Solução Lista 08

Nome: Julia Xavier
E-mail: julia.xavier@aluno.ufabc.edu.br
Nome: Leonardo Bernardes Lério da Silva
E-mail: leonardo.lerio@aluno.ufabc.edu.br

30 novembro, 2024

Exercício 01

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_diabetes
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import accuracy_score, precision_score, recall_score

diabetes = load_diabetes(as_frame=True)
data = diabetes.data
target = diabetes.target

df = pd.DataFrame(data, columns=diabetes.feature_names)

df["diabetes"] = target > target.mean()

X = df.drop("diabetes", axis=1)
y = df["diabetes"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential()
model.add(Dense(16, activation="relu", input_shape=(X_train.shape[1],)))
model.add(Dropout(0.2))
model.add(Dense(8, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1, activation="sigmoid"))

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

```python
history = model.fit(X_train, y_train, epochs=100, batch_size=5, validation_split=0.2, verbose=2)
```

```
## Epoch 1/100
## 57/57 - 0s - loss: 0.7623 - accuracy: 0.4574 - val_loss: 0.6946 - val_accuracy: 0.5634 - 385ms/epoch
## Epoch 2/100
## 57/57 - 0s - loss: 0.6855 - accuracy: 0.5426 - val_loss: 0.6547 - val_accuracy: 0.7042 - 44ms/epoch
## Epoch 3/100
## 57/57 - 0s - loss: 0.6786 - accuracy: 0.5532 - val_loss: 0.6241 - val_accuracy: 0.7324 - 42ms/epoch
## Epoch 4/100
## 57/57 - 0s - loss: 0.6465 - accuracy: 0.5993 - val_loss: 0.6025 - val_accuracy: 0.7324 - 44ms/epoch
## Epoch 5/100
## 57/57 - 0s - loss: 0.6356 - accuracy: 0.6844 - val_loss: 0.5891 - val_accuracy: 0.7606 - 43ms/epoch
## Epoch 6/100
## 57/57 - 0s - loss: 0.6055 - accuracy: 0.7021 - val_loss: 0.5675 - val_accuracy: 0.7465 - 43ms/epoch
## Epoch 7/100
## 57/57 - 0s - loss: 0.5894 - accuracy: 0.6986 - val_loss: 0.5589 - val_accuracy: 0.7324 - 44ms/epoch
## Epoch 8/100
## 57/57 - 0s - loss: 0.5713 - accuracy: 0.7163 - val_loss: 0.5468 - val_accuracy: 0.7465 - 43ms/epoch
## Epoch 9/100
## 57/57 - 0s - loss: 0.5599 - accuracy: 0.7340 - val_loss: 0.5431 - val_accuracy: 0.7465 - 45ms/epoch
## Epoch 10/100
## 57/57 - 0s - loss: 0.5583 - accuracy: 0.7305 - val_loss: 0.5400 - val_accuracy: 0.7324 - 47ms/epoch
## Epoch 11/100
## 57/57 - 0s - loss: 0.5205 - accuracy: 0.7730 - val_loss: 0.5344 - val_accuracy: 0.7324 - 45ms/epoch
## Epoch 12/100
## 57/57 - 0s - loss: 0.5240 - accuracy: 0.7482 - val_loss: 0.5339 - val_accuracy: 0.7183 - 46ms/epoch
## Epoch 13/100
## 57/57 - 0s - loss: 0.5244 - accuracy: 0.7553 - val_loss: 0.5318 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 14/100
## 57/57 - 0s - loss: 0.5054 - accuracy: 0.7553 - val_loss: 0.5284 - val_accuracy: 0.7183 - 42ms/epoch
## Epoch 15/100
## 57/57 - 0s - loss: 0.5266 - accuracy: 0.7447 - val_loss: 0.5276 - val_accuracy: 0.7324 - 44ms/epoch
## Epoch 16/100
## 57/57 - 0s - loss: 0.5255 - accuracy: 0.7553 - val_loss: 0.5250 - val_accuracy: 0.7183 - 44ms/epoch
## Epoch 17/100
## 57/57 - 0s - loss: 0.5187 - accuracy: 0.7801 - val_loss: 0.5296 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 18/100
## 57/57 - 0s - loss: 0.5140 - accuracy: 0.7908 - val_loss: 0.5291 - val_accuracy: 0.7324 - 42ms/epoch
## Epoch 19/100
## 57/57 - 0s - loss: 0.5225 - accuracy: 0.7411 - val_loss: 0.5258 - val_accuracy: 0.7324 - 45ms/epoch
## Epoch 20/100
