

LISTA 2 - EEC1505: REDES NEURAIS

Aluno(a): Beatriz Soares de Souza

Matrícula: 20211020152

Aluno(a): Pedro Victor Andrade Alves

Matrícula: 20211027716

Aluno(a): Tiago de Oliveira Barreto

Matrícula: 20211020125

1°)

Apresente um estudo sobre a rede deep learning convolutiva focando sobre os aspectos das implementações computacionais. Considere neste estudo os aspectos da linguagem, softwares para o processo de treinamento e aplicações.

Resposta:

Epilepsy Radiology Reports Classification Using Deep Learning Networks

Sengul Bayrak^{1,2}, Eylem Yucel^{2,*} and Hidayet Takci³

¹Department of Computer Engineering, Halic University, Istanbul, 34445, Turkey

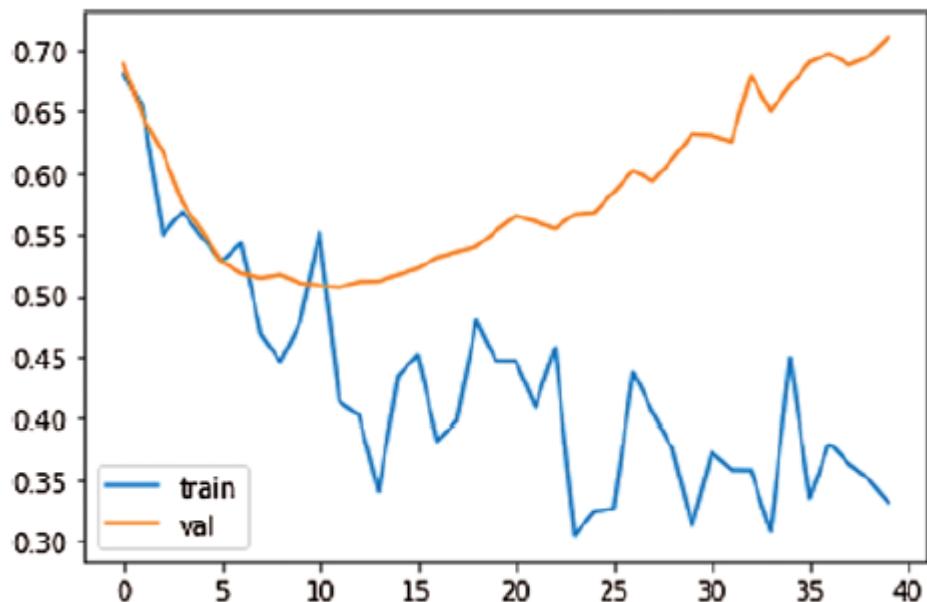
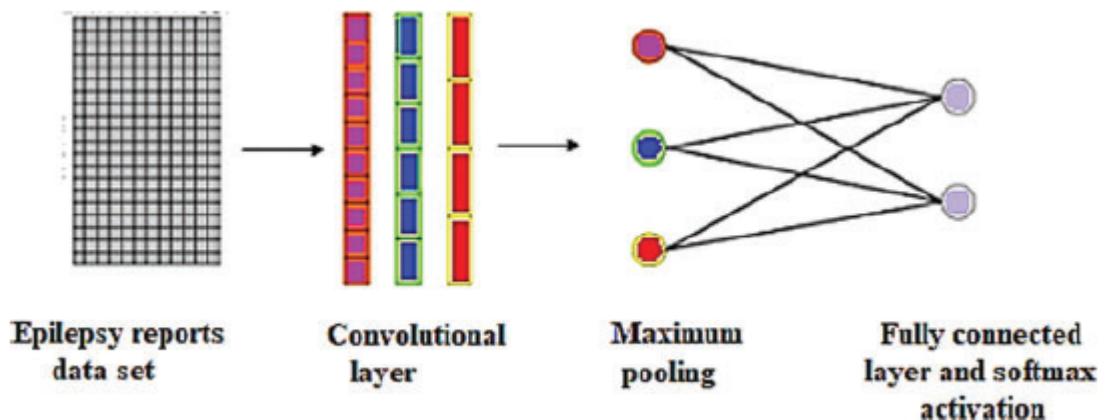
²Department of Computer Engineering, Istanbul University – Cerrahpasa, Istanbul, 34320, Turkey

³Department of Computer Engineering, Sivas Cumhuriyet University, Sivas, 58140, Turkey

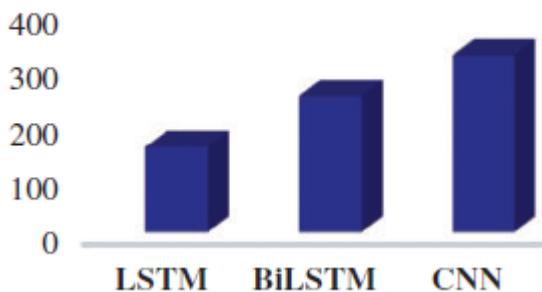
*Corresponding Author: Eylem Yucel. Email: eylem@iuc.edu.tr

Received: 19 March 2021; Accepted: 14 June 2021

Na pesquisa realizada foi feita uma avaliação de relatórios de ressonância magnética e esses dados apurados em três redes neurais: LSTM, BiLSTM e CNN. Os dados foram apurados por meio da avaliação de 122 pacientes (97 epiléticos e 23 saudáveis). Foram utilizados 70% para treinamento, 15% validação e 15% para testes. O objetivo é identificar o nível de aprendizagem para auxiliar no diagnóstico e detecção de pacientes com epilepsia. No treino da CNN foram utilizados 15 mini-batch e 150 epochs. A estruturação de como foi montado a rede CNN e os resultados de treinamento e validação são apresentados nas figuras abaixo.



Quando comparada com outras redes de aprendizado, a CNN demonstrou-se mais demorada, necessitando de mais tempo para o processo de treinamento. E com relação a apresentação da sensibilidade, especificidade e acurácia, a rede CNN também apresenta resultados inferiores.

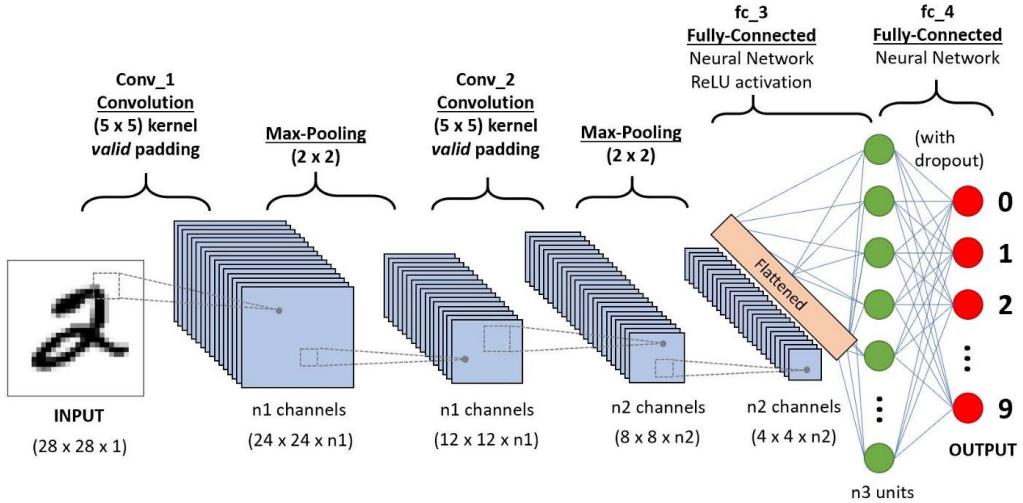


Proposed method	Sensitivity (%)	Specificity (%)	Accuracy (%)
LSTM network	97.05	100	97.67
BiLSTM network	100	100	100
CNN	81.08	58.33	77.90

2º)

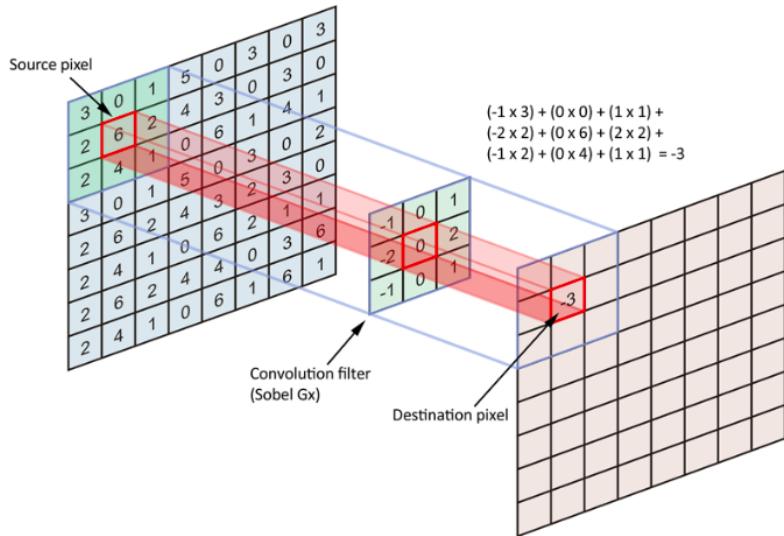
Considere uma rede deep learning convolutiva (treinada) aplicada à classificação de padrões em imagens. A base de dados considerada é a CIFAR-10 (pesquise). A referida base de dados consiste de 60 mil imagens coloridas de 32x32 pixels, com 50 mil para treino e 10 mil para teste. As imagens estão divididas em 10 classes, a saber: avião, navio, caminhão, automóvel, sapo, pássaro, cachorro, gato, cavalo e cervo. Cada imagem possui apenas um dos objetos da classe de interesse, podendo estar parcialmente obstruído por outros objetos que não pertençam a esse conjunto. Apresente o desempenho da rede no processo de classificação usando uma matriz de confusão.

