Aula 6

Inicialização de RNAs

Eduardo Lobo Lustosa Cabral

1. Objetivos

- Apresentar formas de inicializar parâmetros de uma RNA.
- Apresentar algumas formas de transfromação e normalização de dados.
- Verificar o que acontece se os pesos das ligações são inicializados com constantes.

Importação das principais de bibliotecas

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
tf.__version__
```

{"type":"string"}

2. Carregar e processar dados

Nesse exemplo vamos usar o conjunto de dados de doença cardiaca "Heart Disease UCI" (https://www.kaggle.com/ronitf/heart-disease-ucipontos).

2.1 Carregar dados de entrada

No código da célula abaixo é importado o conjunto de dados.

```
df_orig = pd.read_csv('heart.csv')
df_orig
```

{"summary":"{\n \"name\": \"df_orig\",\n \"rows\": 303,\n \"fields\": [\n   {\n   \"column\": \"age\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 9,\n   \"min\": 29,\n   \"max\": 77,\n   \"num_unique_values\": 41,\n   \"samples\": [\n   46,\n   66,\n   48\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"sex\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

```
\"std\": 0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 3,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          2,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trestbps\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 17,\n        \"min\": 94,\n
\"max\": 200,\n        \"num_unique_values\": 49,\n
\"samples\": [\n          104,\n          123\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"chol\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 51,\n        \"min\": 126,\n
\"max\": 564,\n        \"num_unique_values\": 152,\n
\"samples\": [\n          277,\n          169\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"fbs\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"restecg\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"thalach\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 22,\n        \"min\": 71,\n
\"max\": 202,\n        \"num_unique_values\": 91,\n
\"samples\": [\n          159,\n          152\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"exang\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0,\n        \"min\":
0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"oldpeak\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.1610750220686343,\n        \"min\": 0.0,\n        \"max\": 6.2,\n
\"num_unique_values\": 40,\n        \"samples\": [\n          1.9,\n
3.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"slope\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 2,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          0,\n
2\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ca\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 4,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          2,\n
4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"thal\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 3,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          2,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"target\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_orig"}
```


2.2 Calcular estatísticas báscas dos dados

```
df_orig.describe().T
```


```
{"summary":"{\n  \"name\": \"df_orig\",\n  \"rows\": 14,\n
\"fields\": [\n    {\n      \"column\": \"count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 303.0,\n        \"max\": 303.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          303.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"mean\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
77.66715456234161,\n        \"min\": 0.1485148514851485,\n
\"max\": 246.26402640264027,\n        \"num_unique_values\": 14,\n
\"samples\": [\n          1.0396039603960396\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"std\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 14.601099749666268,\n
\"min\": 0.35619787492797594,\n        \"max\": 51.830750987930045,\n
\"num_unique_values\": 14,\n        \"samples\": [\n
1.1610750220686343\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"min\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 42.31722170402085,\n        \"min\": 0.0,\n        \"max\":
126.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"25%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 67.9705010352948,\n        \"min\": 0.0,\n        \"max\":
211.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\n
47.5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"50%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 76.67719262030835,\n        \"min\": 0.0,\n        \"max\":
240.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"75%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 85.7535867317667,\n        \"min\": 0.0,\n        \"max\":
274.5,\n        \"num_unique_values\": 9,\n        \"samples\": [\n
1.6\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"max\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 157.76377388883787,\n        \"min\": 1.0,\n        \"max\":
564.0,\n        \"num_unique_values\": 9,\n        \"samples\": [\n
6.2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


2.3 Transformação dos dados

É necessário transformar colunas com valores categóricos para vetores one_hot.

É conveniente fazer essa transformação antes da divisão dos dados em conjuntos de treinamento e validação/teste.

