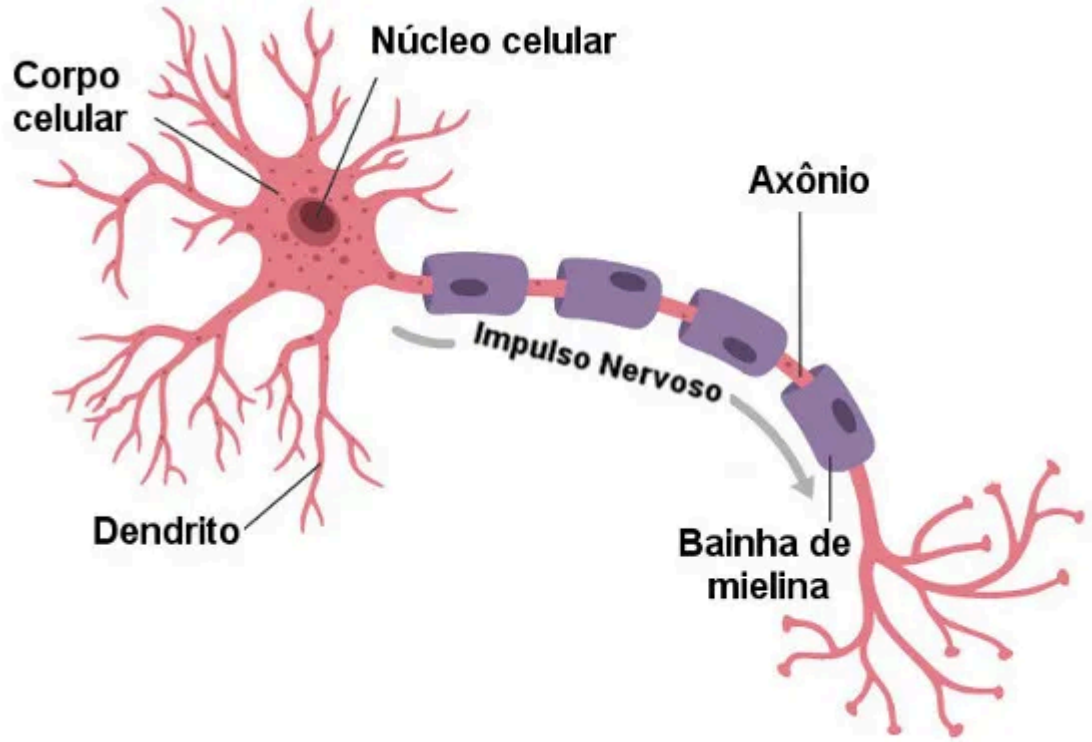
Relatório-card 13 - Prática: Redes Neurais (II)

Jefferson korte junior

3 - Teoria resumida sobre redes neurais artificiais

Aula 07: Essa aula fala sobre o funcionamento do nosso cérebro, em relação aos bilhões de neurônios que existem, e suas conexões que são responsáveis por tudo que fazemos, desde andar, falar, ler e aprender uma língua nova que daí nossos neurônios criam novas conexões.

