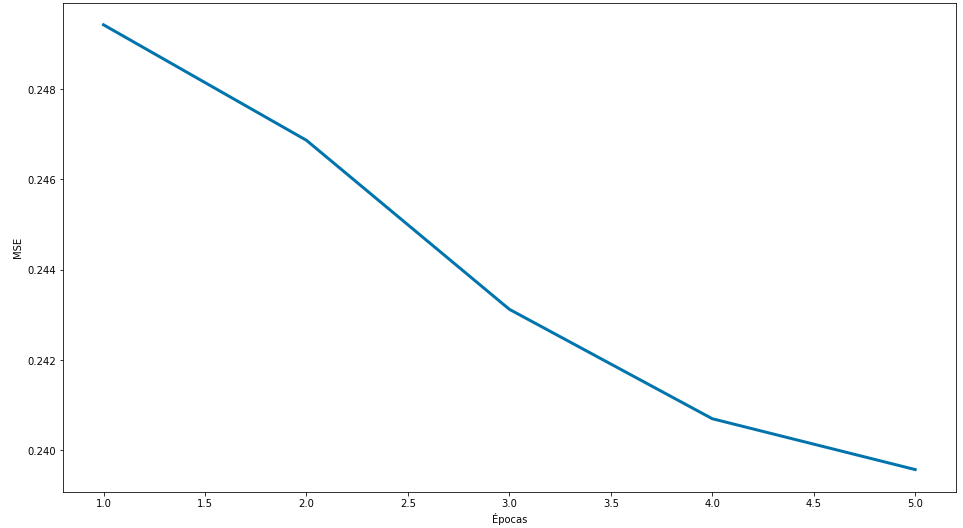
Cabe destacar que a função de ativação da camada de saída continua sendo a função sigmoide, pois ela retorna um valor com probabilidade entre 0 e 1 para a classificação do target de saída. Além disso, a métrica de avaliação dos resultados continua sendo a acurácia.

O algoritmo foi executado durante 5 horas e retornou os melhores parâmetros a serem utilizados na rede neural. A tabela a seguir exibe esses parâmetros retornados.

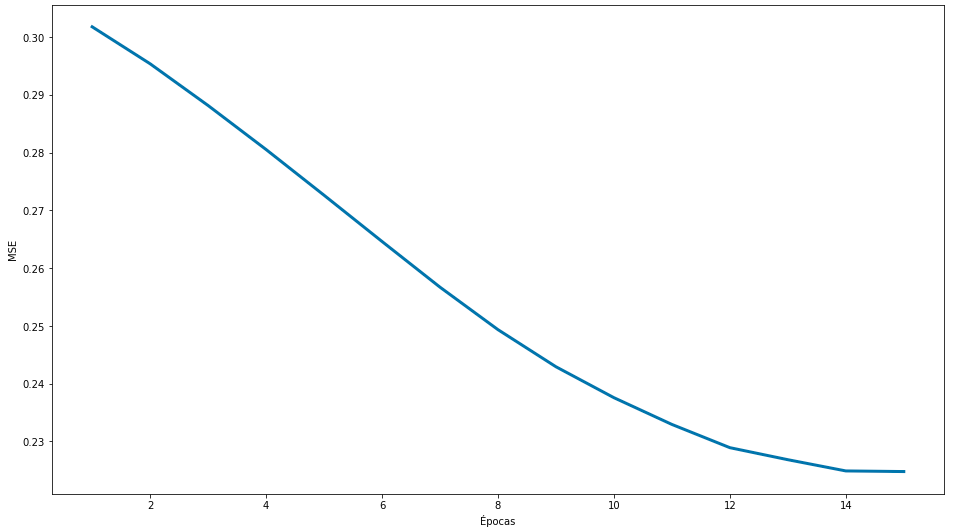
| **Parâmetro** | **Descrição** | **Melhor valor** |
| --- | --- | --- |
| **batch\_size** | Quantidade de registros que tem q passar para calcular o ajuste de pesos | 10 |
| **epochs** | Quantidade de épocas | 100 |
| **optimizer** | Otimizador utilizado no treinamento. adam tem a tendência de dar resultados melhores do que o sgd | adam |
| **loos** | Função de perda. 'binary\_crossentropy' é o padrão (com resultados melhores); ‘hinge’ é outra função que pode ser utilizada para problemas de classificação binária | binary\_crossentropy |
| **kernel\_initializer** | Inicialização dos pesos. A valor ‘normal’ usa uma distribuição estatística normal | random\_uniform |
| **activation** | Função de ativação. ‘tanh’ é a tangente hiperbólica que retorna valores entre -1 e 1 | relu |
| **neurons** | Quantidade de neurônios da camada escondida | 8 |