As colunas com variáveis categóricas são as seguintes:

- cp
- restecg
- slope
- ca
- thal

Primeiramente vamos verifcar os valores únicos dessas colunas.

```
df_orig.head()
```


```
{"summary":"{\n  \"name\": \"df_orig\",\n  \"rows\": 303,\n
\"fields\": [\n    {\n      \"column\": \"age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9,\n        \"min\": 29,\n        \"max\": 77,\n
\"num_unique_values\": 41,\n        \"samples\": [\n          46,\n
66,\n          48\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sex\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 3,\n
```

```
\"num_unique_values\": 4,\n        \"samples\": [\n          2,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trestbps\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 17,\n        \"min\": 94,\n
\"max\": 200,\n        \"num_unique_values\": 49,\n
\"samples\": [\n          104,\n          123\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"chol\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 51,\n        \"min\": 126,\n
\"max\": 564,\n        \"num_unique_values\": 152,\n
\"samples\": [\n          277,\n          169\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"fbs\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"restecg\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"thalach\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 22,\n        \"min\": 71,\n
\"max\": 202,\n        \"num_unique_values\": 91,\n
\"samples\": [\n          159,\n          152\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"exang\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0,\n        \"min\":
0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"oldpeak\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.1610750220686343,\n        \"min\": 0.0,\n        \"max\": 6.2,\n
\"num_unique_values\": 40,\n        \"samples\": [\n          1.9,\n
3.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"slope\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 2,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          0,\n
2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ca\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 4,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          2,\n
4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"thal\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 3,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          2,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"target\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_orig"}
```


```python
# Valores únicos das colunas
print('Valore únicos de cp:', df_orig.cp.unique())
print('Valore únicos de restecg:',df_orig.restecg.unique())
print('Valore únicos de slope:', df_orig.slope.unique())
print('Valore únicos de ca:', df_orig.ca.unique())
print('Valore únicos de thal:',df_orig.thal.unique())
```

```
Valore únicos de cp: [3 2 1 0]
Valore únicos de restecg: [0 1 2]
Valore únicos de slope: [0 2 1]
Valore únicos de ca: [0 2 1 3 4]
Valore únicos de thal: [1 2 3 0]
```

```python
# Salva dataframe original
df = df_orig.copy()

# Codificação one-hot da coluna cp
df_cp_hot = pd.get_dummies(df["cp"], dtype=int, prefix='cp')

# Codificação one-hot da coluna restecg
df_rest_hot = pd.get_dummies(df["restecg"],
dtype=int,prefix='restecg')

# Codificação one-hot da coluna slope
df_slope_hot = pd.get_dummies(df["slope"], dtype=int,prefix='slope')

# Codificação one-hot da coluna cae
df_ca_hot = pd.get_dummies(df["ca"], dtype=int,prefix='ca')

# Codificação one-hot da coluna thal
df_thal_hot = pd.get_dummies(df["thal"], dtype=int,prefix='thal')

# União do resultado das codificações one-hot com dataframe original
df = df.join(df_cp_hot)
df = df.join(df_rest_hot)
df = df.join(df_slope_hot)
df = df.join(df_ca_hot)
df = df.join(df_thal_hot)
```

```
# Remoção das colunas originais
df = df.drop(columns=['cp'])
df = df.drop(columns=['restecg'])
df = df.drop(columns=['slope'])
df = df.drop(columns=['ca'])
df = df.drop(columns=['thal'])

df.head(10)
```

{"type":"dataframe","variable_name":"df"}

2.4 Divisão do conjunto de dados

Vamos dividir o conjunto de dados em conjuntos de treinamento e teste.

```
# Usaremos a função split da biblioteca sklearn para divir os dados
train_df, test_df = train_test_split(df, test_size=0.2, shuffle=True)

# Separa as saídas dos dados de entrada e as transforma em tensores
Numpy
Y_train = np.array(train_df.pop('target'))
Y_test = np.array(test_df.pop('target'))
```

2.5 Normalização dos dados

Vamos normalizar as colunas com valores reais, ou seja, as colunas `age`, `trestbps`, `thalach`, `chol` e `oldpeak`.