Obs. Pesquise e utilize uma rede convolutiva já treinada



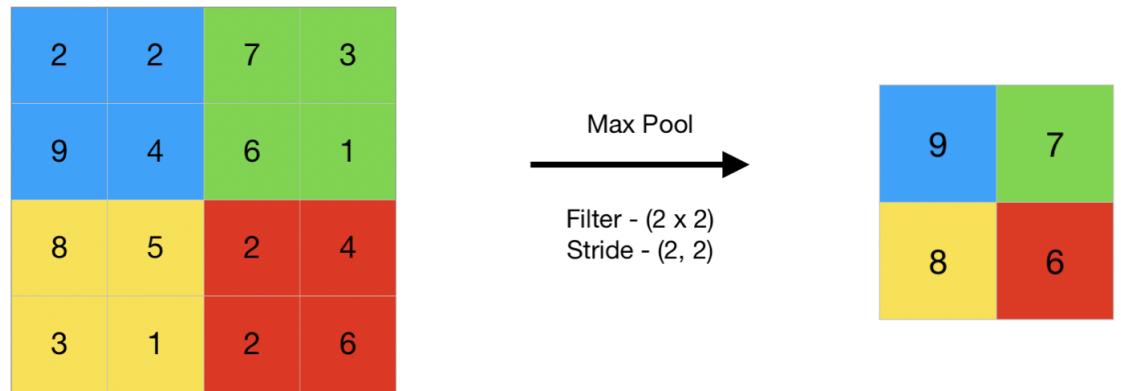
- **Convolução**

- As camadas convolucionais possuem mapas de características obtidas por filtros (forma de destacar características)



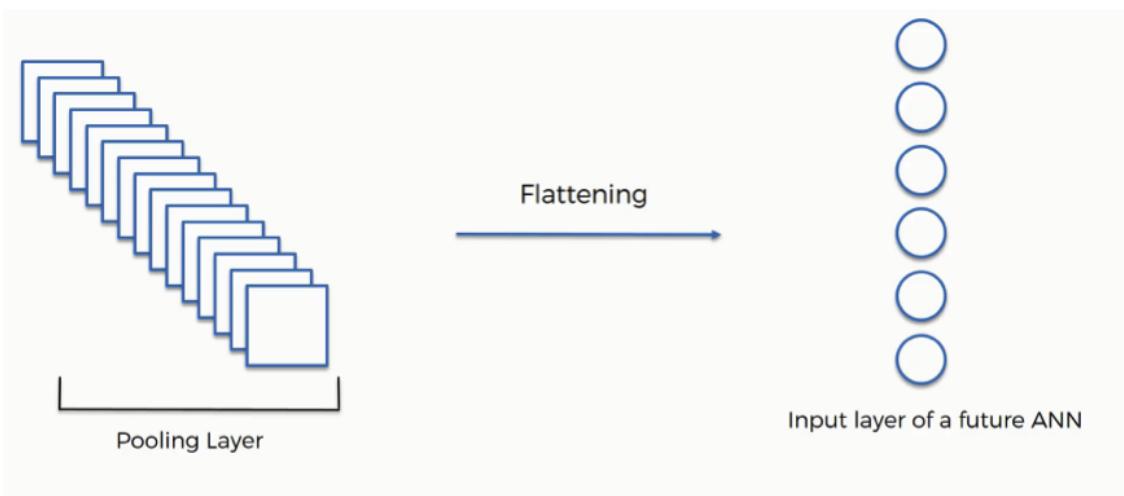
- **“Pooling”**

- Reduz a dimensionalidade da imagem (preservando as características) a partir do resultado pelo mapa de característica gerado pela etapa de convolução. O Pooling facilita o treinamento, porque quanto menor o tamanho da imagem, menor o esforço computacional para o treinamento. No max pooling é utilizado o valor máximo presente em cada região escolhida.



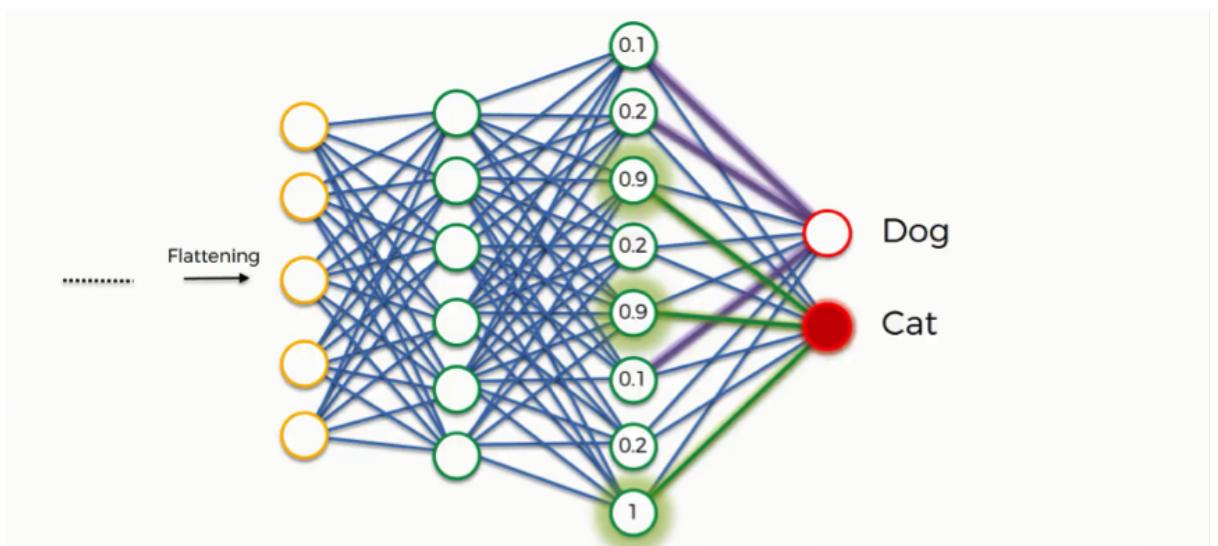
- “Flattening”

- Transforma a etapa de pooling em um vetor de características, que funciona como entrada da MLP



- “Full Connection”

- Para essa etapa pode ser usada uma MLP



Resposta:

```
[52] import tensorflow as tf
     import numpy as np
     import pandas as pd
     import seaborn as sns
     from keras.layers import Dropout
     from sklearn.metrics import confusion_matrix

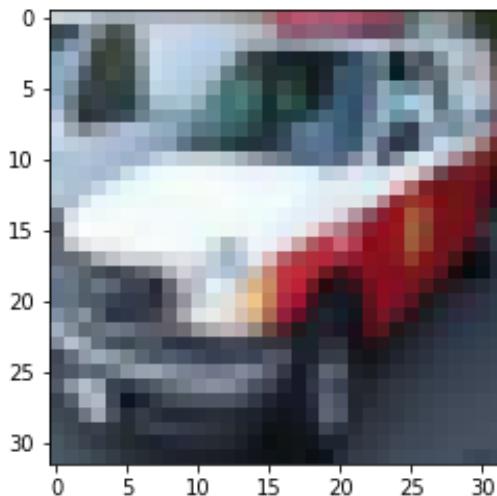
[53] cifar10 = tf.keras.datasets.cifar10

[54] (X_train, y_train) , (X_test, y_test) = cifar10.load_data()

[55] print(X_train.shape)
     (50000, 32, 32, 3)

[56] print(X_test.shape)
     (10000, 32, 32, 3)

[57] import matplotlib.pyplot as plt
     plt.imshow(X_train[60,:])
```