## 57/57 - 0s - loss: 0.5205 - accuracy: 0.7589 - val_loss: 0.5257 - val_accuracy: 0.7324 - 45ms/epoch
## Epoch 21/100
## 57/57 - 0s - loss: 0.5044 - accuracy: 0.7730 - val_loss: 0.5294 - val_accuracy: 0.7324 - 43ms/epoch
## Epoch 22/100
## 57/57 - 0s - loss: 0.4842 - accuracy: 0.7730 - val_loss: 0.5283 - val_accuracy: 0.7324 - 43ms/epoch
## Epoch 23/100
## 57/57 - 0s - loss: 0.5057 - accuracy: 0.7553 - val_loss: 0.5245 - val_accuracy: 0.7324 - 43ms/epoch
## Epoch 24/100
## 57/57 - 0s - loss: 0.4898 - accuracy: 0.7872 - val_loss: 0.5249 - val_accuracy: 0.7324 - 44ms/epoch
## Epoch 25/100
## 57/57 - 0s - loss: 0.4818 - accuracy: 0.7589 - val_loss: 0.5243 - val_accuracy: 0.7324 - 43ms/epoch
## Epoch 26/100
```

```
## 57/57 - 0s - loss: 0.4804 - accuracy: 0.7766 - val_loss: 0.5257 - val_accuracy: 0.7324 - 43ms/epoch
## Epoch 27/100
## 57/57 - 0s - loss: 0.5099 - accuracy: 0.7589 - val_loss: 0.5284 - val_accuracy: 0.7324 - 44ms/epoch
## Epoch 28/100
## 57/57 - 0s - loss: 0.5023 - accuracy: 0.7589 - val_loss: 0.5308 - val_accuracy: 0.7324 - 43ms/epoch
## Epoch 29/100
## 57/57 - 0s - loss: 0.4918 - accuracy: 0.7695 - val_loss: 0.5332 - val_accuracy: 0.7183 - 42ms/epoch
## Epoch 30/100
## 57/57 - 0s - loss: 0.4608 - accuracy: 0.7660 - val_loss: 0.5353 - val_accuracy: 0.7183 - 42ms/epoch
## Epoch 31/100
## 57/57 - 0s - loss: 0.4573 - accuracy: 0.7801 - val_loss: 0.5404 - val_accuracy: 0.7183 - 42ms/epoch
## Epoch 32/100
## 57/57 - 0s - loss: 0.4883 - accuracy: 0.7837 - val_loss: 0.5407 - val_accuracy: 0.7183 - 46ms/epoch
## Epoch 33/100
## 57/57 - 0s - loss: 0.4634 - accuracy: 0.7589 - val_loss: 0.5413 - val_accuracy: 0.7183 - 44ms/epoch
## Epoch 34/100
## 57/57 - 0s - loss: 0.4934 - accuracy: 0.7482 - val_loss: 0.5413 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 35/100
## 57/57 - 0s - loss: 0.4947 - accuracy: 0.7518 - val_loss: 0.5393 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 36/100
## 57/57 - 0s - loss: 0.4936 - accuracy: 0.7624 - val_loss: 0.5381 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 37/100
## 57/57 - 0s - loss: 0.4835 - accuracy: 0.7553 - val_loss: 0.5390 - val_accuracy: 0.7183 - 42ms/epoch
## Epoch 38/100
## 57/57 - 0s - loss: 0.4683 - accuracy: 0.7660 - val_loss: 0.5365 - val_accuracy: 0.7183 - 44ms/epoch
## Epoch 39/100
## 57/57 - 0s - loss: 0.4516 - accuracy: 0.7695 - val_loss: 0.5383 - val_accuracy: 0.7183 - 42ms/epoch
## Epoch 40/100
## 57/57 - 0s - loss: 0.4800 - accuracy: 0.7730 - val_loss: 0.5395 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 41/100
## 57/57 - 0s - loss: 0.4433 - accuracy: 0.7872 - val_loss: 0.5415 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 42/100
## 57/57 - 0s - loss: 0.4683 - accuracy: 0.7482 - val_loss: 0.5427 - val_accuracy: 0.7183 - 44ms/epoch
## Epoch 43/100
## 57/57 - 0s - loss: 0.4610 - accuracy: 0.7872 - val_loss: 0.5479 - val_accuracy: 0.7042 - 44ms/epoch
## Epoch 44/100
## 57/57 - 0s - loss: 0.4682 - accuracy: 0.7766 - val_loss: 0.5473 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 45/100
## 57/57 - 0s - loss: 0.4817 - accuracy: 0.7908 - val_loss: 0.5468 - val_accuracy: 0.7042 - 44ms/epoch
## Epoch 46/100
## 57/57 - 0s - loss: 0.4606 - accuracy: 0.7766 - val_loss: 0.5503 - val_accuracy: 0.6901 - 44ms/epoch
## Epoch 47/100
## 57/57 - 0s - loss: 0.4845 - accuracy: 