A normalização será realizada para as colunas tenham média 0 e desvio padraõa igual a 1.

Nessa normalização devemos usar os valores médios e desvios padrões somente dos dados de treinamento.

```
train_df.head()
```

{"type":"dataframe","variable_name":"train_df"}

```
# Calculo das médias e desvios padrões
mean_age = train_df['age'].mean()
std_age = train_df['age'].std()
mean_tres = train_df['trestbps'].mean()
std_tres = train_df['trestbps'].std()
mean_thal = train_df['thalach'].mean()
std_thal = train_df['thalach'].std()
mean_col = train_df['chol'].mean()
std_col = train_df['chol'].std()
mean_oldpeak = train_df['oldpeak'].mean()
std_oldpeak = train_df['oldpeak'].std()
```

```
# Normalização dos dados de treinamento
train_df['age'] = (train_df['age'] - mean_age)/std_age
train_df['trestbps'] = (train_df['trestbps'] - mean_tres)/std_tres
train_df['thalach'] = (train_df['thalach'] - mean_thal)/std_thal
train_df['chol'] = (train_df['chol'] - mean_col)/std_col
train_df['oldpeak'] = (train_df['oldpeak'] - mean_oldpeak)/std_oldpeak
train_df.head()
```

{"type":"dataframe","variable_name":"train_df"}

```
# Normalização dos dados de testeo
test_df['age'] = (test_df['age'] - mean_age)/std_age
test_df['trestbps'] = (test_df['trestbps'] - mean_tres)/std_tres
test_df['thalach'] = (test_df['thalach'] - mean_thal)/std_thal
test_df['chol'] = (test_df['chol'] - mean_col)/std_col
test_df['oldpeak'] = (test_df['oldpeak'] - mean_oldpeak)/std_oldpeak
test_df.head()
```

{"type":"dataframe","variable_name":"test_df"}

```
# Transforma os dados de entrada em tensores Numpy
X_train = np.array(train_df)
X_test = np.array(test_df)
```

```
train_df.describe().T
```

{"summary":"{\n \"name\": \"train_df\",\n \"rows\": 27,\n \"fields\": [\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 242.0,\n   \"max\": 242.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   242.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.22624080503102265,\n   \"min\": -5.982358777319025e-16,\n   \"max\": 0.6735537190082644,\n   \"num_unique_values\": 25,\n   \"samples\": [\n   0.4834710743801653\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"std\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.28163179406781347,\n   \"min\": 0.0907202867781661,\n   \"max\": 1.0000000000000002,\n   \"num_unique_values\": 25,\n   \"samples\": [\n   0.36457818433742056\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"min\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.9860426779994153,\n   \"min\": -3.3829827767849756,\n   \"max\": 0.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   -2.7390981164444015\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"25%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.30440790060188266,\n

\"min\": -0.9084871715455991,\n   \"max\": 0.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   -0.7917321306060121\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"50%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.337181576008897,\n   \"min\": -0.2206376744348131,\n   \"max\": 1.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"75%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.46566319211419227,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   0.722885858379402\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"max\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.9800613982650335,\n   \"min\": 1.0,\n   \"max\": 4.422346431062993,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   2.453877845791304\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
test_df.describe().T
```

{"summary":"{\n \"name\": \"test_df\",\n \"rows\": 27,\n \"fields\": [\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 61.0,\n   \"max\": 61.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   61.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.2201154893665662,\n   \"min\": -0.07263727034335299,\n   \"max\": 0.7213114754098361,\n   \"num_unique_values\": 23,\n   \"samples\": [\n   0.4098360655737705\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"std\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.28924777215934094,\n   \"min\": 0.0,\n   \"max\": 1.2713891511652526,\n   \"num_unique_values\": 23,\n   \"samples\": [\n   0.4958847036804649\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"min\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.7456692422374843,\n   \"min\": -2.349565432464398,\n   \"max\": 0.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   -1.5490411250987193\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"25%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.2773437391301275,\n   \"min\": -0.9084871715455991,\n   \"max\": 0.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   -

