I. Introdução

O presente trabalho estudou a implementação do perceptron de camada simples e, posteriormente, avaliação do desempenho do algoritmo em reconhecer padrões em base de dados linearmente separáveis e não linearmente separáveis. Foram realizados experimentos variando os hiper-parâmetros da rede com o objetivo de determinar a influência na velocidade de convergência do algoritmo e no erro final da rede.

Desse modo, o trabalho em questão está organizado da seguinte maneira. Na seção II está descrito como foi realizado a implementação do rede perceptron, os principais parâmetros de entradas e demais características presentes na implementação. Na seção III são apresentadas as bases de dados utilizadas, os experimento realizados e os resultados obtidos. Por fim, na seção IV são feitas as conclusões gerais sobre o trabalho e sugestões para futuros experimentos.

II. Implementação do Percetron

A implementação da rede foi feita na linguagem de programação Python de modo a aproveitar as funcionalidades de operações matriciais disponíveis no pacote do numpy. Além disso, utilizou-se também de programação orientada a objetos para definir a estrutura que representa uma camada da rede neural.

Desse modo, foi criado então um pacote (conjunto de bibliotecas) que agrupam a classe, a função de treinamento e a função bootstrap para avaliação de desempenho.

Neste pacote a implementação foi separada em dois scripts: `perceptron.py`, onde foi implementado a estrutura da rede neural, e `practice.py`, onde foram implementadas as funções de treinamento e de avaliação de desempenho.

A. Implementação da Rede Perceptron

Para a criação dos neurônios do perceptron foi criado uma classe chamada `single_layer` onde são passados os seguintes parâmetros de inicialização:

- **neurons**: Número de neurônios.
- **atributes**: Número de atributos.
- **func_activation**: Função de ativação (por padrão a sigmoid).
- **threshold**: Valor de corte (por padrão 0.5).

Com esses parâmetros a classe consegue criar uma matriz de pesos em que cada linha representa os neurônios definidos na inicialização do objeto e as colunas os seus respetivos pesos.

Um ponto interessante de definir uma camada como uma classe é que, com a matriz de pesos é um atributo interno da classe, quando atualizada, não é necessário atribuir a matriz nova à uma variável, já que ela é atualizada no próprio objeto.

Além da matriz de pesos, foram criados alguns métodos para facilitar a manipulação da camada. Os principais métodos são:

- **field_ind**: Calcula o campo induzido.
- **apply**: Calcula o valor da função de ativação no campo induzido.
- **apply_abs**: Utiliza o valor de corte para dizer se o neurônio ativa ou não.
- **weight_aplicate**: Substitui a matriz de pesos atual por uma matriz nova.
- **weight_random_init**: Atribui valores aleatório entre 0 e 1 aos pesos da matriz de pesos.

Os métodos internos do objeto são encarregados de fazer as operações neurônio a neurônio dentro da classe, desse modo, é possível trabalhar com a camada independente da quantidade de neurônios que tenha sido criada inicialmente.

A estrutura de um neurônio pode ser observada na figura 2

Fig. 1. Fonte: Haykin.

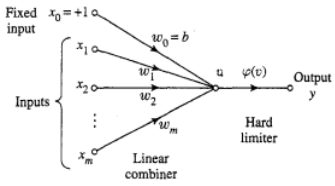

Fig. 2. Estrutura de um neurônio.

A estrutura `single\_layer` foi pensada para que sejam utilizadas as mesmas quantidade de neurônios do que quantidade de classes envolvidas no problema de classificação.

Desse modo, a saída final da camada é um vetor, com valores contidos dentro do intervalo de trabalho da função de ativação escolhida, e cada neurônio indica a chance de entrada pertencer a uma classe.

Utilizando uma função como a sigmoid, por exemplo, em que os valores de saída estão contidos entre 0 e 1, quanto mais próximo a sáida de um neurônio for de 1, isso indica que maior é a chance daquele objeto pertencer a classe que o respectivo neurônio se especializou em classificar.