0.7801 - val_loss: 0.5515 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 48/100
## 57/57 - 0s - loss: 0.4425 - accuracy: 0.7837 - val_loss: 0.5528 - val_accuracy: 0.7042 - 43ms/epoch
## Epoch 49/100
## 57/57 - 0s - loss: 0.4812 - accuracy: 0.7553 - val_loss: 0.5529 - val_accuracy: 0.7042 - 42ms/epoch
## Epoch 50/100
## 57/57 - 0s - loss: 0.4674 - accuracy: 0.7518 - val_loss: 0.5516 - val_accuracy: 0.7183 - 45ms/epoch
## Epoch 51/100
## 57/57 - 0s - loss: 0.4411 - accuracy: 0.7730 - val_loss: 0.5518 - val_accuracy: 0.6901 - 45ms/epoch
## Epoch 52/100
## 57/57 - 0s - loss: 0.4647 - accuracy: 0.7837 - val_loss: 0.5561 - val_accuracy: 0.6901 - 44ms/epoch
## Epoch 53/100
```

```
## 57/57 - 0s - loss: 0.4515 - accuracy: 0.7730 - val_loss: 0.5563 - val_accuracy: 0.6901 - 46ms/epoch
## Epoch 54/100
## 57/57 - 0s - loss: 0.4865 - accuracy: 0.7553 - val_loss: 0.5561 - val_accuracy: 0.6901 - 45ms/epoch
## Epoch 55/100
## 57/57 - 0s - loss: 0.4718 - accuracy: 0.7518 - val_loss: 0.5556 - val_accuracy: 0.6761 - 45ms/epoch
## Epoch 56/100
## 57/57 - 0s - loss: 0.4592 - accuracy: 0.7730 - val_loss: 0.5581 - val_accuracy: 0.6761 - 45ms/epoch
## Epoch 57/100
## 57/57 - 0s - loss: 0.4617 - accuracy: 0.7872 - val_loss: 0.5627 - val_accuracy: 0.6620 - 44ms/epoch
## Epoch 58/100
## 57/57 - 0s - loss: 0.4429 - accuracy: 0.7943 - val_loss: 0.5600 - val_accuracy: 0.6901 - 44ms/epoch
## Epoch 59/100
## 57/57 - 0s - loss: 0.4740 - accuracy: 0.7589 - val_loss: 0.5629 - val_accuracy: 0.6761 - 43ms/epoch
## Epoch 60/100
## 57/57 - 0s - loss: 0.4598 - accuracy: 0.7943 - val_loss: 0.5621 - val_accuracy: 0.7042 - 42ms/epoch
## Epoch 61/100
## 57/57 - 0s - loss: 0.4428 - accuracy: 0.7766 - val_loss: 0.5657 - val_accuracy: 0.6901 - 43ms/epoch
## Epoch 62/100
## 57/57 - 0s - loss: 0.4457 - accuracy: 0.7908 - val_loss: 0.5678 - val_accuracy: 0.7042 - 43ms/epoch
## Epoch 63/100
## 57/57 - 0s - loss: 0.4430 - accuracy: 0.7766 - val_loss: 0.5658 - val_accuracy: 0.6761 - 43ms/epoch
## Epoch 64/100
## 57/57 - 0s - loss: 0.4693 - accuracy: 0.7518 - val_loss: 0.5671 - val_accuracy: 0.6901 - 44ms/epoch
## Epoch 65/100
## 57/57 - 0s - loss: 0.4400 - accuracy: 0.7872 - val_loss: 0.5709 - val_accuracy: 0.6901 - 45ms/epoch
## Epoch 66/100
## 57/57 - 0s - loss: 0.4785 - accuracy: 0.7589 - val_loss: 0.5700 - val_accuracy: 0.6901 - 44ms/epoch
## Epoch 67/100
## 57/57 - 0s - loss: 0.4569 - accuracy: 0.7695 - val_loss: 0.5664 - val_accuracy: 0.7183 - 43ms/epoch
## Epoch 68/100
## 57/57 - 0s - loss: 0.4426 - accuracy: 0.7624 - val_loss: 0.5706 - val_accuracy: 0.7042 - 43ms/epoch
## Epoch 69/100
## 57/57 - 0s - loss: 0.4336 - accuracy: 0.7766 - val_loss: 0.5702 - val_accuracy: 0.7183 - 42ms/epoch
## Epoch 70/100
## 57/57 - 0s - loss: 0.4189 - accuracy: 0.8050 - val_loss: 0.5693 - val_accuracy: 0.7042 - 42ms/epoch
## Epoch 71/100
## 57/57 - 0s - loss: 0.4713 - accuracy: 0.7695 - val_loss: 0.5719 - val_accuracy: 0.6761 - 41ms/epoch
## Epoch 72/100
## 57/57 - 0s - loss: 0.4393 - accuracy: 0.7766 - val_loss: 0.5736 - val_accuracy: 0.7042 - 43ms/epoch
## Epoch 73/100
## 57/57 - 0s - loss: 0.4374 - accuracy: 0.7908 - val_loss: 0.5752 - val_accuracy: 0.6901 - 42ms/epoch
## Epoch 74/100
## 57/57 - 0s - loss: 0.4458 - accuracy: 0.7943 - val_loss: 0.5686 - val_accuracy: 0.6761 - 42ms/epoch
## Epoch 75/100