```
0.6835451313927683\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"50%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.3767933272700339,\n        \"min\": -0.39260004871250964,\n
\"max\": 1.0,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          -0.034423136113305074\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"75%\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.4643242054542464,\n
\"min\": 0.0,\n        \"max\": 1.0,\n        \"num_unique_values\":
7,\n        \"samples\": [\n          0.6146988591661581\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"max\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.1737848787268528,\n
\"min\": 0.0,\n        \"max\": 6.516865697797663,\n
\"num_unique_values\": 7,\n        \"samples\": [\n
1.8047558505118406\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


3. Configuração da rede

A inicialização dos parâmetros de um RNA é realizada por camadas no momento em que ela é incluída na RNA.

Vamos verificar o que ocorre no treinamento de uma rede quando inicilizarmos os parâmetros de diferentes formas.

Os inicializadores existentes do Keras podem ser vistos em https://www.tensorflow.org/api_docs/python/tf/keras/initializers.

```
# Importar do Keras classes de modelos e camadas
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras import initializers

# Define dimensão da entrada
xdim = X_train.shape[1]

# Definiação da inicialização dos pessos das ligações
initializer = tf.keras.initializers.Constant(1.0)
#initializer = tf.keras.initializers.RandomUniform(minval=-0.001,
maxval=0.001, seed=None)
#initializer = tf.keras.initializers.GlorotNormal()

# Configuração da rede com pesos e bias inicilizados com zeros
rna = Sequential()
rna.add(Dense(units=64, activation='relu',
kernel_initializer=initializer, bias_initializer='zeros',
input_dim=xdim))
```

```
rna.add(Dense(units=32, activation='relu',
kernel_initializer=initializer, bias_initializer='zeros'))
rna.add(Dense(units=1, activation='sigmoid',
kernel_initializer=initializer, bias_initializer='zeros'))

rna.summary()
```

```
Model: "sequential_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ dense_9 (Dense)                       │ (None, 64)                  │
1,792 │
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_10 (Dense)                      │ (None, 32)                  │
2,080 │
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_11 (Dense)                      │ (None, 1)                   │
33 │
└───────────────────────────────────────┴─────────────────────────────┴
─────────────────┘

 Total params: 3,905 (15.25 KB)

 Trainable params: 3,905 (15.25 KB)

 Non-trainable params: 0 (0.00 B)
```