**A aplicação desses melhores parâmetros na rede neural, resultou em um precisão de 0.9087, ou seja, 90.87% de precisão na base de dados.**

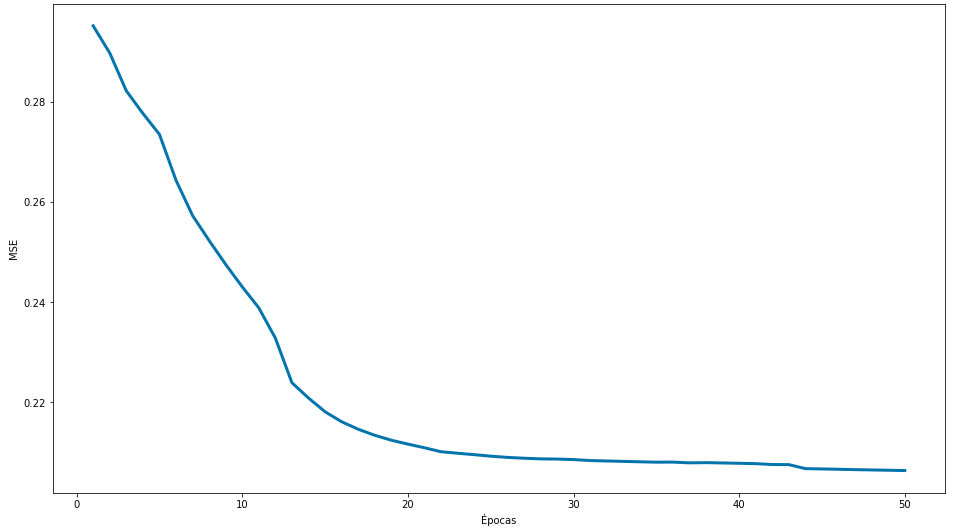
**GRÁFICOS**



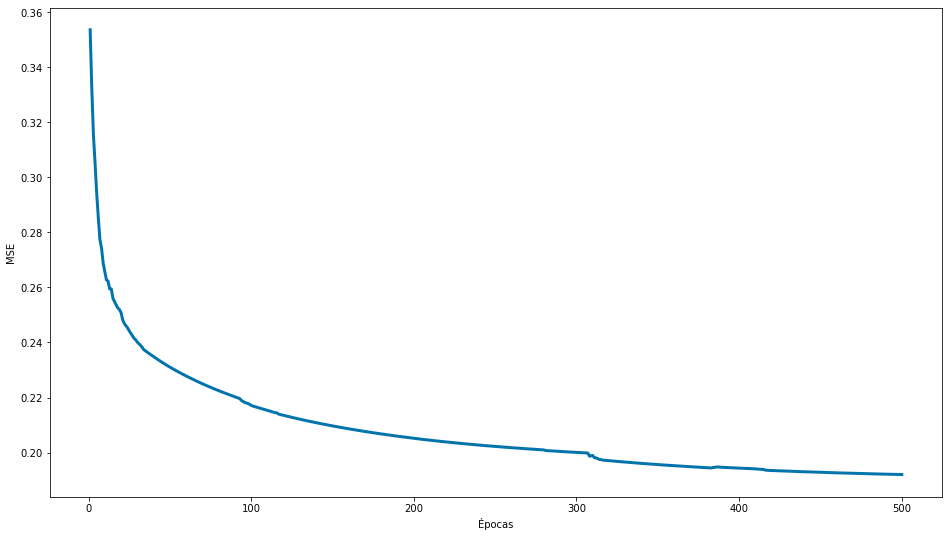
[01]