## 57/57 - 0s - loss: 0.4484 - accuracy: 0.7730 - val_loss: 0.5747 - val_accuracy: 0.6620 - 42ms/epoch
## Epoch 76/100
## 57/57 - 0s - loss: 0.4397 - accuracy: 0.7943 - val_loss: 0.5754 - val_accuracy: 0.6620 - 42ms/epoch
## Epoch 77/100
## 57/57 - 0s - loss: 0.4439 - accuracy: 0.7730 - val_loss: 0.5757 - val_accuracy: 0.6620 - 42ms/epoch
## Epoch 78/100
## 57/57 - 0s - loss: 0.4308 - accuracy: 0.7801 - val_loss: 0.5779 - val_accuracy: 0.6761 - 43ms/epoch
## Epoch 79/100
## 57/57 - 0s - loss: 0.4509 - accuracy: 0.7730 - val_loss: 0.5770 - val_accuracy: 0.6901 - 43ms/epoch
## Epoch 80/100
```

```
## 57/57 - 0s - loss: 0.4181 - accuracy: 0.7872 - val_loss: 0.5778 - val_accuracy: 0.6901 - 43ms/epoch
## Epoch 81/100
## 57/57 - 0s - loss: 0.4424 - accuracy: 0.7801 - val_loss: 0.5787 - val_accuracy: 0.6761 - 43ms/epoch
## Epoch 82/100
## 57/57 - 0s - loss: 0.4150 - accuracy: 0.7801 - val_loss: 0.5835 - val_accuracy: 0.6901 - 44ms/epoch
## Epoch 83/100
## 57/57 - 0s - loss: 0.4555 - accuracy: 0.7766 - val_loss: 0.5776 - val_accuracy: 0.6761 - 47ms/epoch
## Epoch 84/100
## 57/57 - 0s - loss: 0.4332 - accuracy: 0.7943 - val_loss: 0.5747 - val_accuracy: 0.6761 - 51ms/epoch
## Epoch 85/100
## 57/57 - 0s - loss: 0.4642 - accuracy: 0.7801 - val_loss: 0.5730 - val_accuracy: 0.6761 - 58ms/epoch
## Epoch 86/100
## 57/57 - 0s - loss: 0.4239 - accuracy: 0.7766 - val_loss: 0.5759 - val_accuracy: 0.6761 - 50ms/epoch
## Epoch 87/100
## 57/57 - 0s - loss: 0.4421 - accuracy: 0.7837 - val_loss: 0.5771 - val_accuracy: 0.6761 - 52ms/epoch
## Epoch 88/100
## 57/57 - 0s - loss: 0.4540 - accuracy: 0.7908 - val_loss: 0.5733 - val_accuracy: 0.6761 - 46ms/epoch
## Epoch 89/100
## 57/57 - 0s - loss: 0.4473 - accuracy: 0.7872 - val_loss: 0.5753 - val_accuracy: 0.6761 - 44ms/epoch
## Epoch 90/100
## 57/57 - 0s - loss: 0.4310 - accuracy: 0.8050 - val_loss: 0.5796 - val_accuracy: 0.6761 - 44ms/epoch
## Epoch 91/100
## 57/57 - 0s - loss: 0.4493 - accuracy: 0.7660 - val_loss: 0.5829 - val_accuracy: 0.6761 - 43ms/epoch
## Epoch 92/100
## 57/57 - 0s - loss: 0.4537 - accuracy: 0.7837 - val_loss: 0.5807 - val_accuracy: 0.6761 - 42ms/epoch
## Epoch 93/100
## 57/57 - 0s - loss: 0.4255 - accuracy: 0.7872 - val_loss: 0.5799 - val_accuracy: 0.6761 - 42ms/epoch
## Epoch 94/100
## 57/57 - 0s - loss: 0.4421 - accuracy: 0.7730 - val_loss: 0.5797 - val_accuracy: 0.6338 - 43ms/epoch
## Epoch 95/100
## 57/57 - 0s - loss: 0.4186 - accuracy: 0.7801 - val_loss: 0.5793 - val_accuracy: 0.6761 - 44ms/epoch
## Epoch 96/100
## 57/57 - 0s - loss: 0.4456 - accuracy: 0.7730 - val_loss: 0.5786 - val_accuracy: 0.6620 - 44ms/epoch
## Epoch 97/100
## 57/57 - 0s - loss: 0.4323 - accuracy: 0.7872 - val_loss: 0.5751 - val_accuracy: 0.6620 - 42ms/epoch
## Epoch 98/100
## 57/57 - 0s - loss: 0.4315 - accuracy: 0.7624 - val_loss: 0.5745 - val_accuracy: 0.6620 - 42ms/epoch
## Epoch 99/100
## 57/57 - 0s - loss: 0.4435 - accuracy: 0.8050 - val_loss: 0.5794 - val_accuracy: 0.6479 - 42ms/epoch
## Epoch 100/100
## 57/57 - 0s - loss: 0.4271 - accuracy: 0.7695 - val_loss: 0.5823 - val_accuracy: 0.6479 - 44ms/epoch
```

```
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print(f"Acuracia: {accuracy:.4f}")
```

```
## Acuracia: 0.7753
```

```
print(f"Precisao: {precision:.4f}")
```

```
## Precisao: 0.7436
```

```