4. Compilação e treinamento da rede

```
EPOCHS = 100

# Compilação da rede
rna.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])

# Treinamento da RN
history = rna.fit(X_train, Y_train, epochs=EPOCHS,
validation_data=(X_test, Y_test), verbose=1)
```

```
Epoch 1/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 100ms/step - accuracy: 0.5150 - loss:
4548.4229 - val_accuracy: 0.5082 - val_loss: 1582.6173
Epoch 2/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5135 - loss:
```

```
1011.5787 - val_accuracy: 0.5246 - val_loss: 43.0796
Epoch 3/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6132 - loss:
332.8896 - val_accuracy: 0.6066 - val_loss: 173.9747
Epoch 4/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6201 - loss:
204.1907 - val_accuracy: 0.5246 - val_loss: 426.4799
Epoch 5/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5752 - loss:
275.6804 - val_accuracy: 0.6721 - val_loss: 86.1200
Epoch 6/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6219 - loss:
88.0828 - val_accuracy: 0.6066 - val_loss: 101.1573
Epoch 7/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7207 - loss:
37.1928 - val_accuracy: 0.6066 - val_loss: 141.0809
Epoch 8/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7391 - loss:
33.3002 - val_accuracy: 0.5246 - val_loss: 29.9133
Epoch 9/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6681 - loss:
46.4276 - val_accuracy: 0.6066 - val_loss: 4.9514
Epoch 10/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6541 - loss:
37.5560 - val_accuracy: 0.5902 - val_loss: 74.0695
Epoch 11/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6456 - loss:
31.0067 - val_accuracy: 0.5902 - val_loss: 45.9961
Epoch 12/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7155 - loss:
21.5138 - val_accuracy: 0.5902 - val_loss: 78.2879
Epoch 13/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7523 - loss:
15.2643 - val_accuracy: 0.6230 - val_loss: 13.5870
Epoch 14/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6798 - loss: 9.6843
- val_accuracy: 0.6066 - val_loss: 34.7062
Epoch 15/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6856 - loss:
12.7323 - val_accuracy: 0.5246 - val_loss: 14.7456
Epoch 16/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6635 - loss:
24.6170 - val_accuracy: 0.5246 - val_loss: 9.7715
Epoch 17/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6169 - loss:
10.3727 - val_accuracy: 0.6066 - val_loss: 74.9204
Epoch 18/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7116 - loss:
35.1174 - val_accuracy: 0.6066 - val_loss: 7.9748
```

```
Epoch 19/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7194 - loss: 5.1339
- val_accuracy: 0.6066 - val_loss: 12.1038
Epoch 20/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6860 - loss: 9.9295
- val_accuracy: 0.5902 - val_loss: 64.8825
Epoch 21/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7176 - loss:
14.5264 - val_accuracy: 0.5902 - val_loss: 24.2866
Epoch 22/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6554 - loss: 6.4880
- val_accuracy: 0.6066 - val_loss: 17.9524
Epoch 23/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7049 - loss: 4.3251
- val_accuracy: 0.6066 - val_loss: 24.3612
Epoch 24/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7037 - loss: 3.6913
- val_accuracy: 0.6066 - val_loss: 30.8083
Epoch 25/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7118 - loss: 6.7404
- val_accuracy: 0.5902 - val_loss: 4.5554
Epoch 26/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6804 - loss: 2.4013
- val_accuracy: 0.5902 - val_loss: 15.5900
Epoch 27/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7175 - loss: 2.0173
- val_accuracy: 0.5246 - val_loss: 12.4075
Epoch 28/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6357 - loss: 5.9392
- val_accuracy: 0.5246 - val_loss: 0.9851
Epoch 29/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6263 - loss: 2.2455
- val_accuracy: 0.5246 - val_loss: 1.5793
Epoch 30/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6598 - loss: 2.9354
- val_accuracy: 0.5902 - val_loss: 11.7056
Epoch 31/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7041 - loss: 1.8461
- val_accuracy: 0.5246 - val_loss: 16.1729
Epoch 32/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6714 - loss:
10.1418 - val_accuracy: 0.5902 - val_loss: 10.0808
Epoch 33/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7278 - loss: 2.6363
- val_accuracy: 0.5902 - val_loss: 16.7464
Epoch 34/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6978 - loss: 3.6964
- val_accuracy: 0.5902 - val_loss: 2.4475
Epoch 35/100
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6429 - loss: 3.6342
- val_accuracy: 0.5902 - val_loss: 10.4426
Epoch 36/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6792 - loss: 1.6129
- val_accuracy: 0.5902 - val_loss: 7.6530
Epoch 37/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7035 - loss: 2.3761
- val_accuracy: 0.5902 - val_loss: 15.3111
Epoch 38/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7040 - loss: 1.7779
- val_accuracy: 0.5246 - val_loss: 2.7148
Epoch 39/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6230 - loss: 1.3013
- val_accuracy: 0.5902 - val_loss: 13.9820
Epoch 40/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6858 - loss: 2.5520
- val_accuracy: 0.5902 - val_loss: 2.5250