Uma classificação perfeita é dada pelo caso em apenas um neurônio ativa, tem como saída valor igual a 1, e todos os outros não ativam, saída igual a zero.

B. Função de Treinamento

A função de treinamento, `training_layer`, foi definida de modo que as entradas principais fossem:

- **single_layer**: Objeto do tipo `{single\_layer`.
- **database**: Dados de entrada.
- **data_out**: Saídas esperadas.

São setados ainda parâmetros opcionais que podem ser modificados pelos usuários a fim de caracterizar o modelo, sendo eles:

- η: Taxa de aprendizagem.
- **number_of_epoca**: Número de épocas de treinamento.
- **show_per_epoca**: Intervalo para mostrar resultados parciais na tela.
- **random**: Condição que define se os dados serão embaralhados ou não a cada época.
- **tol**: Tolerância erro aceitável para convergência do algoritmo.

O treinamento foi realizado conforme mostrado na figura **??** e segue. Os dados são apresentados para a rede, a resposta é

computada e comparada com a saída desejada. O erro gerado por meio da comparação é utilizado para atualizar os pesos dos neurônios conforme a equação 1 apresentada por Haykin.

$$w(n+1) = w(n) + \eta e(n) x(n) \tag{1}$$

onde n representa a instância atual.

Para avaliar a performance da rede neural foi implementado o método bootstrap, visto que todas as bases de dados são pequenas, e foi extraído o erro médio e o desvio padrão com base num número pré-definido de iterações do método.

III. Experimentos

A. Base de Dados

1) Base Sintética:

Para validar a implementação, inicialmente, foi gerado uma base de dados linearmente separável com números pseudo aleatórios entre intervalos especificados.

Foram determinadas duas regiões quadradas no plano (duas classes) com 100 pontos cada (figura 3) para gerar os dados e posteriormente realizar o treinamento.

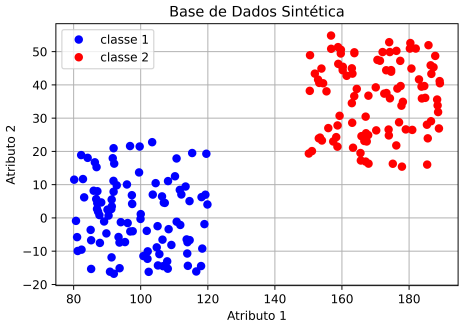

Fig. 3. Base de dados Sintética.

Essa base possui apenas dois atributos de entradas que localiza no plano a posição do ponto de interesse.

2) Base Iris[Disponível em http://archive.ics.uci.edu/ml/datasets/Iris]*:*

Para os testes com uma base não linearmente separável foi escolhido a base da Iris, visto que ela é muito popular no âmbito de treinamento de classificadores e permite avaliar bem o desenvolvimento da rede criada.

A Iris possui 150 objetos e 4 atributos de entrada da rede, seguindo a ordem do arquivo, são eles:

- Comprimento de pétala
- Comprimento de sépala
- Largura de pétala
- Largura de sépala

Esses atributos estão relacionados com 3 classes na base de dados: versicolor, virgínica e setosa. Os dados estão distribuídos de forma uniforme, ou seja, a mesma quantidade de pontos para cada uma das classes.

B. Modelos da Rede e Características de Treinamento

1) Base Sintética:

Para a base sintética foi gerado uma rede com dois neurônico e com duas entradas cada.

Utilizou-se tanto a função sigmoid como a função degrau para realizar o treinamento da rede e, posteriormente, foram comparados os resultados do erro quadrático médio por época de treinamento. A taxa de aprendizado foi fixado como constante em 0.1.

Para verificar a influência da taxa de aprendizagem (η), foi realizado o treinamento da rede com seis valores diferentes de η: 0.5, 0.3, 0.15, 0.1, 0.05, 0.01.

Ainda, foi estudado a influência de se variar o eta por época de treinamento seguindo uma rampa e uma curva exponencial decrescente. As curvas tinham como valor inicial $\eta = 0.5$ e valor final $\eta = 0.05$.