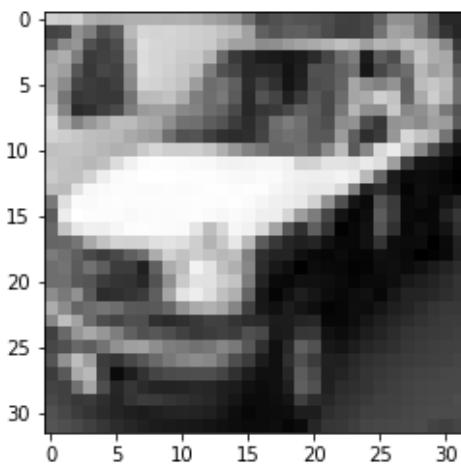


```
[58] X_train = X_train[:, :, :, 1]
      X_test = X_test[:, :, :, 1]
```

```
[59] print( X_train.shape )
      print( X_test.shape )
```

```
(50000, 32, 32)
(10000, 32, 32)
```

```
● plt.imshow(X_train[60,:], cmap='gray')
```



```
[61] X_train = X_train.reshape(X_train.shape[0], 32, 32, 1 )
      X_test = X_test.reshape(X_test.shape[0], 32, 32, 1 )
```

```
[62] X_train = X_train.astype('float32')
      X_test = X_test.astype('float32')
```

```
[63] X_train = X_train / 255
      X_test = X_test / 255
```

```
[64] y_test[0:2]

array([[3],
       [8]], dtype=uint8)
```

```
[65] num_classes = 10
      y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

```
[66] y_train = tf.keras.utils.to_categorical(y_train, num_classes)
```

```
[67] y_test[0:2]

array([[0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0.]], dtype=float32)
```

```
[71] # 2 camadas convolucionais (30 filtros na primeira camada e 15 na segunda)
# 2 camadas de max pooling
# 1 camada de flattening
# Full connection (MLP) => 1 camada oculta de 500 neurônios

# Dropout => realiza o desligamento temporário de alguns neurônios, baseado na porcentagem indicada
# O dropout faz com que alguns neurônios aprendam mais sobre determinadas características do que outros,
# durante o treinamento. Alguns neurônios ficam mais especializados em determinadas características.
# O dropout também ajuda a rede a não decorar, evitando o overfitting.

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(30, (5,5), input_shape=(32,32,1), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
#model.add(Dropout(0.25))
model.add(tf.keras.layers.Conv2D(15, (3,3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
#model.add(Dropout(0.25))
model.add(tf.keras.layers.Flatten())
# Hidden layer
model.add(tf.keras.layers.Dense(500, activation='relu'))
# Output layer
# num_classes = 10 (resultados possíveis)
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))
model.compile(tf.keras.optimizers.Adam(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/optimizer_v2.py:356: UserWarning: The `lr` arg
"The `lr` argument is deprecated, use `learning_rate` instead.")
```

```
[72] print(model.summary())
```

Model: "sequential_13"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_28 (Conv2D)	(None, 28, 28, 30)	780
max_pooling2d_26 (MaxPooling2D)	(None, 14, 14, 30)	0
conv2d_29 (Conv2D)	(None, 12, 12, 15)	4065
max_pooling2d_27 (MaxPooling2D)	(None, 6, 6, 15)	0
flatten_12 (Flatten)	(None, 540)	0
dense_28 (Dense)	(None, 500)	270500
dense_29 (Dense)	(None, 10)	5010
<hr/>		
Total params: 280,355		
Trainable params: 280,355		
Non-trainable params: 0		
<hr/>		
None		

```
[73] # 10 épocas
# Validação com 10% dos dados no final das épocas

history = model.fit(X_train, y_train, epochs=10, validation_split=0.1, batch_size=400, verbose=1, shuffle=1)

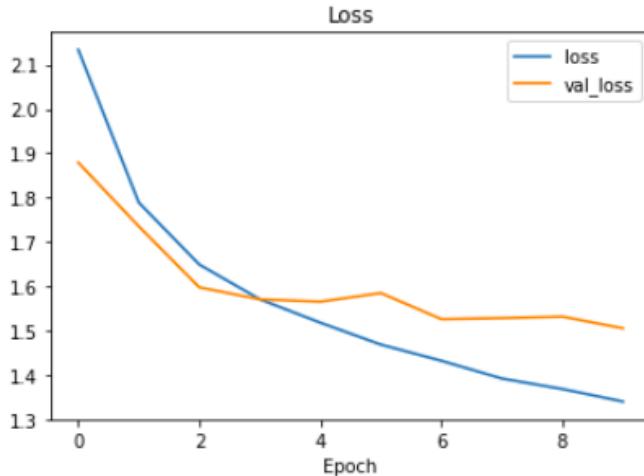
Epoch 1/10
113/113 [=====] - 31s 268ms/step - loss: 2.1337 - accuracy: 0.2212 - val_loss: 1.8789 - val_accuracy: 0.3234
Epoch 2/10
113/113 [=====] - 30s 267ms/step - loss: 1.7885 - accuracy: 0.3555 - val_loss: 1.7348 - val_accuracy: 0.3808
Epoch 3/10
113/113 [=====] - 30s 265ms/step - loss: 1.6485 - accuracy: 0.4134 - val_loss: 1.5970 - val_accuracy: 0.4326
Epoch 4/10
113/113 [=====] - 30s 266ms/step - loss: 1.5701 - accuracy: 0.4414 - val_loss: 1.5700 - val_accuracy: 0.4336
Epoch 5/10
113/113 [=====] - 30s 265ms/step - loss: 1.5173 - accuracy: 0.4629 - val_loss: 1.5648 - val_accuracy: 0.4406
Epoch 6/10
113/113 [=====] - 30s 266ms/step - loss: 1.4678 - accuracy: 0.4825 - val_loss: 1.5842 - val_accuracy: 0.4370
Epoch 7/10
113/113 [=====] - 30s 266ms/step - loss: 1.4313 - accuracy: 0.4955 - val_loss: 1.5253 - val_accuracy: 0.4698
Epoch 8/10
113/113 [=====] - 30s 263ms/step - loss: 1.3912 - accuracy: 0.5077 - val_loss: 1.5277 - val_accuracy: 0.4626
Epoch 9/10
113/113 [=====] - 30s 265ms/step - loss: 1.3675 - accuracy: 0.5194 - val_loss: 1.5310 - val_accuracy: 0.4706
Epoch 10/10
113/113 [=====] - 30s 266ms/step - loss: 1.3393 - accuracy: 0.5303 - val_loss: 1.5046 - val_accuracy: 0.4730
```

▼ Avaliação do treinamento

✓ [74] # loss => é o valor da função custo para o conjunto de treino
val_loss => é valor da função custo o cross-validation
Quanto menor esses valores melhor o resultado

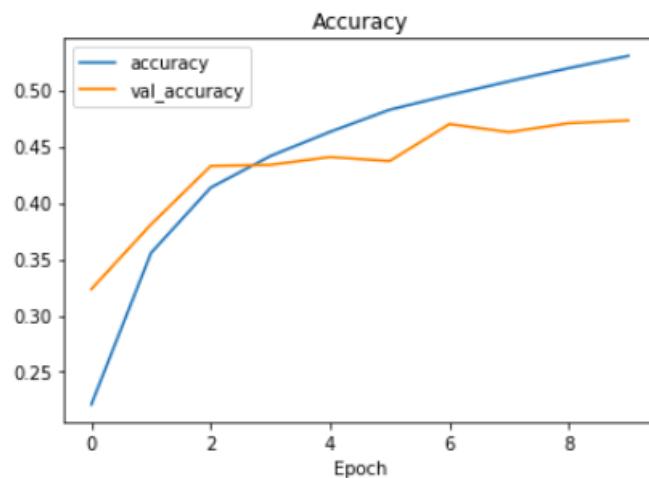
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.title('Loss')
plt.xlabel('Epoch')
```

Text(0.5, 0, 'Epoch')



```
[75] # accuracy => é o valor da acurácia para o conjunto de treino  
# val_accuracy => é valor da acurácia para o cross-validation  
  
plt.plot(history.history['accuracy'])  
plt.plot(history.history['val_accuracy'])  
plt.legend(['accuracy', 'val_accuracy'])  
plt.title('Accuracy')  
plt.xlabel('Epoch')
```

Text(0.5, 0, 'Epoch')