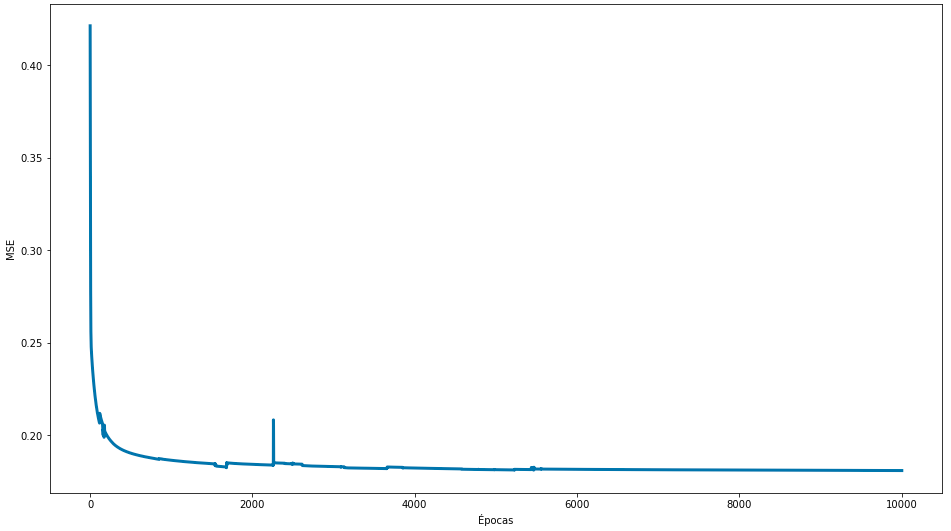
[02]



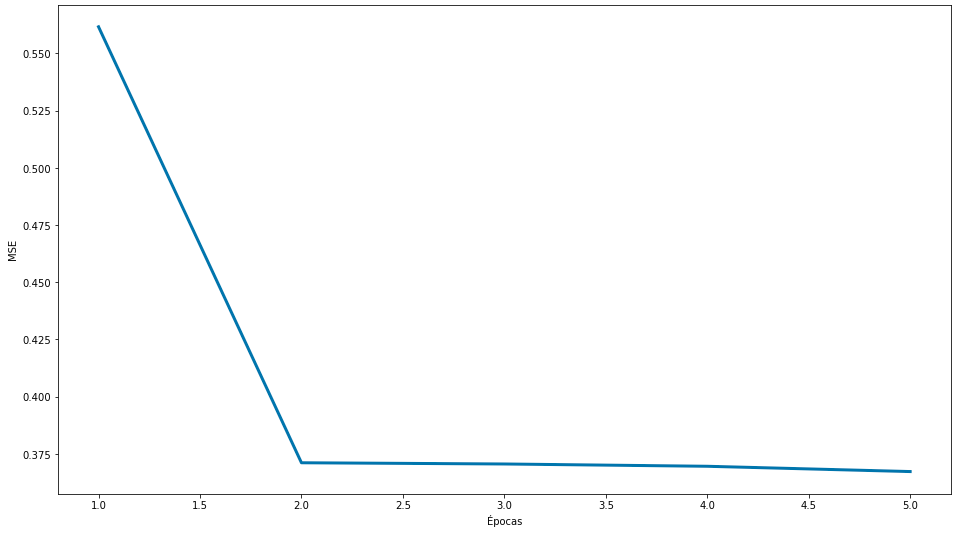
[03]