Epoch 41/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6609 - loss: 1.5046
- val_accuracy: 0.5902 - val_loss: 3.1267
Epoch 42/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6451 - loss: 2.3482
- val_accuracy: 0.5902 - val_loss: 10.7332
Epoch 43/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6818 - loss: 4.0222
- val_accuracy: 0.5902 - val_loss: 3.6894
Epoch 44/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6957 - loss: 1.8878
- val_accuracy: 0.5902 - val_loss: 10.5165
Epoch 45/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6891 - loss: 2.2109
- val_accuracy: 0.5902 - val_loss: 15.8162
Epoch 46/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6809 - loss: 2.2628
- val_accuracy: 0.5902 - val_loss: 5.5869
Epoch 47/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6879 - loss: 1.4673
- val_accuracy: 0.5902 - val_loss: 16.2393
Epoch 48/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6699 - loss: 3.7429
- val_accuracy: 0.5902 - val_loss: 6.5915
Epoch 49/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6477 - loss: 1.7489
- val_accuracy: 0.5902 - val_loss: 19.6578
Epoch 50/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6927 - loss: 3.4476
- val_accuracy: 0.5902 - val_loss: 10.3882
Epoch 51/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6651 - loss: 2.3622
```

```
- val_accuracy: 0.5246 - val_loss: 4.3799
Epoch 52/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6290 - loss: 4.5466
- val_accuracy: 0.5902 - val_loss: 9.8467
Epoch 53/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6767 - loss: 1.7572
- val_accuracy: 0.5902 - val_loss: 1.5420
Epoch 54/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6302 - loss: 2.6696
- val_accuracy: 0.5902 - val_loss: 15.9038
Epoch 55/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6781 - loss: 1.7914
- val_accuracy: 0.5902 - val_loss: 7.8410
Epoch 56/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6399 - loss: 1.5833
- val_accuracy: 0.5902 - val_loss: 6.7289
Epoch 57/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6762 - loss: 1.7538
- val_accuracy: 0.5902 - val_loss: 2.2851
Epoch 58/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6840 - loss: 0.8025
- val_accuracy: 0.5902 - val_loss: 3.5461
Epoch 59/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6534 - loss: 0.9276
- val_accuracy: 0.5902 - val_loss: 6.8134
Epoch 60/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6656 - loss: 1.6916
- val_accuracy: 0.5902 - val_loss: 10.4776
Epoch 61/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7036 - loss: 1.3875
- val_accuracy: 0.5902 - val_loss: 3.1416
Epoch 62/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6928 - loss: 0.8015
- val_accuracy: 0.5902 - val_loss: 2.8919
Epoch 63/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6008 - loss: 1.8072
- val_accuracy: 0.5902 - val_loss: 6.2452
Epoch 64/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6761 - loss: 1.0772
- val_accuracy: 0.5902 - val_loss: 8.1379
Epoch 65/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6935 - loss: 0.9677
- val_accuracy: 0.5246 - val_loss: 2.5111
Epoch 66/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5890 - loss: 1.9890
- val_accuracy: 0.5902 - val_loss: 2.3372
Epoch 67/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6149 - loss: 2.1924
- val_accuracy: 0.5902 - val_loss: 6.7150
```

```
Epoch 68/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7045 - loss: 1.7697
- val_accuracy: 0.5246 - val_loss: 0.8607
Epoch 69/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6047 - loss: 3.6536
- val_accuracy: 0.5902 - val_loss: 12.8002
Epoch 70/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6901 - loss: 2.0372
- val_accuracy: 0.5902 - val_loss: 6.6183
Epoch 71/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6347 - loss: 0.8402
- val_accuracy: 0.5246 - val_loss: 2.2420
Epoch 72/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6301 - loss: 1.4615
- val_accuracy: 0.5902 - val_loss: 4.0918
Epoch 73/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6162 - loss: 1.3165
- val_accuracy: 0.5902 - val_loss: 5.4741
Epoch 74/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6329 - loss: 0.9078
- val_accuracy: 0.5902 - val_loss: 21.9770
Epoch 75/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6672 - loss: 5.7174
- val_accuracy: 0.5902 - val_loss: 15.9142
Epoch 76/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6736 - loss: 4.5228
- val_accuracy: 0.5738 - val_loss: 10.6107
Epoch 77/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6599 - loss: 1.4538
- val_accuracy: 0.5738 - val_loss: 5.6631
Epoch 78/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6590 - loss: 0.8487
- val_accuracy: 0.5738 - val_loss: 1.1015
Epoch 79/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6405 - loss: 0.9296
- val_accuracy: 0.5738 - val_loss: 5.7446
Epoch 80/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6485 - loss: 1.2883
- val_accuracy: 0.5738 - val_loss: 1.3920
Epoch 81/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6088 - loss: 0.8962
- val_accuracy: 0.5738 - val_loss: 8.7472
Epoch 82/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6154 - loss: 2.4896
- val_accuracy: 0.5738 - val_loss: 4.1708
Epoch 83/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6590 - loss: 0.8515
- val_accuracy: 0.5246 - val_loss: 6.1672
Epoch 84/100
```