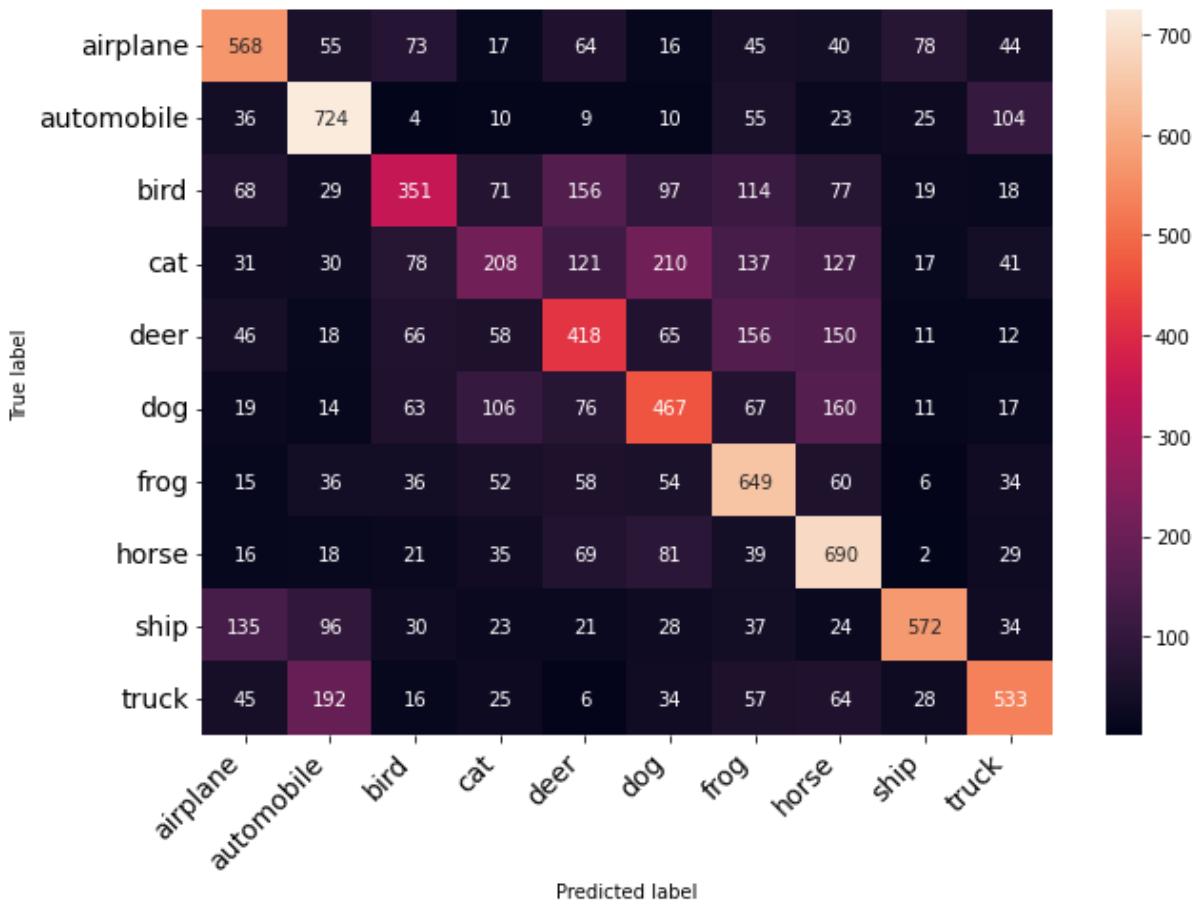
```
[76] # Label Names(not gaven as metadata)  
Label_names = [  
    'airplane',  
    'automobile',  
    'bird',  
    'cat',  
    'deer',  
    'dog',  
    'frog',  
    'horse',  
    'ship',  
    'truck',  
]  
y_pred = np.argmax(model.predict(X_test), axis=1)  
y_label = np.argmax(y_test, axis=1)
```

```

def print_confusion_matrix(confusion_matrix, class_names,
                           figsize = (10,7), fontsize=14):

    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(),
                                rotation=0, ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,
                                ha='right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

```



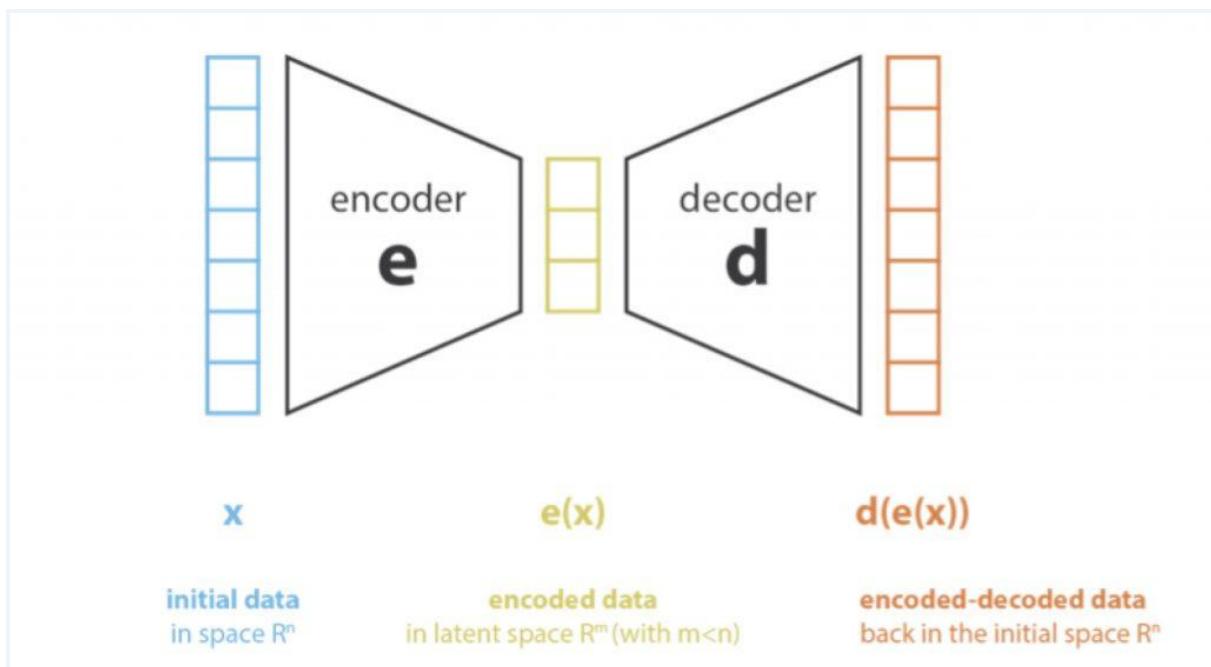
Código:

<https://colab.research.google.com/drive/10G344X4sq-8FTQm6vf5km6ShMuxCsOP5?usp=sharing>

3º)

Considere quatro distribuições gaussianas, C_1, C_2, C_3 , e C_4 , em um espaço de entrada de dimensionalidade igual a oito, isto é $\mathbf{x} = (x_1, x_2, \dots, x_8)^t$. Todas as nuvens de dados formadas têm variâncias unitária, mas os centros (vetores média) são diferentes e dados por $\mathbf{m}_1 = (0,0,0,0,0,0,0,0)^t, \mathbf{m}_2 = (4,0,0,0,0,0,0,0)^t, \mathbf{m}_3 = (0,0,0,4,0,0,0,0)^t, \mathbf{m}_4 = (0,0,0,0,0,0,0,4)^t$.

Utilize uma rede de autoencoder para reduzir a dimensionalidade e possibilitar com isto a visualização dos dados em duas dimensões. O objetivo é visualizar os dados de dimensão 8 em um espaço de dimensão 2. Esboce um gráfico com as distribuições no novo espaço (bidimensional).



Resposta:

```
[30] import numpy as np
from numpy import mean
from numpy import std
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf
import pandas as pd

# Vetores média que representam os centros das nuvens de
# distribuições gaussianas

m1 = np.array([0,0,0,0,0,0,0,0]).T
m2 = np.array([4,0,0,0,0,0,0,0]).T
m3 = np.array([0,0,0,4,0,0,0,0]).T
m4 = np.array([0,0,0,0,0,0,0,4]).T
```

```
# Gerando as nuvens de dados

nuvem_dados_1 = np.random.normal(m1,1,(2000,8))
nuvem_dados_2 = np.random.normal(m2,1,(2000,8))
nuvem_dados_3 = np.random.normal(m3,1,(2000,8))
nuvem_dados_4 = np.random.normal(m4,1,(2000,8))
```

▼ Criação do Dataset

```
[31] dataset = np.concatenate([nuvem_dados_1,nuvem_dados_2,nuvem_dados_3,nuvem_dados_4])
```

▼ Estrutura da Rede

```
[32] # Rede Autoencoder
```

```
model = Sequential()
model.add(Dense(8,activation='linear',input_dim=8))
model.add(Dense(2, activation='linear'))
model.add(Dense(8,activation='linear'))
```

```
model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_3 (Dense)	(None, 8)	72
dense_4 (Dense)	(None, 2)	18
dense_5 (Dense)	(None, 8)	24
<hr/>		
Total params:	114	
Trainable params:	114	
Non-trainable params:	0	

▼ Treino

7m

```
model.compile(loss=keras.losses.mean_squared_error, metrics=['accuracy'])
model.fit(dataset,dataset,epochs=2000,verbose=False)