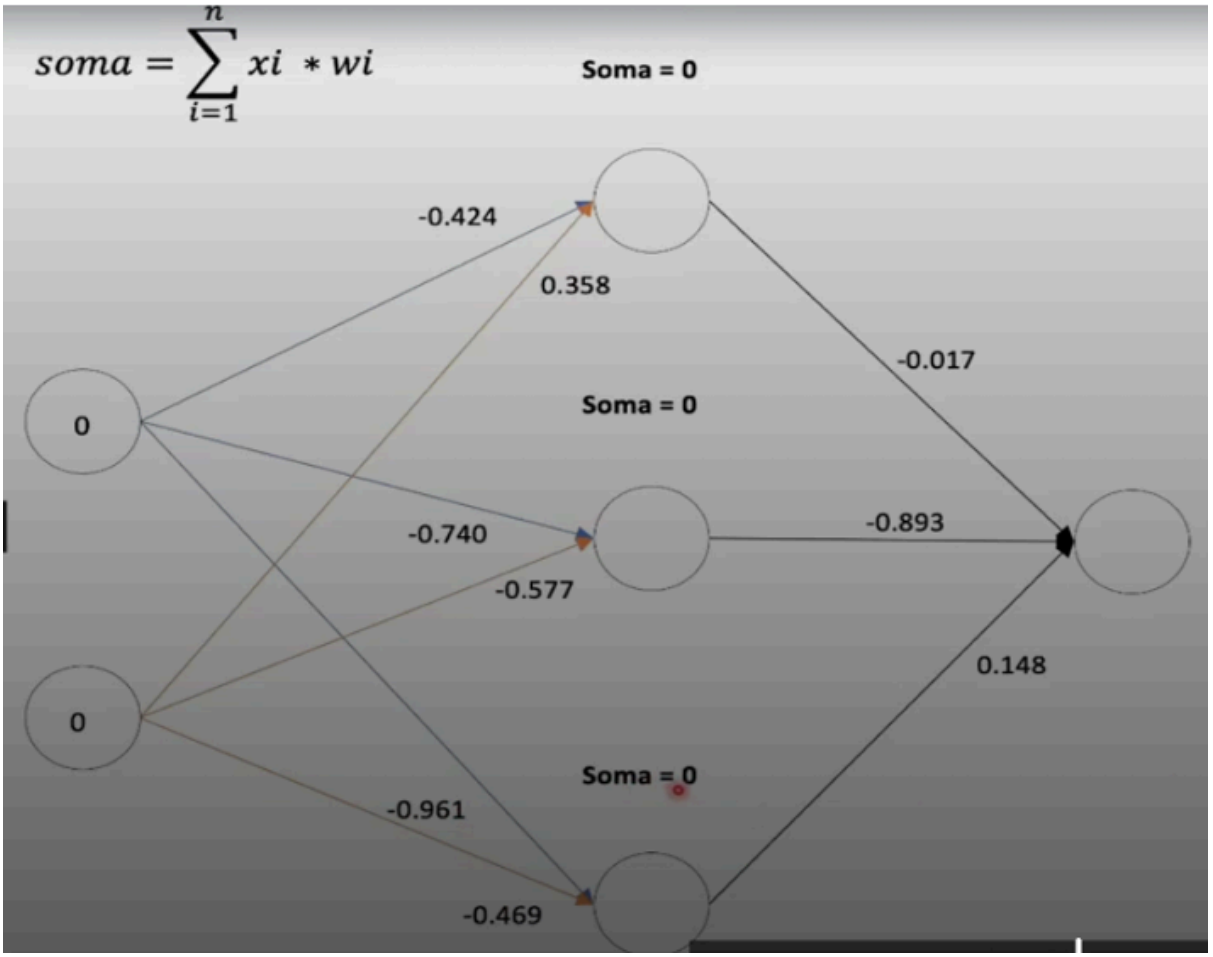


Os dendritos é onde as informações chegam, no corpo celular é onde essa informação é processada, axônio transmite a informação processada para os terminais do axônio, esses terminais se conectam a outros dendritos e fazendo uma junção de vários neurônios e essa conexão se chama de sinapse.

Aula 08: Foi nos ensinado como é a representação de um neurônio artificial e de como fazer os cálculos, o neurônio artificial é separado em 3 partes, de começo a entrada, após o peso para cada entrada e depois é a ativação de duas funções, uma função soma e uma função de ativação. É explicado que o **objetivo** de uma rede neural é **encontrar** os **melhores pesos** para as entradas com o intuito de **diminuir** os **erros**.