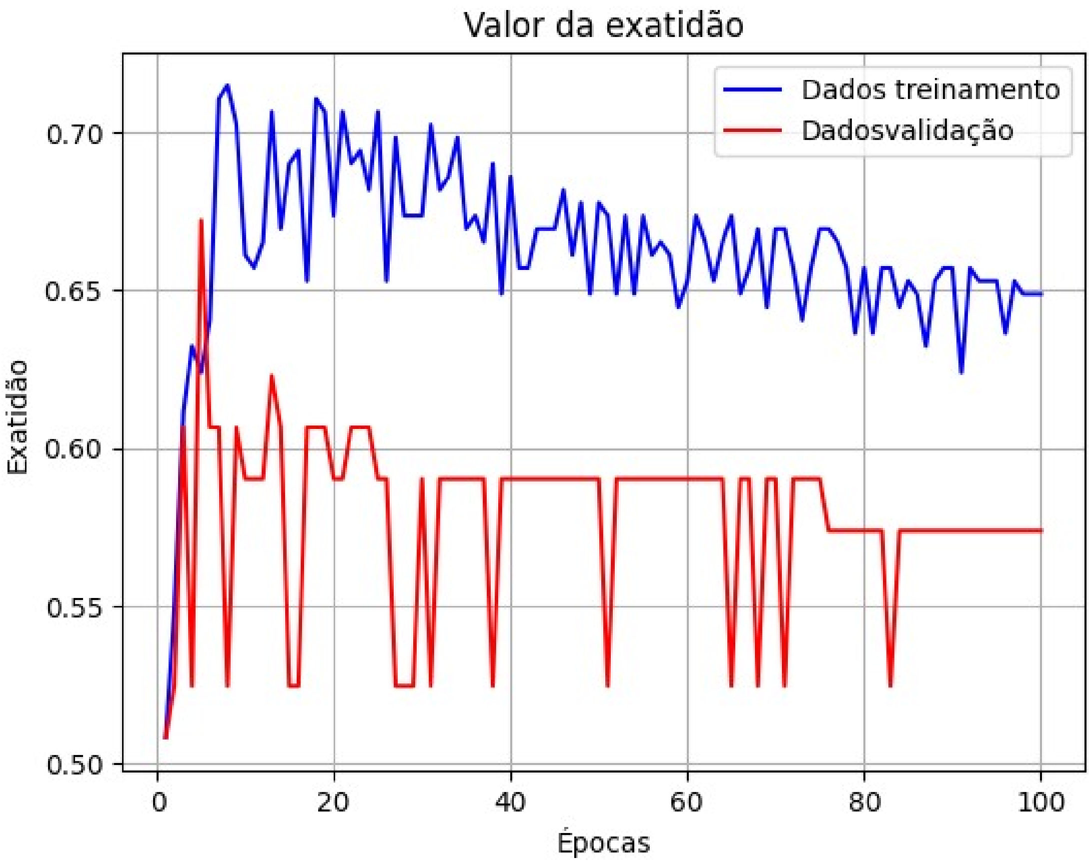
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6281 - loss: 4.9396
- val_accuracy: 0.5738 - val_loss: 9.3974
Epoch 85/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6819 - loss: 1.2847
- val_accuracy: 0.5738 - val_loss: 5.1978
Epoch 86/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6203 - loss: 1.2167
- val_accuracy: 0.5738 - val_loss: 1.2965
Epoch 87/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5951 - loss: 1.3777
- val_accuracy: 0.5738 - val_loss: 15.0900
Epoch 88/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6354 - loss: 2.7780
- val_accuracy: 0.5738 - val_loss: 10.9606
Epoch 89/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6631 - loss: 1.5521
- val_accuracy: 0.5738 - val_loss: 7.1854
Epoch 90/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6454 - loss: 1.1074
- val_accuracy: 0.5738 - val_loss: 3.6136
Epoch 91/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6093 - loss: 1.2745
- val_accuracy: 0.5738 - val_loss: 15.5894
Epoch 92/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6820 - loss: 1.9403
- val_accuracy: 0.5738 - val_loss: 11.9604
Epoch 93/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6746 - loss: 1.5789
- val_accuracy: 0.5738 - val_loss: 8.5997
Epoch 94/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6729 - loss: 1.7064
- val_accuracy: 0.5738 - val_loss: 5.4496
Epoch 95/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6492 - loss: 1.0797
- val_accuracy: 0.5738 - val_loss: 2.4075
Epoch 96/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6291 - loss: 0.7792
- val_accuracy: 0.5738 - val_loss: 17.9654
Epoch 97/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6648 - loss: 1.6932
- val_accuracy: 0.5738 - val_loss: 14.7005
Epoch 98/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6929 - loss: 2.5703
- val_accuracy: 0.5738 - val_loss: 11.7317
Epoch 99/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6222 - loss: 2.1844
- val_accuracy: 0.5738 - val_loss: 9.0322
Epoch 100/100
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6388 - loss: 1.2524
- val_accuracy: 0.5738 - val_loss: 6.5218
```

5. Resultados do treinamento