print(f"Recall: {recall:.4f}")
```

```
## Recall: 0.7436
```

Exercício 02

```
library(car)
library(tidyverse)
df <- as_tibble(Salaries)
write.csv(df, file = "Salaries.csv")
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

data = pd.read_csv("Salaries.csv")
data
```

```
##      Unnamed: 0      rank discipline  yrs.since.phd  yrs.service   sex  salary
## 0             1      Prof          B             19           18  Male  139750
## 1             2      Prof          B             20           16  Male  173200
## 2             3  AsstProf          B              4            3  Male   79750
## 3             4      Prof          B             45           39  Male  115000
## 4             5      Prof          B             40           41  Male  141500
## ..          ...       ...        ...            ...          ...   ...     ...
## 392         393      Prof          A             33           30  Male  103106
## 393         394      Prof          A             31           19  Male  150564
## 394         395      Prof          A             42           25  Male  101738
## 395         396      Prof          A             25           15  Male   95329
## 396         397  AsstProf          A              8            4  Male   81035
##
## [397 rows x 7 columns]
```

```
X = data.drop("salary", axis=1)
y = data["salary"]

ohe = OneHotEncoder(drop="first", sparse=False)
X_encoded = ohe.fit_transform(X[["rank", "discipline", "sex"]])
X_encoded = pd.DataFrame(X_encoded, columns=ohe.get_feature_names_out(["rank", "discipline", "sex"]), i
X_encoded = pd.concat([X.drop(["rank", "discipline", "sex"], axis=1), X_encoded], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation="linear", input_shape=(X_encoded.shape[1],)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation="linear"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation="linear")
])

model.compile(optimizer="adam", loss="mean_squared_error")

model.fit(X_train, y_train, epochs=100, batch_size=5, verbose=2)
```

```
## Epoch 1/100
## 64/64 - 0s - loss: 13870575616.0000 - 203ms/epoch - 3ms/step
## Epoch 2/100
## 64/64 - 0s - loss: 13236379648.0000 - 30ms/epoch - 463us/step
## Epoch 3/100
## 64/64 - 0s - loss: 11347302400.0000 - 28ms/epoch - 444us/step
## Epoch 4/100
## 64/64 - 0s - loss: 8088059392.0000 - 29ms/epoch - 456us/step
## Epoch 5/100
## 64/64 - 0s - loss: 4824221696.0000 - 29ms/epoch - 449us/step
## Epoch 6/100
## 64/64 - 0s - loss: 3403094528.0000 - 28ms/epoch - 443us/step
## Epoch 7/100
## 64/64 - 0s - loss: 3058721536.0000 - 29ms/epoch - 459us/step
## Epoch 8/100
## 64/64 - 0s - loss: 3135052288.0000 - 29ms/epoch - 448us/step
## Epoch 9/100
## 64/64 - 0s - loss: 2995212544.0000 - 30ms/epoch - 461us/step
## Epoch 10/100
## 64/64 - 0s - loss: 2982556416.0000 - 28ms/epoch - 442us/step
## Epoch 11/100
## 64/64 - 0s - loss: 2910930688.0000 - 29ms/epoch - 456us/step
## Epoch 12/100
## 64/64 - 0s - loss: 2926589440.0000 - 29ms/epoch - 454us/step
## Epoch 13/100
## 64/64 - 0s - loss: 2894524416.0000 - 28ms/epoch - 442us/step
## Epoch 14/100
## 64/64 - 0s - loss: 2901468672.0000 - 29ms/epoch - 456us/step
## Epoch 15/100
## 64/64 - 0s - loss: 2791745024.0000 - 29ms/epoch - 447us/step
## Epoch 16/100
## 64/64 - 0s - loss: 2719170816.0000 - 29ms/epoch - 454us/step
## Epoch 17/100
## 64/64 - 0s - loss: 2708631296.0000 - 30ms/epoch - 463us/step
## Epoch 18/100
## 64/64 - 0s - loss: 2561618944.0000 - 29ms/epoch - 448us/step
## Epoch 19/100
## 64/64 - 0s - loss: 2577453568.0000 - 29ms/epoch - 454us/step
## Epoch 20/100
## 64/64 - 0s - loss: 2752118528.0000 - 29ms/epoch - 454us/step
## Epoch 21/100
```