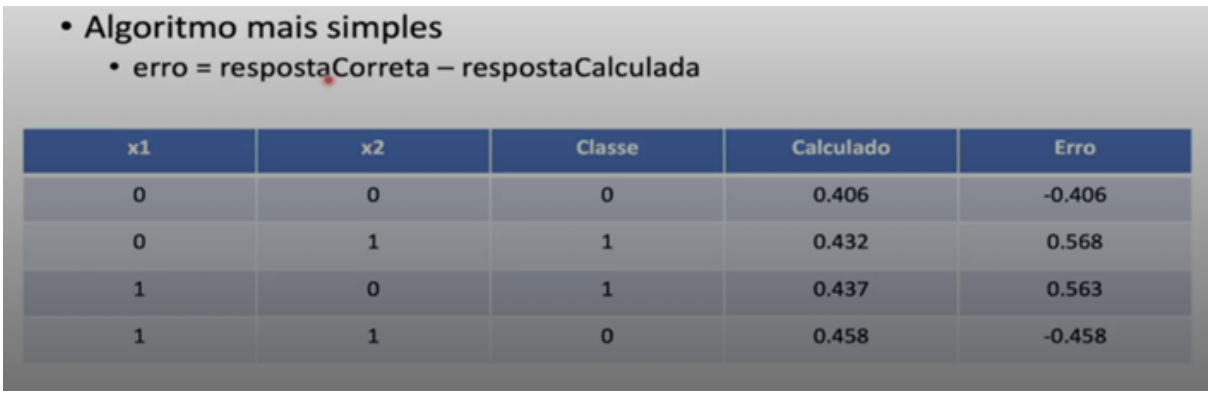
Aula 09 - redes neurais multicamadas - função soma e função de ativação Nessa aula é utilizado o operador ‘Xor’ para dar exemplo a rede neural de multicamada, onde o resultado é 1 quando as entradas são diferentes.

No meio da aula ele vai testando caso a caso,



A entrada é 0, é multiplicada pelo peso e passa para a camada oculta onde é aplicado a função de sigmóide para onde nesse caso a função retornará 0,5 para ambas entradas.

Aula 10 - Redes Multicamada - cálculo de erro:

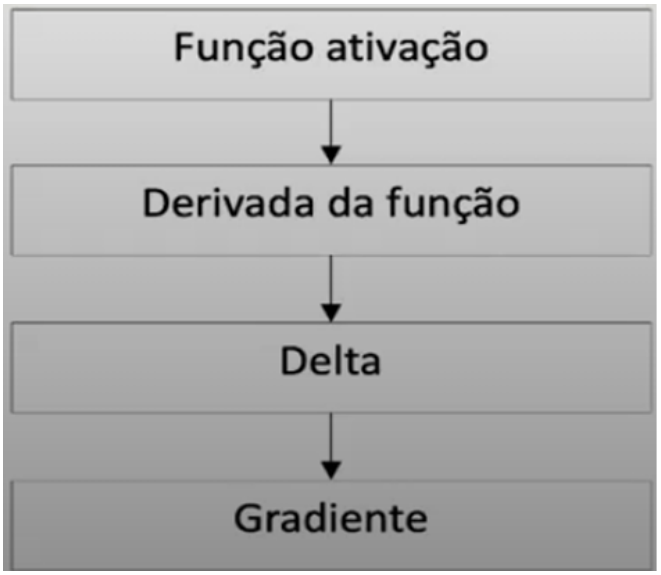


x1	x2	Classe	Calculado	Erro
0	0	0	0.406	-0.406
0	1	1	0.432	0.568
1	0	1	0.437	0.563
1	1	0	0.458	-0.458

Cálculo feito foi esse, do erro ser a resposta certa menos a resposta calculada, a média de erro é a média absoluta, onde o resultado é 0.49, lembrando que o objetivo é sempre ir alterando os pesos fazendo com que esse erro se aproxime cada vez mais de 0.

Aula 11 - Gradiente: Descida do gradiente é a técnica utilizada para atualizar os pesos com o objetivo de diminuir o erro, basicamente ele encontra a combinação de pesos onde o erro é o menor possível.

Aula 12 - Cálculo do delta: Esse cálculo do parâmetro delta é útil para saber a direção da atualização dos pesos.



Função de ativação, a partir disso a derivada do resultado e assim o delta e só assim para conseguirmos o gradiente dito da aula passada.

Primeira equação é a função de sigmoide e a segunda é a derivada utilizando o resultado obtido, esse cálculo para chegar no final do gradiente serve para conseguirmos chegar no mínimo global da função onde o erro vai ser o menor possível.

O delta é obtido com erro multiplicado com a derivada, é feito esse cálculo para cada registro do caso de estudo que é o XOR, porém esse cálculo é feito para as camadas de saída as camadas ocultas recebem outra fórmula.

O delta da camada oculta é obtido com a derivada multiplicada pelo peso e pelo delta da saída.