Para determinar essa curvas foi necessário estimar a quantidade máxima de épocas de treinamento, assim foi utilizado o caso de duas vezes a quantidade máxima de épocas nos casos em que η era constante. Garantido uma segurança caso o algoritmo apresentasse uma piora com essa estratégia.

2) Base da Iris:

Já para a base da Iris foi gerado um modelo com três neurônios na camada, um para cada classe.

Tendo em vista que era uma base de dados muito mais custosa computacionalmente de realizar as análises, e que, por ser não linearmente separável, o perceptron naturalmente não convergiria, foi feito uma única rodada de treinamento com o modelo com η fixo igual 0.1, a fim de analisar o RMS por época.

3) Critério de Parada:

O critério de parada para todos os treinamentos aqui utilizados foram de 1×10^{-3}, de modo a garantir uma boa equivalência entre quantidade de iterações e acurácia dos métodos.

C. Avaliação de Desempenho

Para realizar a avaliação de desempenho dos modelos supracitados, foi utilizado o bootstrap como medida de performance. O método foi implementado com auxílio de funções do pacote do sci-kit learn.

Foram rodadas 10 e 12 iterações do boostrap, respectivamente, para a base de dados sintéticas e Iris, avaliando a taxa de erro a cada fim de treinamento.

Para a base sintética utilizou-se o valor fixo de η que apresentou o melhor comportamento das análises anteriores e para a base da Iris utilizou-se o valor fixo de $\eta = 0.1$.

D. Resultados

1) Base Sintética - Comparação entre funções de ativação:

Na figura 4 é possível observar a oscilação do erro RMS para cada uma das funções. A função degrau levou 81 épocas a mais do que a função sigmoid para convergir.

Além disso, note que os picos de oscilação de erro da função degrau são maiores. Isso se deve pelo fato de ser uma função descontínua com uma mudança muito abrupta de valores.

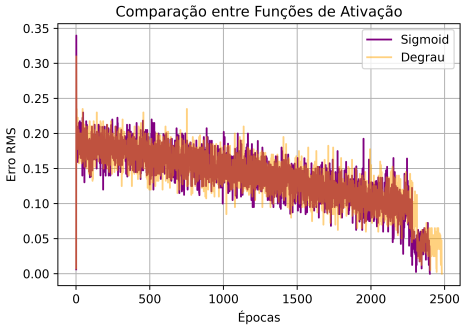

Fig. 4. Erro RMS por época para as funções sigmoid e degrau.

2) Base sintética - Comparação entre η constantes:

Nesse experimento a quantidade de épocas necessárias para a convergência variou bastante entre as taxas de aprendizado. Dentre elas, $\eta = 0.3$, obteve o melhor resultado com 2390 iterações para convergir, 392 iterações a menos do pior resultado ($\eta = 0.01$).

A figura 5 mostra a comparação dos erros RMS para cada uma das taxas de aprendizagem. As curvas mais transparentes possuem η cada vez menor.

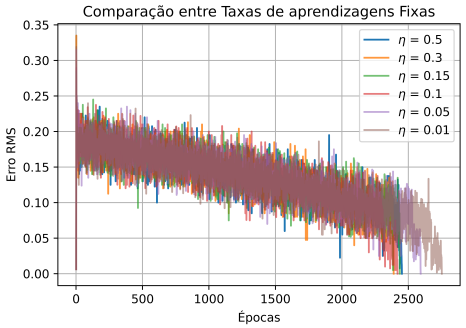

Fig. 5. Erro RMS por época para valores de η constante.

3) Base sintética - Comparação entre η variáveis:

Utilizando as curvas de rampa e exponencial para a taxa de aprendizagem é possível perceber pela figura 7 que a quantidade de épocas necessárias para atingir a convergência diminuiu significativamente.

Em torno de 800 épocas a menos foram necessárias para a rede ter resultados abaixo da tolerância.