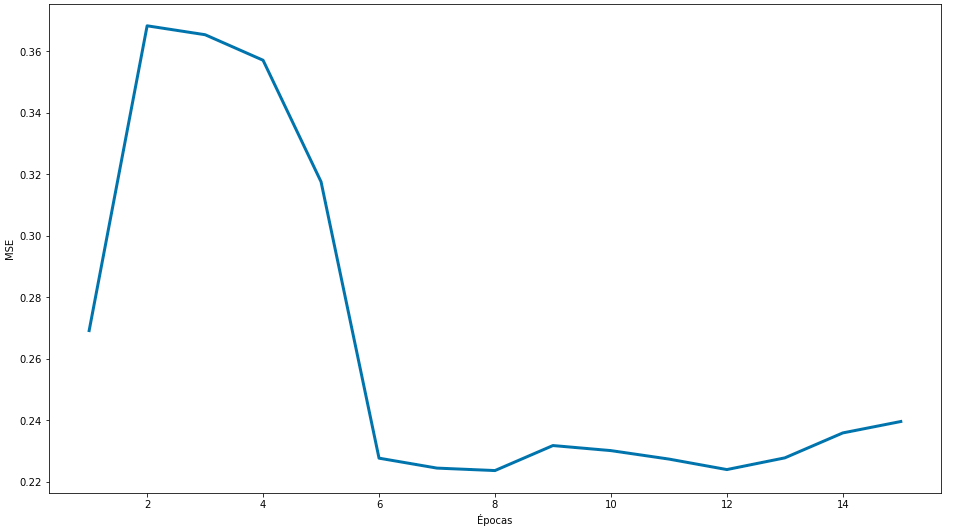
[04]



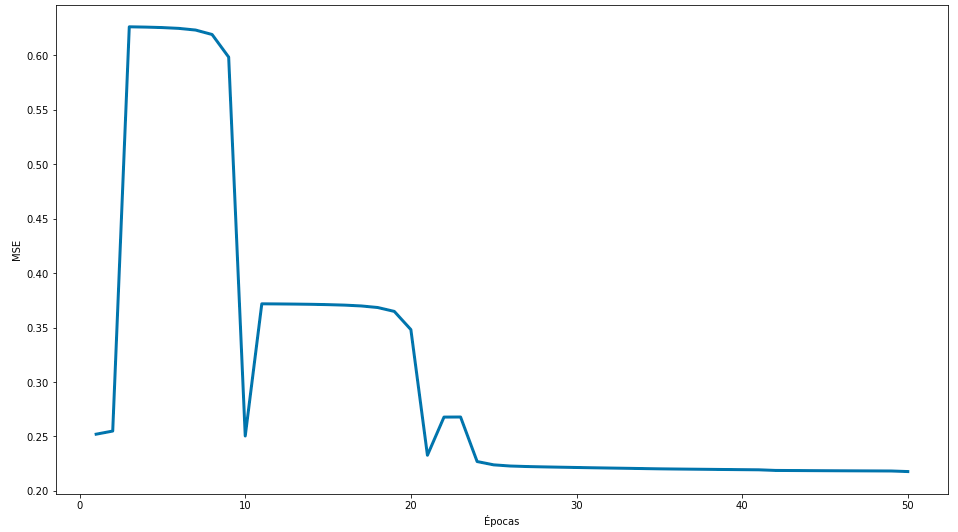
[05]