```
## 64/64 - 0s - loss: 2684369152.0000 - 28ms/epoch - 440us/step
## Epoch 22/100
## 64/64 - 0s - loss: 2563159808.0000 - 29ms/epoch - 445us/step
## Epoch 23/100
## 64/64 - 0s - loss: 2624130304.0000 - 27ms/epoch - 430us/step
## Epoch 24/100
## 64/64 - 0s - loss: 2526401536.0000 - 28ms/epoch - 443us/step
## Epoch 25/100
## 64/64 - 0s - loss: 2467617536.0000 - 30ms/epoch - 462us/step
## Epoch 26/100
## 64/64 - 0s - loss: 2497084416.0000 - 28ms/epoch - 445us/step
## Epoch 27/100
## 64/64 - 0s - loss: 2463083520.0000 - 29ms/epoch - 458us/step
## Epoch 28/100
## 64/64 - 0s - loss: 2482507264.0000 - 29ms/epoch - 446us/step
## Epoch 29/100
## 64/64 - 0s - loss: 2434187520.0000 - 29ms/epoch - 460us/step
## Epoch 30/100
## 64/64 - 0s - loss: 2453713408.0000 - 29ms/epoch - 446us/step
## Epoch 31/100
## 64/64 - 0s - loss: 2345251072.0000 - 28ms/epoch - 441us/step
## Epoch 32/100
## 64/64 - 0s - loss: 2454418688.0000 - 29ms/epoch - 450us/step
## Epoch 33/100
## 64/64 - 0s - loss: 2368058368.0000 - 29ms/epoch - 456us/step
## Epoch 34/100
## 64/64 - 0s - loss: 2299143168.0000 - 29ms/epoch - 456us/step
## Epoch 35/100
## 64/64 - 0s - loss: 2353022976.0000 - 29ms/epoch - 447us/step
## Epoch 36/100
## 64/64 - 0s - loss: 2230628864.0000 - 29ms/epoch - 448us/step
## Epoch 37/100
## 64/64 - 0s - loss: 2206956544.0000 - 29ms/epoch - 450us/step
## Epoch 38/100
## 64/64 - 0s - loss: 2301836544.0000 - 29ms/epoch - 454us/step
## Epoch 39/100
## 64/64 - 0s - loss: 2309199616.0000 - 29ms/epoch - 446us/step
## Epoch 40/100
## 64/64 - 0s - loss: 2279211520.0000 - 30ms/epoch - 461us/step
## Epoch 41/100
## 64/64 - 0s - loss: 2203555072.0000 - 28ms/epoch - 443us/step
## Epoch 42/100
## 64/64 - 0s - loss: 2190178816.0000 - 29ms/epoch - 459us/step
## Epoch 43/100
## 64/64 - 0s - loss: 2238413056.0000 - 29ms/epoch - 460us/step
## Epoch 44/100
## 64/64 - 0s - loss: 2101625600.0000 - 29ms/epoch - 451us/step
## Epoch 45/100
## 64/64 - 0s - loss: 2182059776.0000 - 28ms/epoch - 442us/step
## Epoch 46/100
## 64/64 - 0s - loss: 2202998528.0000 - 28ms/epoch - 438us/step
## Epoch 47/100
## 64/64 - 0s - loss: 2096355584.0000 - 29ms/epoch - 447us/step
## Epoch 48/100
```

```
## 64/64 - 0s - loss: 2150119424.0000 - 28ms/epoch - 443us/step
## Epoch 49/100
## 64/64 - 0s - loss: 2066050944.0000 - 30ms/epoch - 461us/step
## Epoch 50/100
## 64/64 - 0s - loss: 1959621120.0000 - 30ms/epoch - 466us/step
## Epoch 51/100
## 64/64 - 0s - loss: 2188641536.0000 - 31ms/epoch - 485us/step
## Epoch 52/100
## 64/64 - 0s - loss: 2080090752.0000 - 29ms/epoch - 448us/step
## Epoch 53/100
## 64/64 - 0s - loss: 2157679872.0000 - 28ms/epoch - 441us/step
## Epoch 54/100
## 64/64 - 0s - loss: 2025008640.0000 - 29ms/epoch - 451us/step
## Epoch 55/100
## 64/64 - 0s - loss: 2148918784.0000 - 29ms/epoch - 459us/step
## Epoch 56/100
## 64/64 - 0s - loss: 2109315200.0000 - 28ms/epoch - 440us/step
## Epoch 57/100
## 64/64 - 0s - loss: 2023868672.0000 - 28ms/epoch - 442us/step
## Epoch 58/100
## 64/64 - 0s - loss: 