Aula 13: Ajuste dos pesos (Backpropagation): BackPropagation é o algoritmo que é utilizado nas redes neurais ajustando os pesos das conexões, é utilizado esse cálculo em cada camada oculta de cada registro:

$$Peso_{n+1} = (peso_n * momento) + (entrada * delta * taxa\ de\ aprendizagem)$$

É feito este mesmo processo para a camada de entrada, esse processo todo é feito para atualizar os pesos para a menor quantidade de erros, porém humanamente é complicado e isso passado em códigos com várias épocas é bem mais eficiente.
Aula 14: Bias, erro, descida do gradiente estocástico e mais parâmetros: Aqui vai ser mostrado alguns parâmetros adicionais das redes neurais, a primeira é a unidade de bias, é um neurônio especial adicionado a cada camada, para ajustar a ativação de outros neurônios, não é necessário se preocupar com isso pois as bibliotecas já fazem isso transparentemente.

existem formas mais complexas e melhores, MSE e RMSE, mse ou mean squared error é a média quadrada dos erros, tem essa fórmula:

$$\text{MSE} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{Mean}}\underbrace{(Y_i - \hat{Y}_i)^2}_{\text{Error}\quad\text{Squared}}$$

E o RMSE é a raiz quadrada dessa mesma conta, trazendo uma maior penalização dos erros, essa é a fórmula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}},$$

Existem duas maneiras de calcular a descida do gradiente, a padrão que já foi nos mostrada em aulas passadas onde é calculado o erro para todos os registros e assim atualizar os pesos, mas a descida do gradiente estocástica funciona de outra maneira, ela calcula o erro para cada registro e atualiza os pesos, a vantagem é que é mais rápido e ajuda a prevenir os mínimos locais dentro da função. E existe o mini batch gradient descent que escolhe um número e registros para rodar e atualizar os pesos.

Aula 24 validação cruzada
K-fold cross-validation é uma técnica de validação onde os dados são divididos em K partes iguais. Em cada iteração, uma parte é usada para teste, enquanto as outras K-1 partes são

usadas para treino. O processo é repetido K vezes, garantindo que cada parte seja usada como teste uma vez. No final, uma média das métricas de avaliação é calculada para medir o desempenho geral do modelo, exemplo dos conjuntos.

Aula 26: Overfitting e Underfitting:
Overfitting e Underfitting são problemas que acompanham toda essa área de machine learning, redes neurais e entre outras, esses problemas acontecem quando os dados se ajustam de mais ou de menos para modelos e se tornando não tão úteis e eficientes. **Overfitting** é quando o modelo ou a rede se ajusta demais aos dados incluindo outliers e outros, basicamente quando ele se torna muito próximo apenas aos dados de treinamento. **Underfitting** é exatamente o contrário quando ele é muito simples e não consegue aprender uma relação concreta, acontece quando os dados de treinamento são limitados. Porém existem correções para esses casos como K-Fold-Cross-Validation é feito para corrigir o Overfitting.

Aula 28 - Tuning dos parâmetros - Nessa aula também prática é mostrado como a própria rede neural pode nos mostrar os melhores parâmetros para aquele teste em específico, na aula é passado 2 tipos diferentes para cada parâmetro e é feito o teste, graças a uma biblioteca do sklearn ele consegue nos retornar quais parâmetros juntos tiveram a melhor taxa de acertos.

Seções 5,6,7:

Essas seções são apenas práticas utilizando dados diferentes em cada seçao,, e praticando o que aprendemos nas seções anteriores.

Validação cruzada do conjunto de dados de Flores Iris

```
[ ] import pandas as pd
    import tensorflow as tf
    import numpy as np
    import sklearn
    import scikeras
```

```
[ ] from scikeras.wrappers import KerasClassifier
    from sklearn.preprocessing import LabelEncoder
    from sklearn.model_selection import cross_val_score
    from tensorflow.keras.models import Sequential
    from tensorflow.keras import backend as k
    from tensorflow.keras import utils as np_utils
```

```
[ ] base = pd.read_csv('iris_aula.csv')
    x = base.iloc[:, 0:4].values
    y = base.iloc[:, 4].values
```

```
[ ] labelencoder = LabelEncoder()
    y = labelencoder.fit_transform(y)
    y = np_utils.to_categorical(y)
```