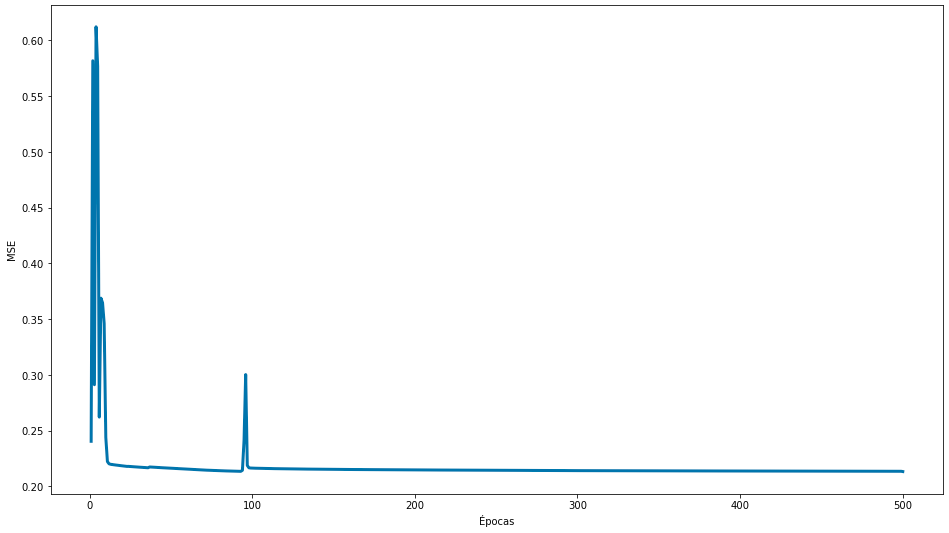
[06]



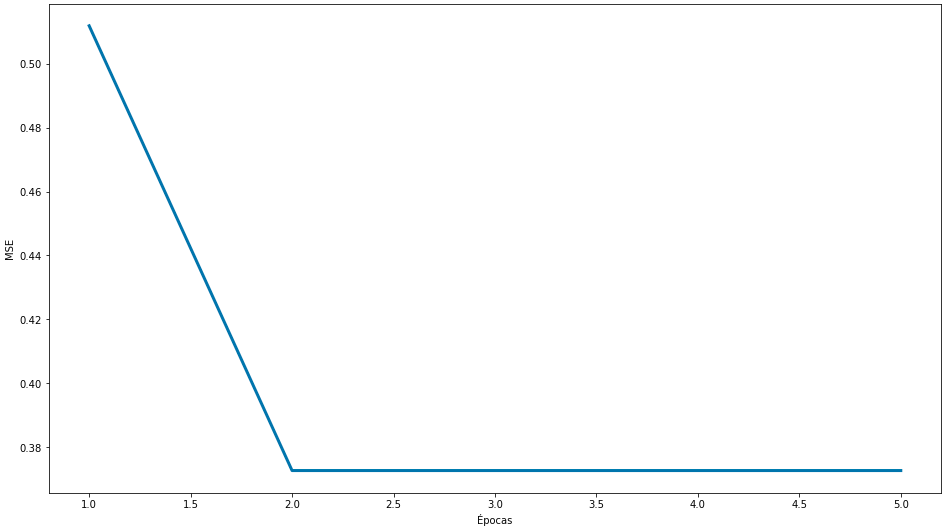
[07]



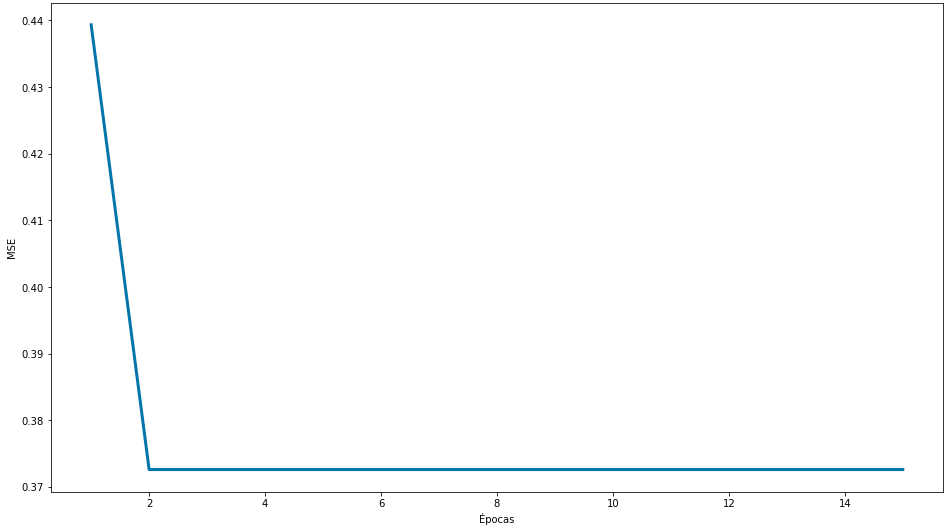
[08]