1952362752.0000 - 29ms/epoch - 451us/step
## Epoch 59/100
## 64/64 - 0s - loss: 2008302080.0000 - 29ms/epoch - 446us/step
## Epoch 60/100
## 64/64 - 0s - loss: 1922831232.0000 - 29ms/epoch - 452us/step
## Epoch 61/100
## 64/64 - 0s - loss: 2054284672.0000 - 28ms/epoch - 441us/step
## Epoch 62/100
## 64/64 - 0s - loss: 2088897536.0000 - 29ms/epoch - 454us/step
## Epoch 63/100
## 64/64 - 0s - loss: 1867281920.0000 - 30ms/epoch - 462us/step
## Epoch 64/100
## 64/64 - 0s - loss: 1997790336.0000 - 29ms/epoch - 449us/step
## Epoch 65/100
## 64/64 - 0s - loss: 1954372224.0000 - 31ms/epoch - 486us/step
## Epoch 66/100
## 64/64 - 0s - loss: 2001794432.0000 - 30ms/epoch - 472us/step
## Epoch 67/100
## 64/64 - 0s - loss: 1908756736.0000 - 30ms/epoch - 467us/step
## Epoch 68/100
## 64/64 - 0s - loss: 2004985600.0000 - 31ms/epoch - 478us/step
## Epoch 69/100
## 64/64 - 0s - loss: 1981330816.0000 - 29ms/epoch - 454us/step
## Epoch 70/100
## 64/64 - 0s - loss: 1962007936.0000 - 30ms/epoch - 461us/step
## Epoch 71/100
## 64/64 - 0s - loss: 1976508672.0000 - 29ms/epoch - 459us/step
## Epoch 72/100
## 64/64 - 0s - loss: 2113335936.0000 - 29ms/epoch - 448us/step
## Epoch 73/100
## 64/64 - 0s - loss: 1894972032.0000 - 29ms/epoch - 455us/step
## Epoch 74/100
## 64/64 - 0s - loss: 1999587712.0000 - 29ms/epoch - 446us/step
## Epoch 75/100
```

```
## 64/64 - 0s - loss: 1966112000.0000 - 30ms/epoch - 461us/step
## Epoch 76/100
## 64/64 - 0s - loss: 1965865216.0000 - 28ms/epoch - 443us/step
## Epoch 77/100
## 64/64 - 0s - loss: 1974793344.0000 - 29ms/epoch - 448us/step
## Epoch 78/100
## 64/64 - 0s - loss: 1961202048.0000 - 29ms/epoch - 446us/step
## Epoch 79/100
## 64/64 - 0s - loss: 1895109248.0000 - 28ms/epoch - 445us/step
## Epoch 80/100
## 64/64 - 0s - loss: 1930611200.0000 - 29ms/epoch - 446us/step
## Epoch 81/100
## 64/64 - 0s - loss: 1934841472.0000 - 29ms/epoch - 454us/step
## Epoch 82/100
## 64/64 - 0s - loss: 1840117888.0000 - 29ms/epoch - 448us/step
## Epoch 83/100
## 64/64 - 0s - loss: 1937502208.0000 - 28ms/epoch - 439us/step
## Epoch 84/100
## 64/64 - 0s - loss: 1845640576.0000 - 29ms/epoch - 447us/step
## Epoch 85/100
## 64/64 - 0s - loss: 1843803520.0000 - 29ms/epoch - 456us/step
## Epoch 86/100
## 64/64 - 0s - loss: 1780038400.0000 - 30ms/epoch - 463us/step
## Epoch 87/100
## 64/64 - 0s - loss: 1786943872.0000 - 29ms/epoch - 448us/step
## Epoch 88/100
## 64/64 - 0s - loss: 1827689728.0000 - 28ms/epoch - 444us/step
## Epoch 89/100
## 64/64 - 0s - loss: 1815847552.0000 - 29ms/epoch - 452us/step
## Epoch 90/100
## 64/64 - 0s - loss: 1901572736.0000 - 29ms/epoch - 453us/step
## Epoch 91/100
## 64/64 - 0s - loss: 1804931584.0000 - 28ms/epoch - 445us/step
## Epoch 92/100
## 64/64 - 0s - loss: 1788034176.0000 - 29ms/epoch - 456us/step
## Epoch 93/100
## 64/64 - 0s - loss: 1842593792.0000 - 28ms/epoch - 445us/step
## Epoch 94/100
## 64/64 - 0s - loss: 1790608640.0000 - 29ms/epoch - 449us/step
## Epoch 95/100
## 64/64 - 0s - loss: 1757925632.0000 - 29ms/epoch - 450us/step
## Epoch 96/100
## 64/64 - 0s - loss: 1891174016.0000 - 28ms/epoch - 441us/step
## Epoch 97/100
## 64/64 - 0s - loss: 1697284352.0000 - 28ms/epoch - 444us/step
## Epoch 98/100
## 64/64 - 0s - loss: 1788745600.0000 - 28ms/epoch - 439us/step
## Epoch 99/100
## 64/64 - 0s - loss: 1813246720.0000 - 28ms/epoch - 442us/step
## Epoch 100/100
## 64/64 - 0s - loss: 1748876160.0000 - 28ms/epoch - 445us/step
## <keras.callbacks.History object at 0x0000019146971DC0>
```