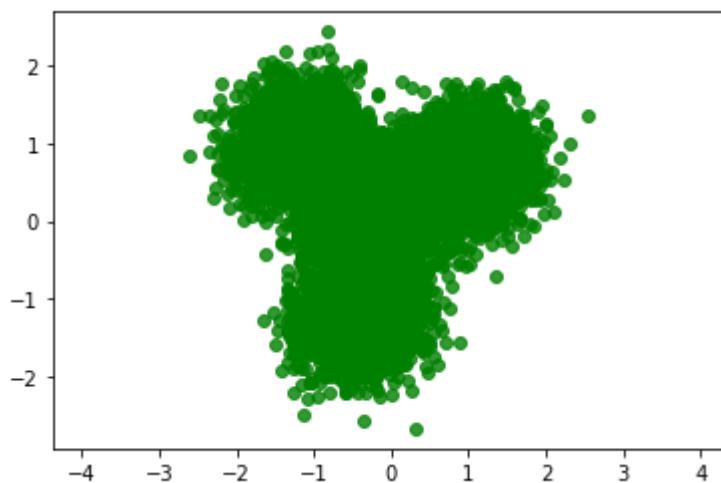
<keras.callbacks.History at 0x7fd2bf385b90>
```

▼ Teste

```
✓ [41] from keras import backend as knd  
0s  
get_first_layer_output = knd.function([model.layers[0].input],  
                                     [model.layers[1].output])  
  
layer_output = get_first_layer_output([dataset])  
layer_output = np.array(layer_output[0])  
  
predict = model.predict(dataset)  
np.mean(predict-dataset)  
  
-0.0016884817445840046
```

Visualização dos Dados em 2 Dimensões

```
[44] plt.scatter(layer_output[:,0], layer_output[:,1], alpha=0.8, c='g')  
plt.axis('equal');
```



Código:https://colab.research.google.com/drive/1eYBeGFXFIxN2JlnurX6Ye_RUaTUk2-Q8?usp=sharing

4º)

Considere o problema de predição de uma série temporal definida como $x(n) = v(n) + \beta v(n-1)v(n-2)$, com média zero e variância dada por $\sigma_x^2 = \sigma_v^2 + \beta^2 \sigma_v^2$ onde $v(n)$ é um ruído branco gaussiano, com variância unitária e $\beta = 0.5$. Utilizando uma rede LSTM estime $x(n+1)=f(x(n), x(n-1), x(n-2), x(n-3), y(n), y(n-1), y(n-2))$. Esboce a curva da série e a curva de predição da série com relação a n. Esboce também o erro de predição. No treinamento utilize a estratégia da resposta forçada do professor. Isto é na estimativa de $x(n+1)=f(x(n), x(n-1), x(n-2), x(n-3), d(n), d(n-1), d(n-2))$. Onde $d(n)$ é resposta desejada, isto é, $d(n)=x(n+1)$. Avalie também predição de 2 passos $x(n+2)$ e a predição de três passos $x(n+3)$ apresentando o erro de predição para cada uma das situações.

Resposta:

- 3200 elementos para treino e predição dos últimos 800 elementos

Setup

```
[1] !pip install -q tensorflow-gpu
|██████████| 458.3 MB 10 kB/s

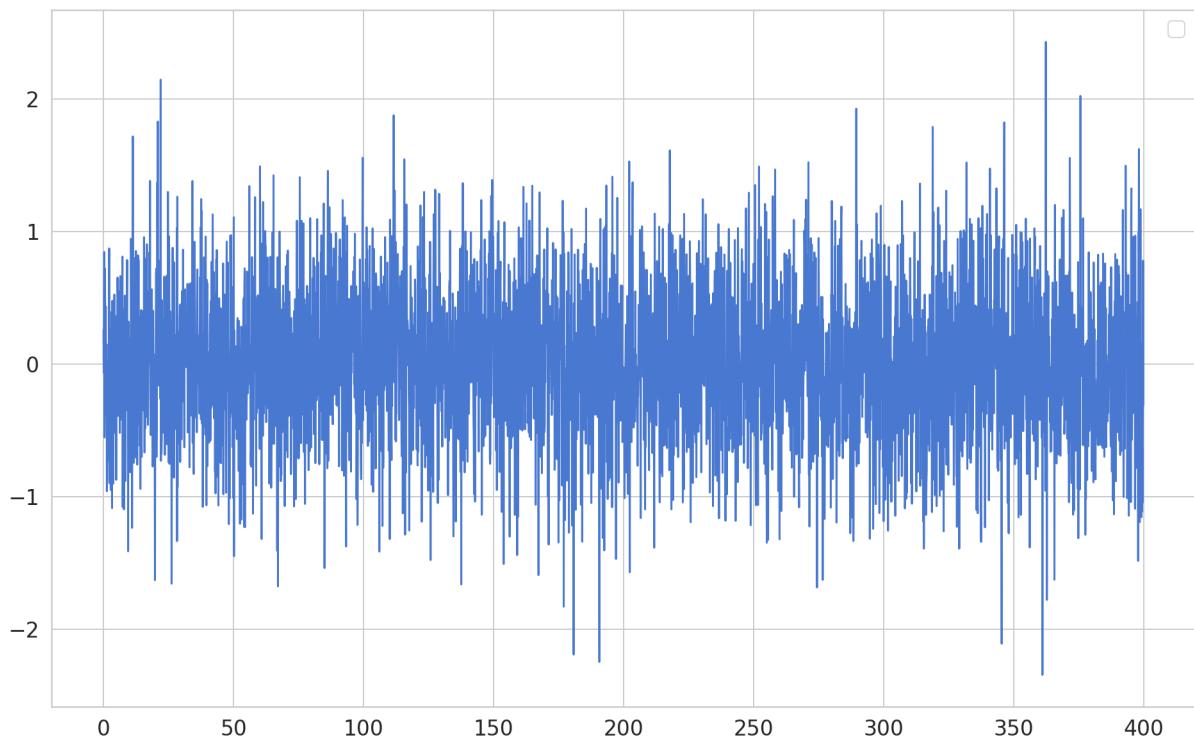
[2] import numpy as np
import math
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
%matplotlib inline
%config InlineBackend.figure_format='retina'
sns.set(style='whitegrid', palette='muted', font_scale=1.5)
rcParams['figure.figsize'] = 16, 10
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)
from tensorflow.keras import layers
from tensorflow.keras.layers import Embedding, Dense, LSTM
```

▼ Predição da Série Temporal em Análise

▼ Série Temporal em Análise

```
✓ [3] beta = 1
    n = np.arange(0, 400, 0.1)
    v_n = np.random.normal(scale=0.5, size=len(n))
    v_n_1 = np.random.normal(scale=0.5, size=len(n-1))
    v_n_2 = np.random.normal(scale=0.5, size=len(n-2))
    x = v_n + beta*v_n_1*v_n_2
```

```
✓ [4] plt.plot(n, x)
    plt.legend();
```



▼ Data Preprocessing

```
[5] df = pd.DataFrame(dict(X_x=x), index=n, columns=['X_x'])
df.head()
```

	X_x
0.0	0.255486
0.1	-0.065203
0.2	0.323068
0.3	0.845966
0.4	-0.553739

```
[6] train_size = int(len(df) * 0.8)
test_size = len(df) - train_size
train, test = df.iloc[0:train_size], df.iloc[train_size:len(df)]
print(len(train), len(test))
```

```
3200 800
```

```
[7] def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        v = X.iloc[i:(i + time_steps)].values
        Xs.append(v)
        ys.append(y.iloc[i + time_steps])
    return np.array(Xs), np.array(ys)
```

```
[8] time_steps = 10
# reshape to [samples, time_steps, n_features]
X_train, y_train = create_dataset(train, train.X_x, time_steps)
X_test, y_test = create_dataset(test, test.X_x, time_steps)
print(X_train.shape, y_train.shape)
```

```
(3190, 10, 1) (3190,)
```

```
[9] # reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

```
[10] X_train[0]
```

```
array([[ 0.25548635, -0.06520324,  0.32306785,  0.8459659 , -0.55373895,
       -0.03255545,  0.72083156,  0.14782375, -0.19654973,  0.43450724]])
```

```
[11] y_train[0]
```

```
-0.23333862309402473
```

▼ Predição da Série Temporal com LSTM

```
✓ [12] inputs = tf.keras.Input(shape=(X_train.shape[1], X_train.shape[2]))
      x = layers.Bidirectional(layers.LSTM(64, return_sequences=False))(inputs)
      #x = layers.Dropout(0.2)(x)
      outputs = layers.Dense(1, activation="linear")(x)
      model = tf.keras.Model(inputs, outputs)

      model.compile(loss='mean_squared_error', optimizer=keras.optimizers.Adam(0.001))
```

```
✓ [13] model.summary()
```

Model: "model"