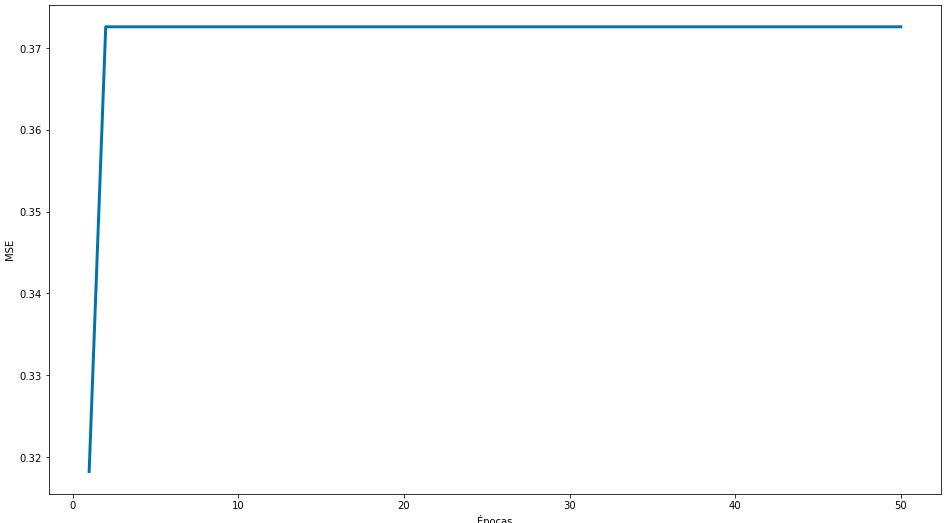
[09]



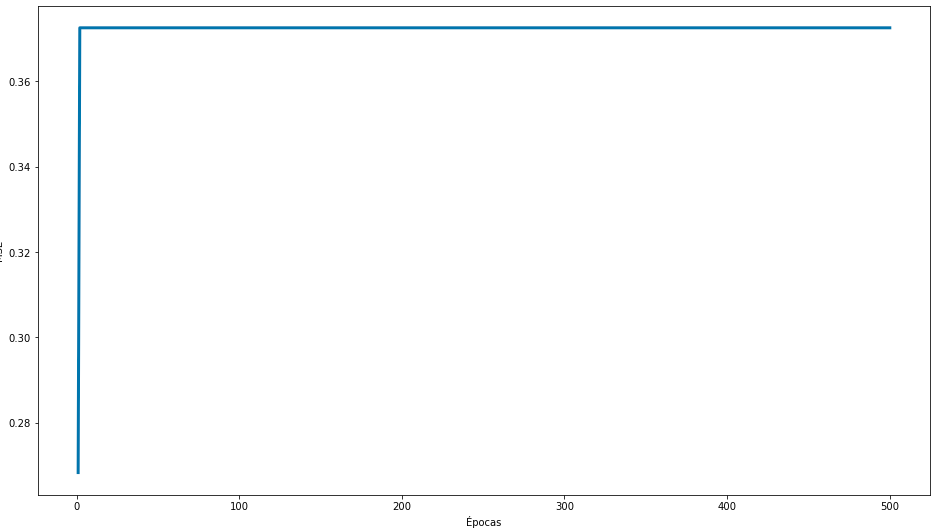
[10]