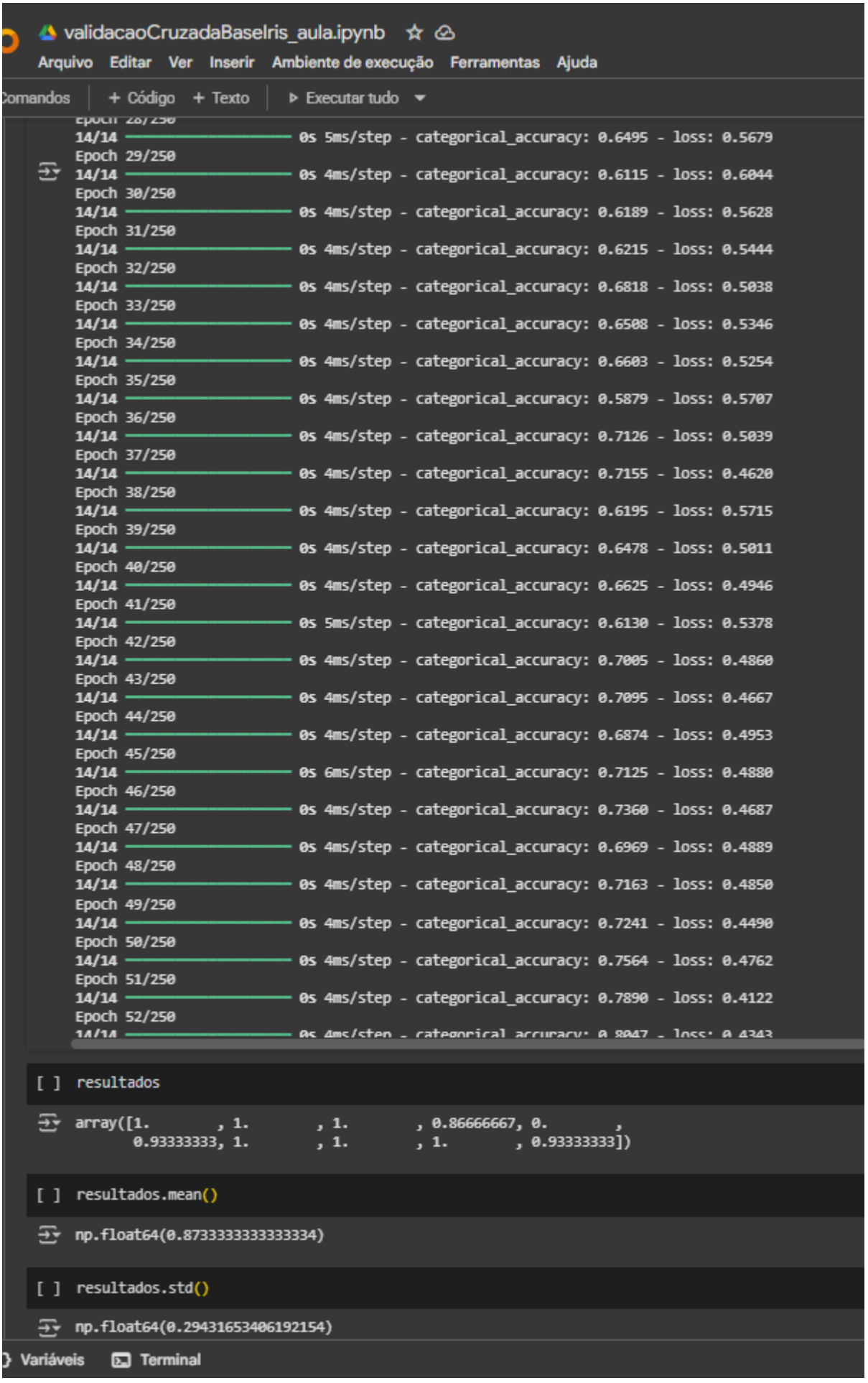
```
[ ] def criar_rede():
        k.clear_session()
        rede_neural = Sequential([
        tf.keras.layers.InputLayer(shape=(4,)),
        tf.keras.layers.Dense(units=4, activation="relu"),
        tf.keras.layers.Dense(units=4, activation="relu"),
        tf.keras.layers.Dense(units=3, activation="softmax")
    ])
        rede_neural.compile(optimizer="Adam", loss = "categorical_crossentropy", metrics = ["categorical_accuracy"])
        return rede_neural
```

```
[ ] rede_neural = KerasClassifier(model = criar_rede, epochs=250, batch_size = 10)
```

```
    resultados = cross_val_score(estimator=rede_neural, X=x,y=y, cv=10, scoring="accuracy")

Epoch 1/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - categorical_accuracy: 0.3735 - loss: 1.2601
Epoch 2/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - categorical_accuracy: 0.4241 - loss: 1.1446
Epoch 3/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - categorical_accuracy: 0.3998 - loss: 1.1339
Epoch 4/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - categorical_accuracy: 0.3881 - loss: 1.1596
Epoch 5/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - categorical_accuracy: 0.4354 - loss: 1.0502
Epoch 6/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - categorical_accuracy: 0.5749 - loss: 1.0032
Epoch 7/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - categorical_accuracy: 0.6616 - loss: 0.9205
Epoch 8/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - categorical_accuracy: 0.6786 - loss: 0.8219
Epoch 9/250
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - categorical_accuracy: 0.6062 - loss: 0.8421
Epoch 10/250
```