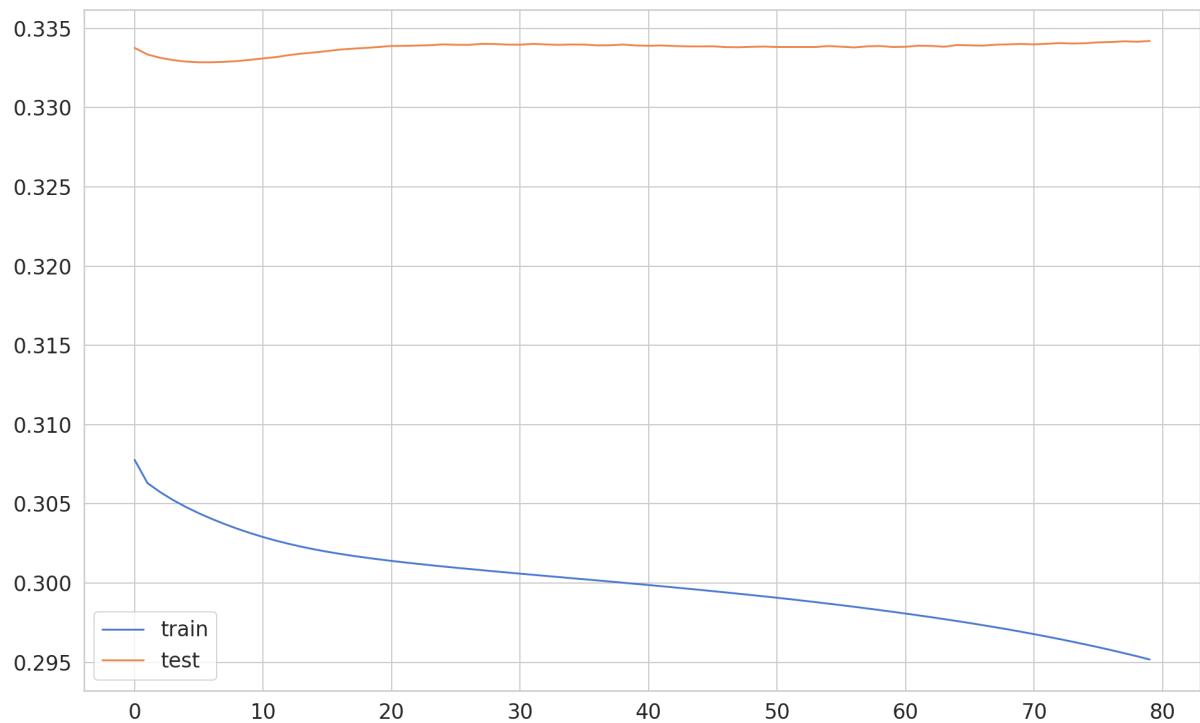
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 1, 10)]	0
bidirectional (Bidirectional (None, 128)		38400
dense (Dense)	(None, 1)	129
Total params:	38,529	
Trainable params:	38,529	
Non-trainable params:	0	

▼ Training

```
✓ [14] history = model.fit(
      np.asarray(X_train).astype('float32'),
      np.asarray(y_train).astype('float32'),
      epochs=80,
      batch_size=16,
      validation_split=0.1,
      verbose=1,
      shuffle=False
    )
```

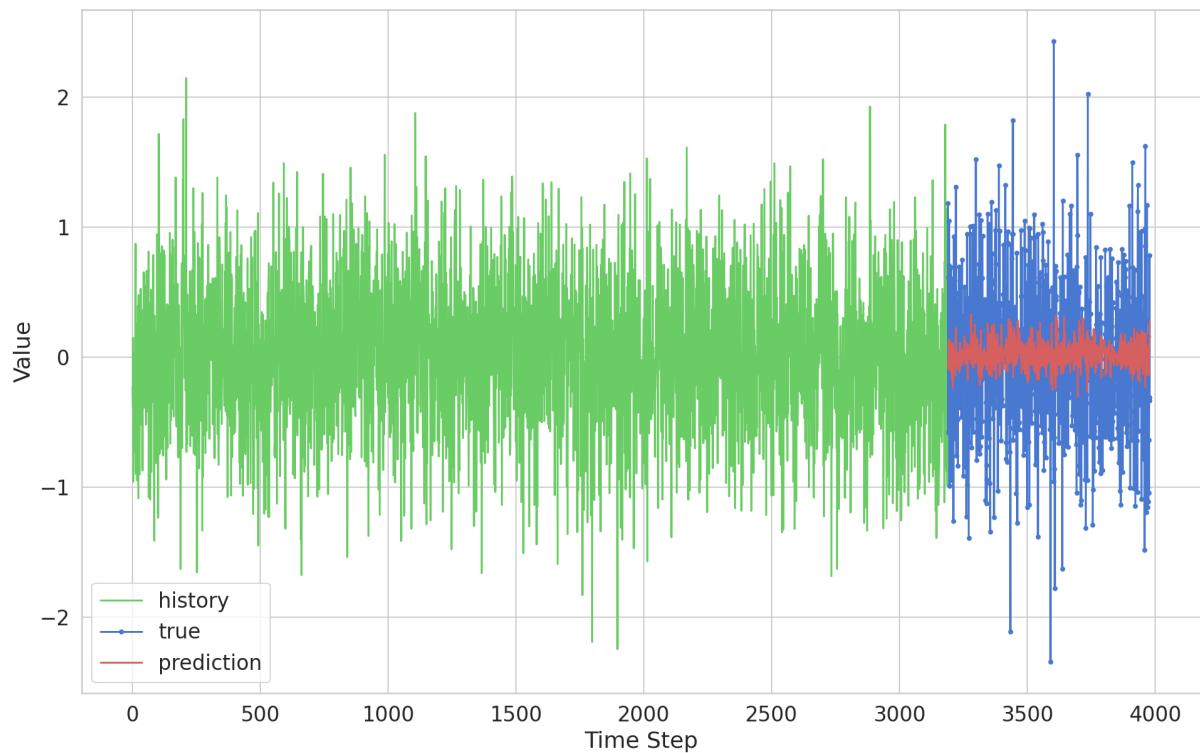
▼ Evaluation

```
✓ [15] plt.plot(history.history['loss'], label='train')
      plt.plot(history.history['val_loss'], label='test')
      plt.legend();
```

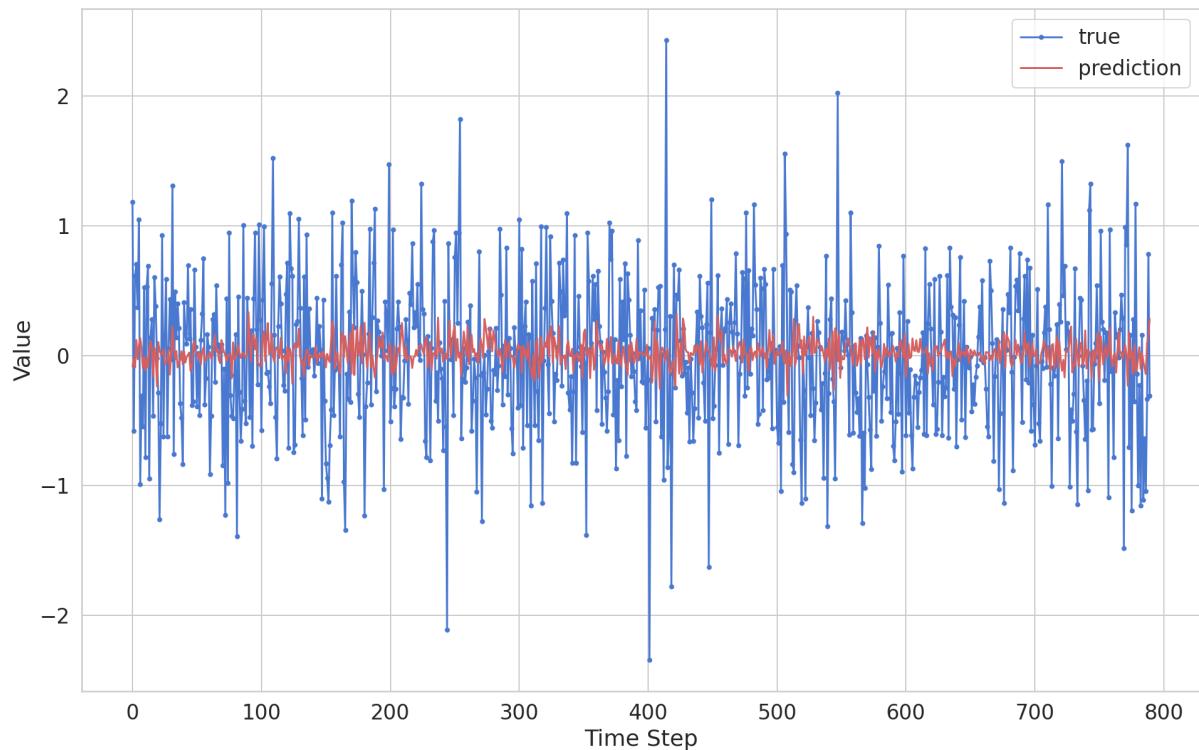


```
✓ [16] y_pred = model.predict(np.asarray(X_test).astype('float32'))
```