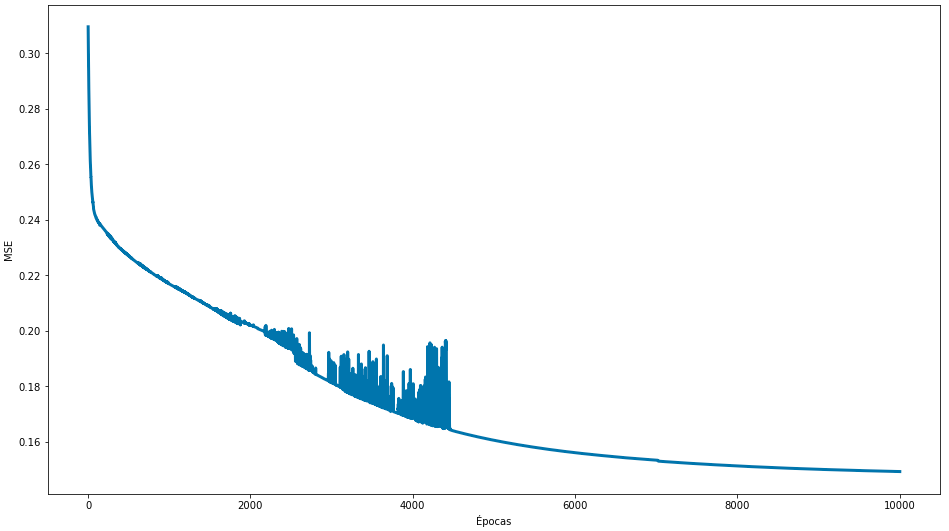
[11]



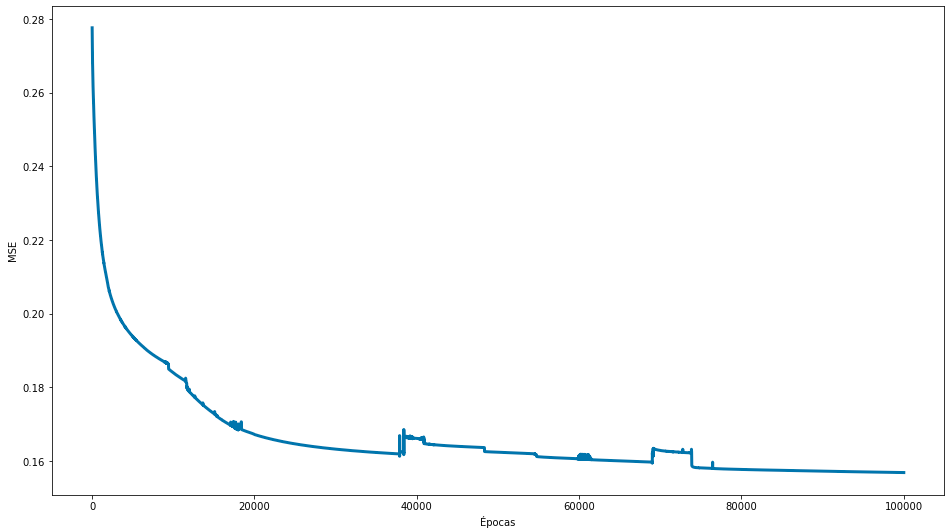
[12]



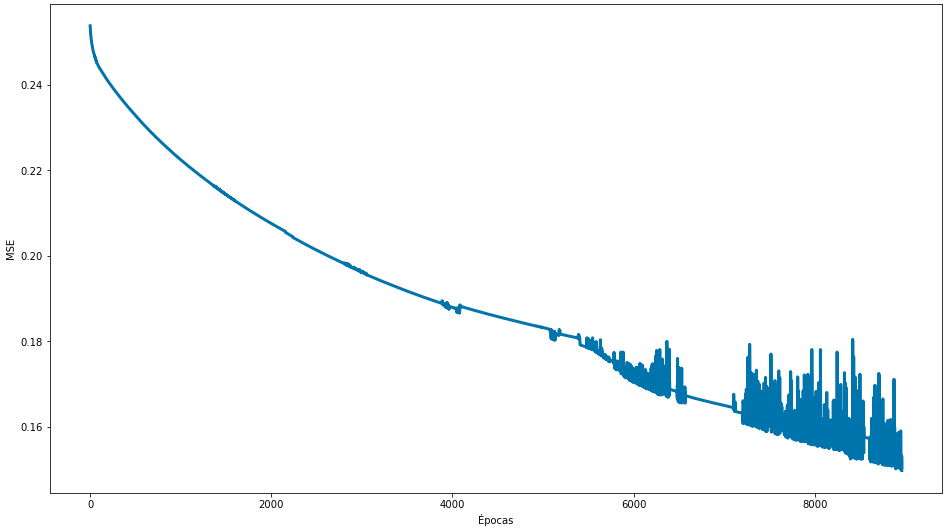
[13]