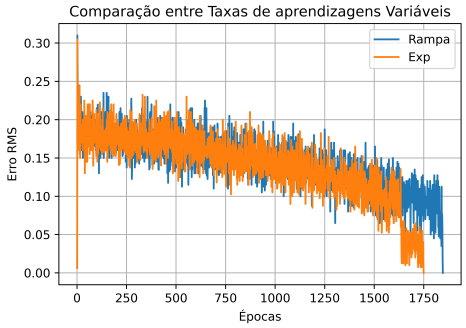

Fig. 6. Erro RMS por época para curvas de η.

4) Base Sintética - Performance:

Na tabela **??** encontram-se os valores de média e desvio padrão do erro calculado pelo bootstrap com o caso de $\eta = 0.3$, melhor entre as taxas de aprendizado constante, e para a curva exponencial decrescente, melhor entre as curvas da taxa de aprendizagem.

TABLE I
BOOTSTRAP BASE SINTÉTICA

Taxa de Aprendizagem (η)	**Média**	**Desvio padrão**
0.3	0.751%	1.291%
Curva Exponencial	0.278%	0.556%

5) Iris - Erro RMS :

O database da Iris foir treinado com um valor constante de taxa de aprendizado igual a 0.1, o número de épocas totais para o treinamento foi setado em 80 mil épocas.

Como mostrado por Jaskowiak (2011), um das classes da Iris é linearmente separável das outras duas, desse modo, utilizou-se um alto número de épocas para buscar um queda no erro RMS que indicasse que o perceptron estava conseguindo separar aquela classe das demais.

Na figura **??**, no entanto, não foi possível notar eventos que pudessem indicar a hipótese sugerida.

6) Iris - Bootstrap:

O número máximo de épocas da função de treinamento passada para o bootstrpa da Iris foi alterado para 60 mil de modo que isso baixasse o custo computacional envolvido. A taxa de aprendizado foi mantida fixa igual 0.1.

A fim de buscar mais confiabilidade, o bootstrap foi rodado com 12 iterações. Os resultados da média e desvio padrão erro podem ser apreciados na tabela II.

É interessante notar que apesar do erro RMS não indicar um comportamento de classificação de uma das classes, o

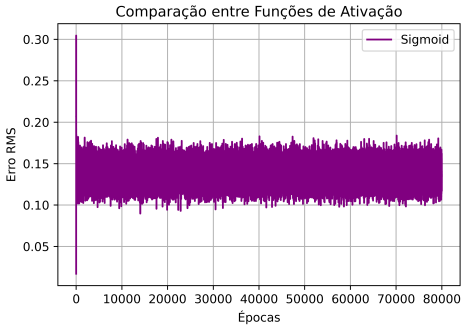

Fig. 7. Erro RMS por época.

TABLE II
BOOTSTRAP BASE IRIS

Taxa de Aprendizagem (η)	**Média**	**Desvio padrão**
0.1	31.955%	6.472%

bootstrap mostrou que o perceptron teve um erro 32% aproximadamente.

Isso implica dizer que ele ainda obtém um acerto considerável, não esperado para o caso de uma base não linear.

IV. CONCLUSÃO

Este trabalho estudou a implementação do perceptron e aplicação da rede para problemas de classificações utilizando uma base linearmente separável e não linearmente separável.

A utilização de orientação a objeto para implementação da rede e seus respectivos métodos tornaram o algoritmo mais genérico e facilitaram as análises para diferentes modelos de rede.

O perceptron se mostrou bastante eficaz na separação da base sintética gerada pelo autor. Foram feitas algumas mudanças na base de dados para verificar a influência na fase de treinamento e foi notado que isso poderia aumentar ou diminuir o tempo de convergência do algoritmo.

Para trabalhos futuros é interessante buscar base de dados reais para testar o algoritmo com uma base gerada com pouco influência do individuo gerador.

Por fim, o perceptron surpreendeu na aplicação para a base de dados da Iris. Era esperado um maior erro médio durante a avaliação da performance. É interessante nos próximos experimentos testar as curvas de taxa de aprendizado em bases não linearmente separáveis para medir o impacto na resposta obtida.

REFERENCES

Haykin, S. and Hakin, S., n.d. The Neural networks a comprehensive foundation. 2nd ed. Macmillan College Publishing Company