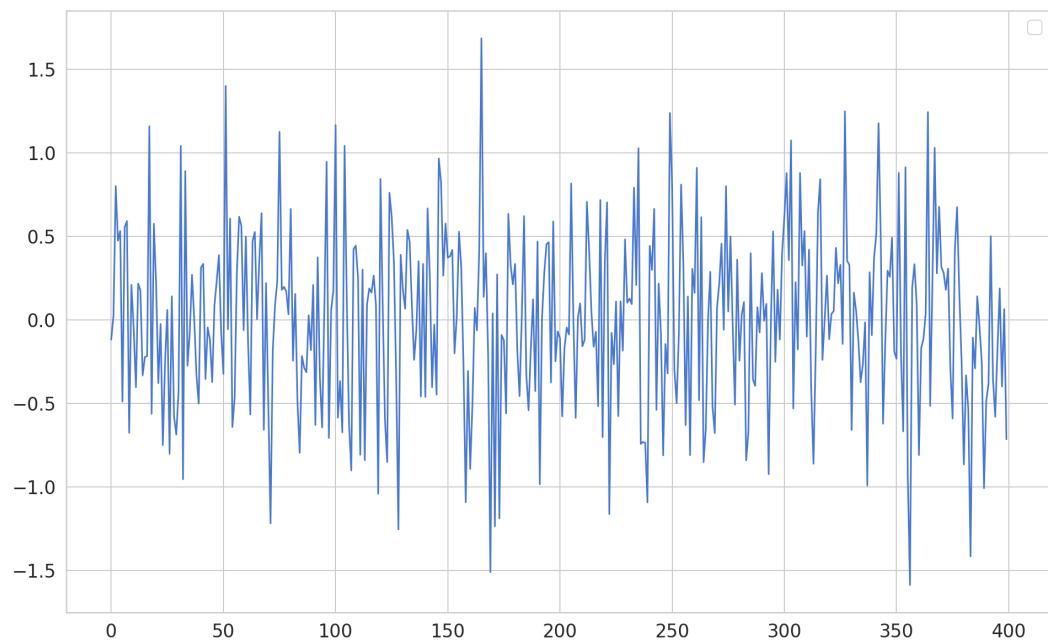
```
✓ [17] plt.plot(np.arange(0, len(y_train)), y_train, 'g', label="history")
    plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)), y_test, marker='.', label="true")
    plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)), y_pred, 'r', label="prediction")
    plt.ylabel('Value')
    plt.xlabel('Time Step')
    plt.legend()
    plt.show();
```

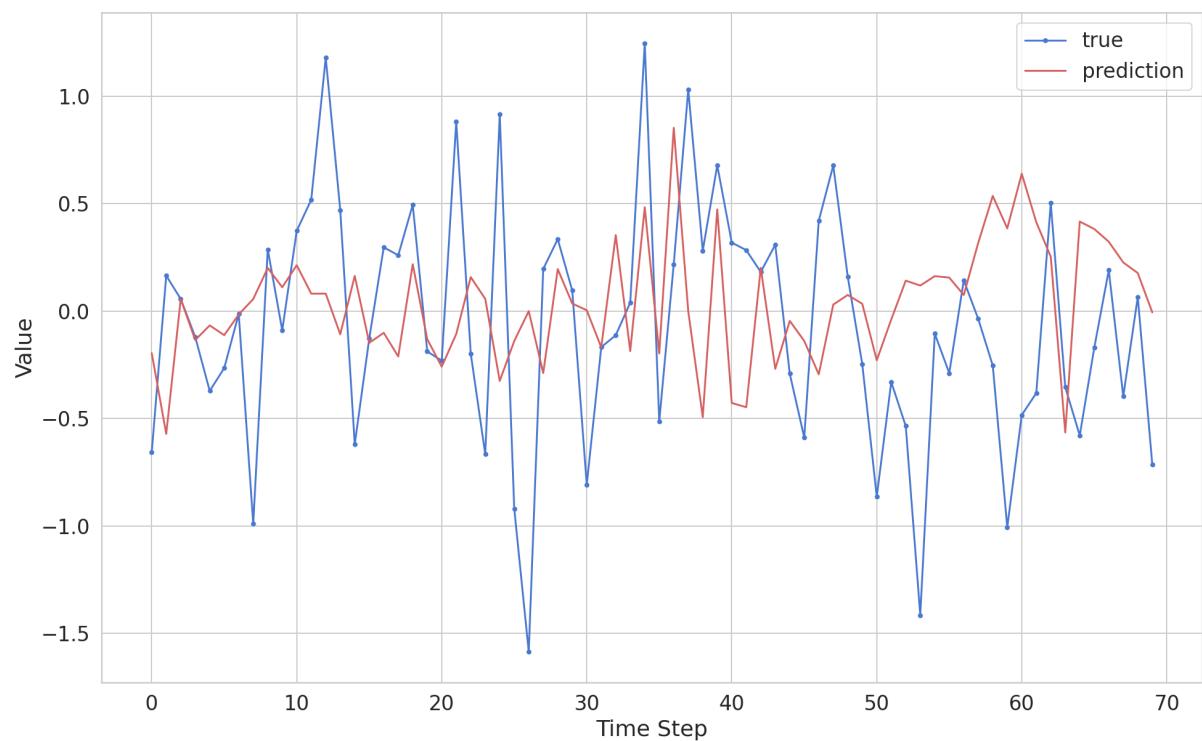
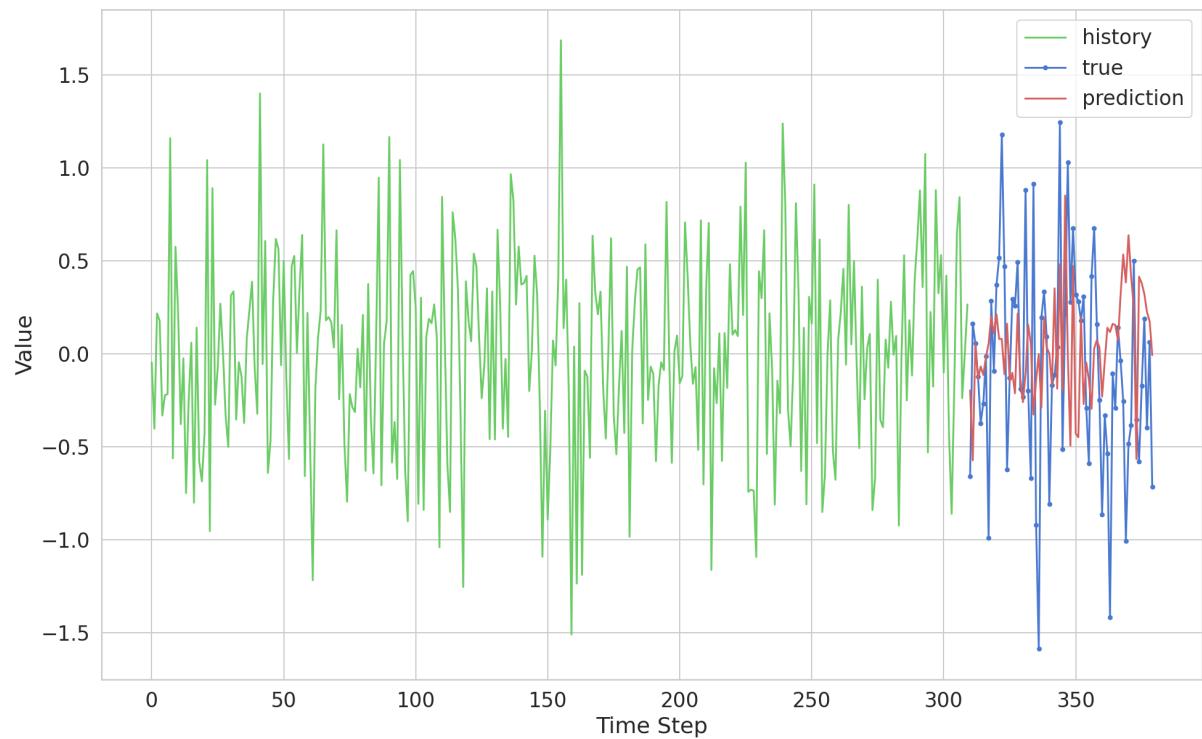


```
[18] plt.plot(y_test, marker='.', label="true")
    plt.plot(y_pred, 'r', label="prediction")
    plt.ylabel('Value')
    plt.xlabel('Time Step')
    plt.legend()
    plt.show();
```