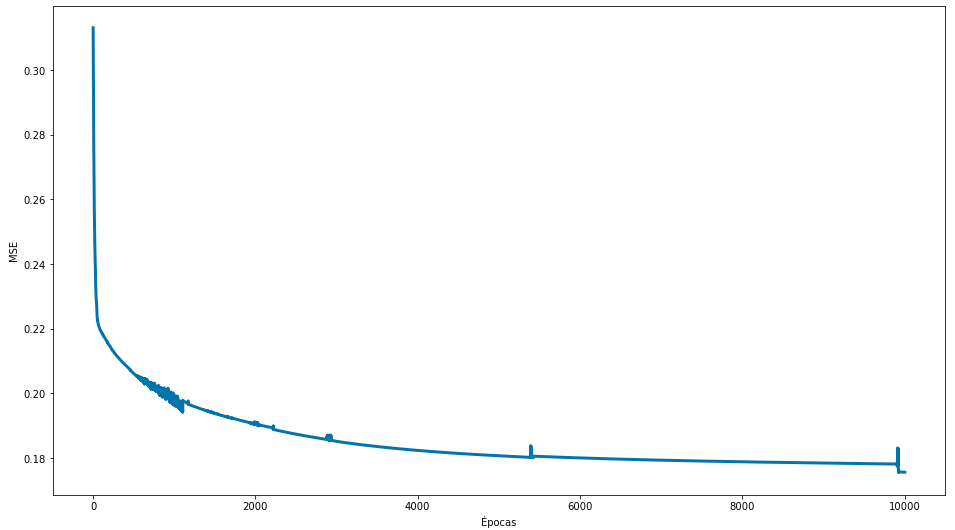
[14]



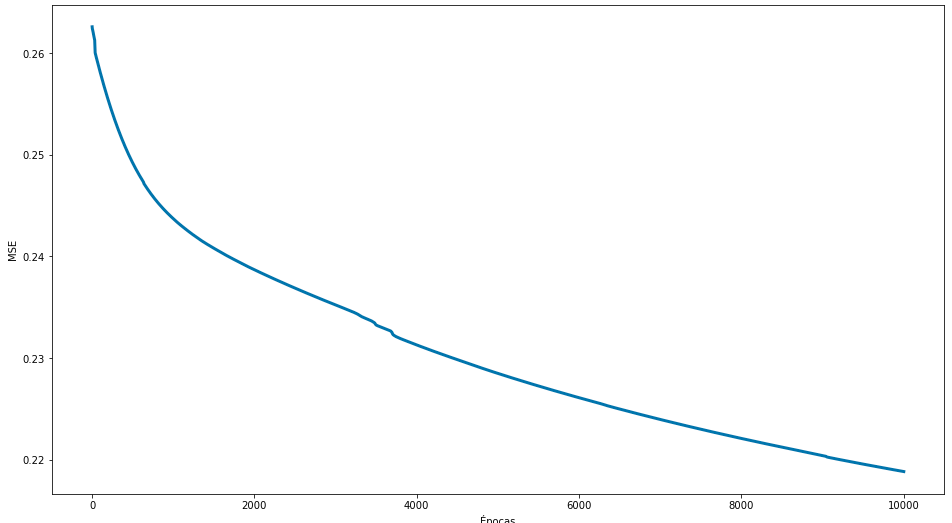
[15]



[16]



[17]



[18]