Variáveis Terminal


validacaoCruzadaBaselris_aula.ipynb
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Comandos + Código + Texto Executar tudo
14/14 0s 5ms/step - categorical_accuracy: 0.6495 - loss: 0.5679
Epoch 29/250
14/14 0s 4ms/step - categorical_accuracy: 0.6115 - loss: 0.6044
Epoch 30/250
14/14 0s 4ms/step - categorical_accuracy: 0.6189 - loss: 0.5628
Epoch 31/250
14/14 0s 4ms/step - categorical_accuracy: 0.6215 - loss: 0.5444
Epoch 32/250
14/14 0s 4ms/step - categorical_accuracy: 0.6818 - loss: 0.5038
Epoch 33/250
14/14 0s 4ms/step - categorical_accuracy: 0.6508 - loss: 0.5346
Epoch 34/250
14/14 0s 4ms/step - categorical_accuracy: 0.6603 - loss: 0.5254
Epoch 35/250
14/14 0s 4ms/step - categorical_accuracy: 0.5879 - loss: 0.5707
Epoch 36/250
14/14 0s 4ms/step - categorical_accuracy: 0.7126 - loss: 0.5039
Epoch 37/250
14/14 0s 4ms/step - categorical_accuracy: 0.7155 - loss: 0.4620
Epoch 38/250
14/14 0s 4ms/step - categorical_accuracy: 0.6195 - loss: 0.5715
Epoch 39/250
14/14 0s 4ms/step - categorical_accuracy: 0.6478 - loss: 0.5011
Epoch 40/250
14/14 0s 4ms/step - categorical_accuracy: 0.6625 - loss: 0.4946
Epoch 41/250
14/14 0s 5ms/step - categorical_accuracy: 0.6130 - loss: 0.5378
Epoch 42/250
14/14 0s 4ms/step - categorical_accuracy: 0.7005 - loss: 0.4860
Epoch 43/250
14/14 0s 4ms/step - categorical_accuracy: 0.7095 - loss: 0.4667
Epoch 44/250
14/14 0s 4ms/step - categorical_accuracy: 0.6874 - loss: 0.4953
Epoch 45/250
14/14 0s 6ms/step - categorical_accuracy: 0.7125 - loss: 0.4880
Epoch 46/250
14/14 0s 4ms/step - categorical_accuracy: 0.7360 - loss: 0.4687
Epoch 47/250
14/14 0s 4ms/step - categorical_accuracy: 0.6969 - loss: 0.4889
Epoch 48/250
14/14 0s 4ms/step - categorical_accuracy: 0.7163 - loss: 0.4850
Epoch 49/250
14/14 0s 4ms/step - categorical_accuracy: 0.7241 - loss: 0.4490
Epoch 50/250
14/14 0s 4ms/step - categorical_accuracy: 0.7564 - loss: 0.4762
Epoch 51/250
14/14 0s 4ms/step - categorical_accuracy: 0.7890 - loss: 0.4122
Epoch 52/250
[ ] resultados
array([1. , 1. , 1. , 0.86666667, 0. ,
0.93333333, 1. , 1. , 1. , 0.93333333])
[ ] resultados.mean()
np.float64(0.8733333333333334)
[ ] resultados.std()
np.float64(0.29431653406192154)
Variáveis Terminal