```
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)
```

```
## Mean Absolute Error: 37632.517797851564
```

Exercício 03

```
library(keras)

# carregando o dataset mnist e convertendo os valores de pixels
# que são entre 0-255 para valores entre 0 e 1
mnist <- dataset_mnist()
mnist$train$x <- mnist$train$x/255
mnist$test$x <- mnist$test$x/255

###################################
## 1. Define o modelo Keras
##################################
# A primeira camada deve especificar o argumento
# input_shape que representa as dimensões da entrada (28x28).
# Você deve completar o código adicionando:
# - uma camada densa (multilayer perceptron) com 128 neurônios e ativação relu
# - uma camada de dropout com taxa 0.2
# - uma camada de saída adequada
model <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(28, 28)) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 10, activation = "softmax")

# Para checar seu modelo
summary(model)
```

```
## Model: "sequential_2"
## ________________________________________________________________________
##  Layer (type)                    Output Shape                  Param #
## ========================================================================
##  flatten (Flatten)               (None, 784)                   0
##  dense_7 (Dense)                 (None, 128)                   100480
##  dropout_4 (Dropout)             (None, 128)                   0
##  dense_6 (Dense)                 (None, 10)                    1290
## ========================================================================
## Total params: 101,770
## Trainable params: 101,770
## Non-trainable params: 0
## ________________________________________________________________________
```

```
###################################
## 2. Compilamos o modelo
```

```r
##################################
# Compile aqui seu modelo. Utilize:
# - otimizador "adam",
# - função de perda "sparse_categorical_crossentropy"
# - métrica "accuracy"
model %>%
  compile(
    optimizer = "adam",
    loss = "sparse_categorical_crossentropy",
    metrics = c("accuracy")
  )

##############################################
## 3. Ajustamos os dados ao conjunto de testes
##############################################
model %>%
  fit(
    x = mnist$train$x, y = mnist$train$y,
    epochs = 5,
    validation_split = 0.3,
    verbose = 2
  )

#########################################################
## 4. Vamos testar o resultado usando o conjunto de testes
#########################################################
predictions <- predict(model, mnist$test$x)
head(predictions, 2)
```

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 2.299187e-07 3.589888e-10 6.100850e-06 4.137513e-04 9.189872e-11
## [2,] 1.002994e-06 1.527273e-04 9.998411e-01 4.678933e-06 2.143459e-13
##              [,6]         [,7]         [,8]         [,9]        [,10]
## [1,] 1.749957e-07 3.275903e-13 9.995542e-01 9.363356e-07 2.473063e-05
## [2,] 4.122268e-07 3.231517e-08 9.027245e-13 6.321021e-08 5.226106e-12
```

```r
model %>%
  evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##       loss   accuracy
## 0.08118469 0.97549999
```