```python
# Recupera hostorico do treinamento
history_dict = history.history

# Salva custo e exatidão em vetores
custo = history_dict['loss']
exatidao = history_dict['accuracy']
val_custo = history_dict['val_loss']
val_exatidao = history_dict['val_accuracy']

# Cria vetor de épocas
epocas = range(1, len(custo) + 1)

# Gráfico do custo em funçaõ das épocas
plt.plot(epocas, custo, 'b', label='Dados treinamento')
plt.plot(epocas, val_custo, 'r', label='Dados validação')
plt.title('Valor da função de custo')
plt.xlabel('Épocas')
plt.ylabel('Custo')
plt.legend()
plt.grid()
plt.show()

# Gráfico da exatidão em função das épocas
plt.plot(epocas, exatidao, 'b', label='Dados treinamento')
plt.plot(epocas, val_exatidao, 'r', label='Dadosvalidação')
plt.title('Valor da exatidão')
plt.xlabel('Épocas')
plt.ylabel('Exatidão')
plt.legend()
plt.grid()
plt.show()
```


Valor da função de custo
Dados treinamento
Dados validação
Custo
Épocas
3500
3000
2500
2000
1500
1000
500
0
0
20
40
60
80
100



Avaliação dos resultados

Vamos avaliar a RN com o conjunto de dados de teste e obter os resultados para as métricas que foram utilizadas.

```
# Usando método evaluate calcule o custo e a exatidão para os dados de
treinamento e depois apresente os resultados
custo_e_metricas_train = rna.evaluate(X_train,Y_train)
print(custo_e_metricas_train)

# Usando método evaluate calcule o custo e a exatidão para os dados de
teste e depois apresente os resultados
custo_e_metricas_test = rna.evaluate(X_test, Y_test)
print(custo_e_metricas_test)

8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6833 - loss:
1.4906
[1.2480435371398926, 0.6487603187561035]
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5804 - loss: 7.1236

[6.5217790603637695, 0.5737704634666443]
```