Validação cruzada do conjunto de dados de carros

```
[ ] import pandas as pd
    import tensorflow as tf
    import sklearn
    import scikeras
```

```
[ ] import time
    from tensorflow.keras import backend as K
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from sklearn.model_selection import cross_val_score
    from sklearn.preprocessing import OneHotEncoder
    from sklearn.compose import ColumnTransformer
    from sklearn import metrics
    from scikeras.wrappers import KerasRegressor
```

```
[ ] inicio = time.time()
```

```
[ ] inicio
```

```
1752604326.4410048
```

```
[ ] base = pd.read_csv("carro.csv", encoding='Latin1')
```

```
[ ] base = base.drop('dateCrawled', axis=1)
    base = base.drop('dateCreated', axis=1)
    base = base.drop('nrOfPictures', axis=1)
    base =base.drop('postalCode', axis=1)
    base = base.drop("lastSeen", axis=1)
    base = base.drop('name', axis=1)
    base = base.drop('seller', axis=1)
    base = base.drop('offerType', axis=1)
```

```
[ ] base = base[base.price > 10]
    base = base.loc[base.price < 350000]
```

```
[ ] valores = {'vehicleType': 'limousine', 'gearbox': 'manuell', 'model': 'golf', 'fuelType': 'benzin', 'notRepairedDamage': 'nein'}
    base = base.fillna(value=valores)
```

```
[ ] X = base.iloc[:, 1:12]
    Y = base.iloc[:, 0]
```

```
[ ] onehotencoder = ColumnTransformer(transformers=[("OneHot", OneHotEncoder(), [0,1,3,5,8,9,10])], remainder='passthrough')
    X = onehotencoder.fit_transform(X).toarray()
```

```
[ ] X.shape
```

```
(359291, 316)
```

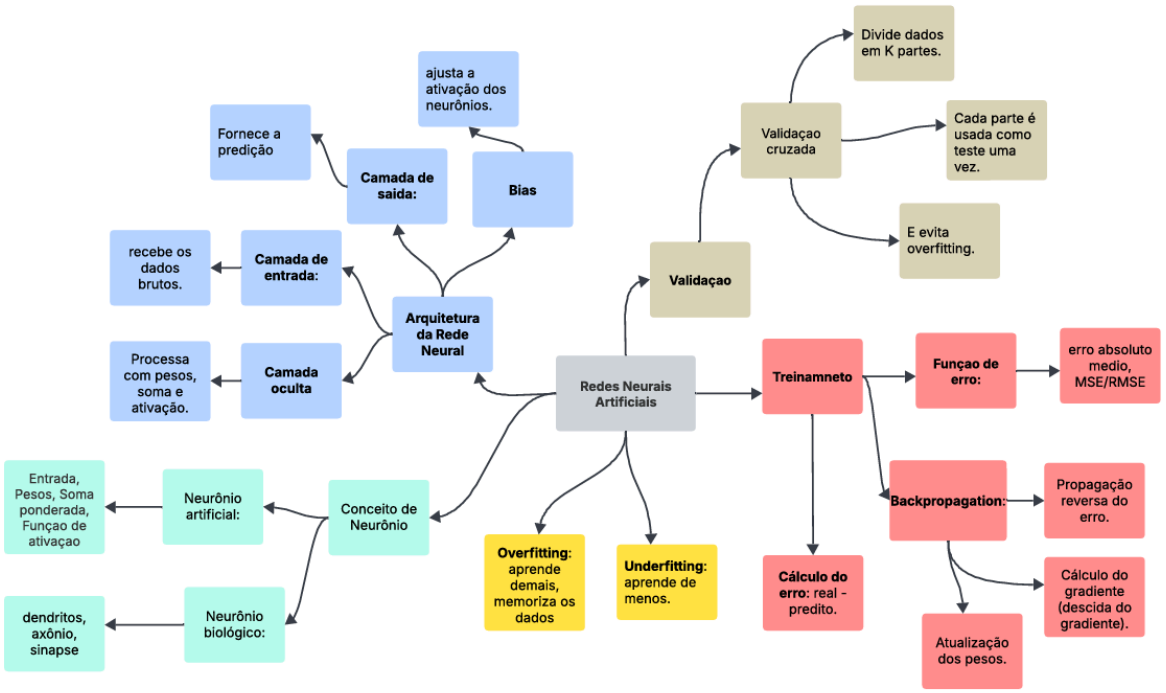
```
[12] def criar_rede():
         from tensorflow.keras import backend as K
         K.clear_session()
         regressor = Sequential([
           tf.keras.layers.InputLayer(shape = (316,)),
           tf.keras.layers.Dense(units = 158, activation = 'relu'),
           tf.keras.layers.Dense(units = 158, activation = 'relu'),
           tf.keras.layers.Dense(units = 1, activation = 'linear'),
     ])
         regressor.compile(loss = 'mean_absolute_error', optimizer = 'adam', metrics = ['mean_absolute_error'])
         return regressor
```

```
[13] regressor = KerasRegressor(model = criar_rede, epochs= 50, batch_size = 300)
```

```
[14] resultados = cross_val_score(estimator = regressor, X = X, y = Y, cv = 5, scoring = 'neg_mean_absolute_error')
```

```
Epoch 7/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 2767.5513 - mean_absolute_error: 2767.5513
Epoch 8/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - loss: 2731.2263 - mean_absolute_error: 2731.2261
Epoch 9/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - loss: 2702.1035 - mean_absolute_error: 2702.1035
Epoch 10/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 2652.0610 - mean_absolute_error: 2652.0610
Epoch 11/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - loss: 2639.6272 - mean_absolute_error: 2639.6272
Epoch 12/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - loss: 2644.9443 - mean_absolute_error: 2644.9443
Epoch 13/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 2565.4172 - mean_absolute_error: 2565.4172
Epoch 14/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - loss: 2550.5227 - mean_absolute_error: 2550.5227
Epoch 15/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - loss: 2544.3953 - mean_absolute_error: 2544.3953
Epoch 16/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 2485.8999 - mean_absolute_error: 2485.8999
Epoch 17/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - loss: 2489.8718 - mean_absolute_error: 2489.8718
Epoch 18/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 2466.6514 - mean_absolute_error: 2466.6514
Epoch 19/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - loss: 2469.1179 - mean_absolute_error: 2469.1179
Epoch 20/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - loss: 2443.1313 - mean_absolute_error: 2443.1313
Epoch 21/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - loss: 2430.5698 - mean_absolute_error: 2430.5698
Epoch 22/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - loss: 2424.8459 - mean_absolute_error: 2424.8459
Epoch 23/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - loss: 2430.8816 - mean_absolute_error: 2430.8816
Epoch 24/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - loss: 2414.5527 - mean_absolute_error: 2414.5527
Epoch 25/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - loss: 2403.8118 - mean_absolute_error: 2403.8118
Epoch 26/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - loss: 2404.2900 - mean_absolute_error: 2404.2900
Epoch 27/50
959/959 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 2415.6965 - mean_absolute_error: 2415.6965
```

Vou colocar agora um insight visual e Original sobre tudo que aprendi nesse card - 13:



Overfitting e Underfitting: Overfitting a rede aprende demais os detalhes do treino, inclusive os ruídos, e perde capacidade de generalização. Underfitting a rede não aprende o suficiente, tem baixa performance tanto no treino quanto no teste.

K-Fold-Cross-Validation: Divide os dados em K partes; treina em K – 1 e testa na parte restante, repetindo o processo K vezes.Ajuda a avaliar o desempenho do modelo de forma mais confiável e evitar overfitting.

Camada de entrada: Recebe os dados brutos e os repassa para a próxima camada sem processamento.

Camada Intermediária: Realiza o processamento principal usando pesos, soma, ativação e aprende padrões complexos.

Camada Oculta: Gera a resposta final da rede, como uma classe ou valor, com ativação apropriada para a tarefa.

Função de soma: Combina os valores de entrada multiplicados por seus respectivos pesos e soma com o bias

Função de ativação: Aplicada após cada neurônio, transforma a saída linear em uma forma não-linear para aprender padrões complexos.

Backpropagation: Propaga o erro da saída para trás, atualizando os pesos com base nos deltas para reduzir o erro da rede.

Cálculo de erro: Compara a saída da rede com o valor real, gerando um valor escalar que representa o quão errada está a predição.

Conclusão

As redes neurais são ferramentas extremamente úteis para realizar previsões de dados. Quando há o conhecimento adequado para modelar corretamente e realizar o pré-processamento dos dados e parâmetros.

Referência:

Bootcamp lamia - Card 13