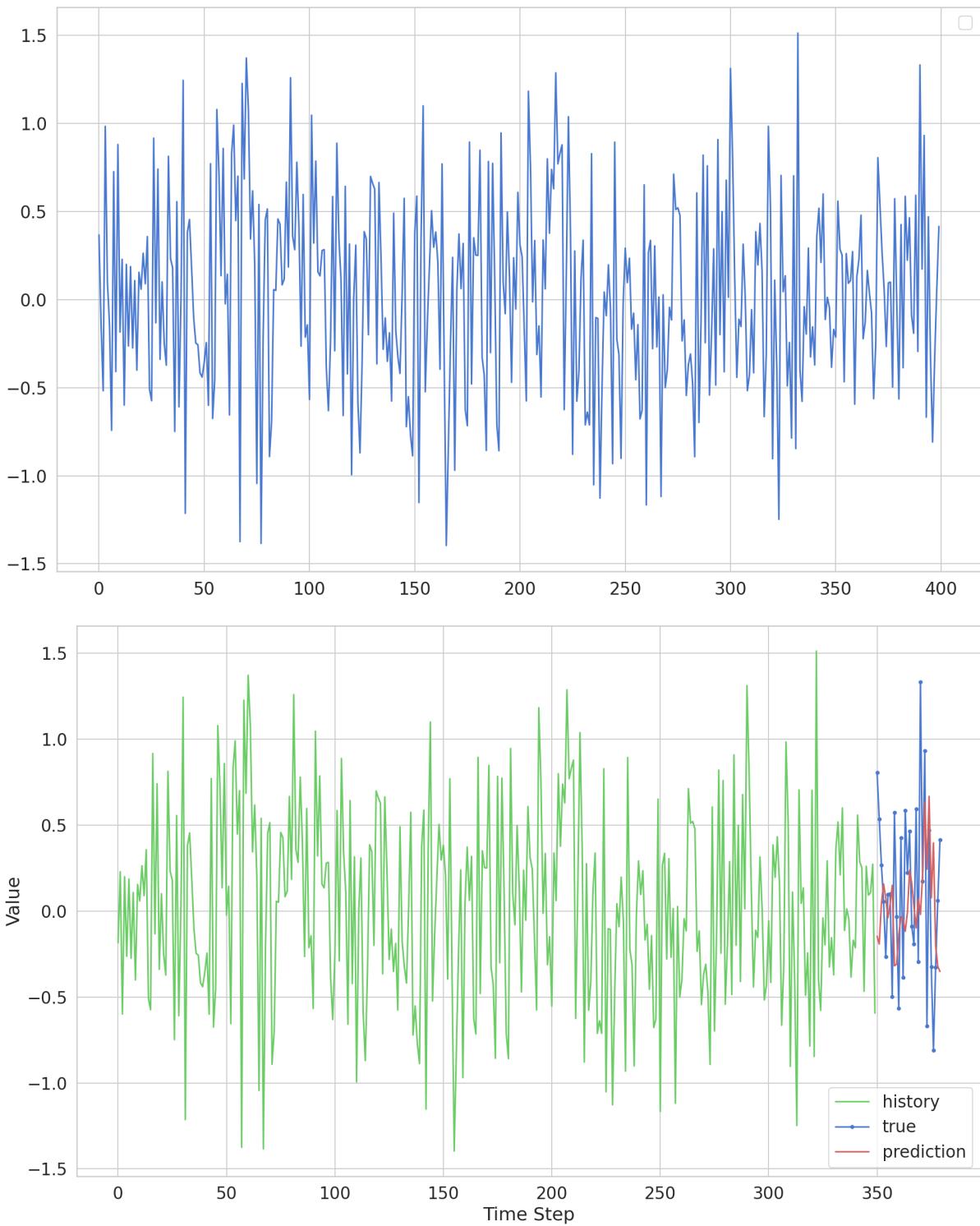


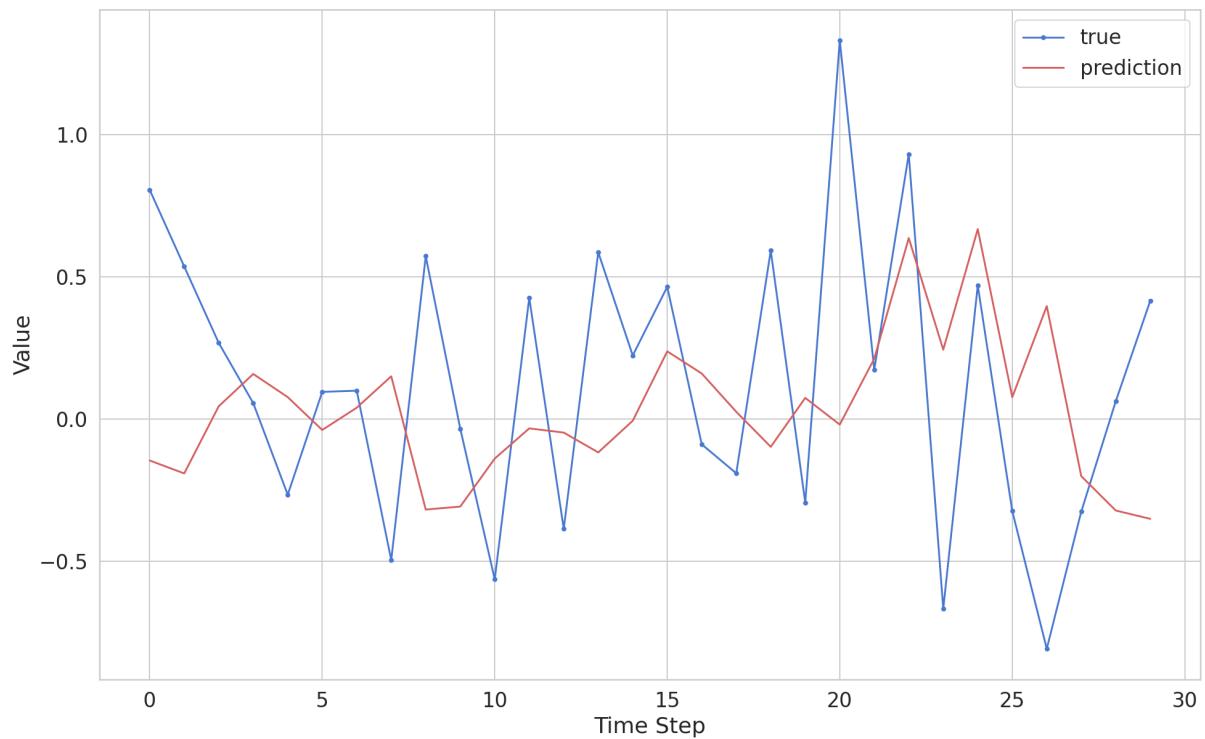
- 320 elementos para treino e predição dos últimos 80 elementos



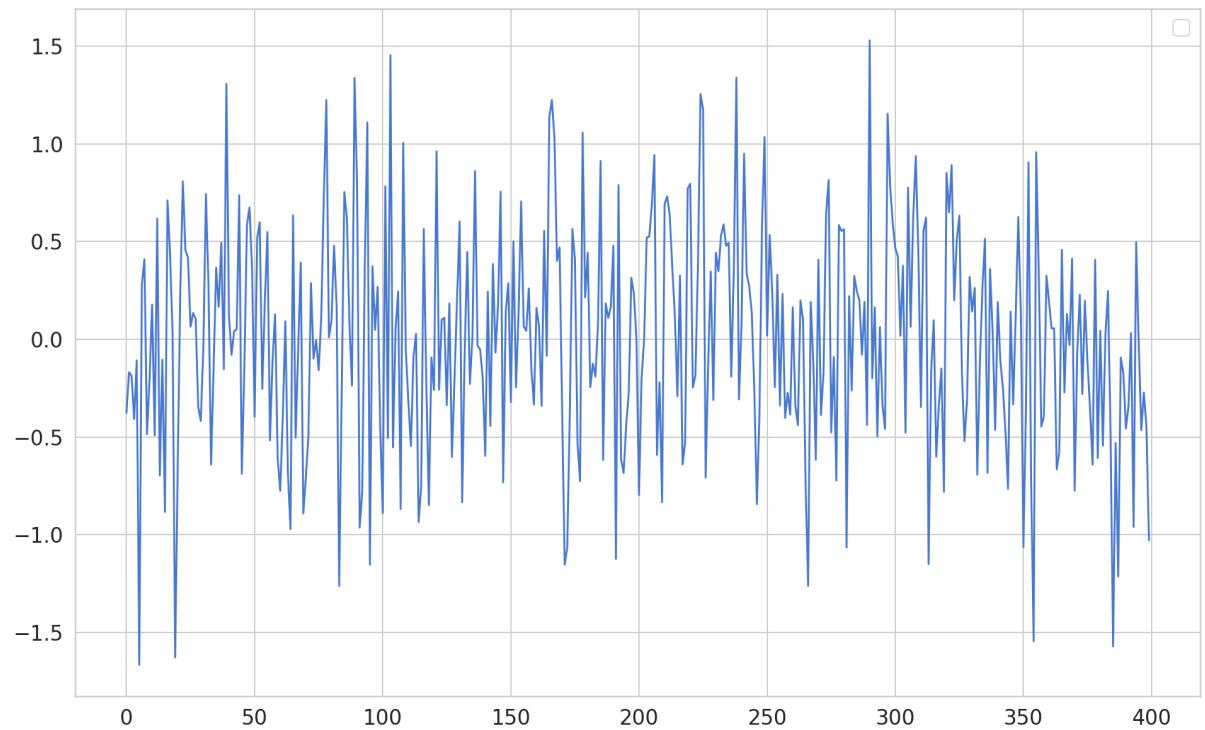


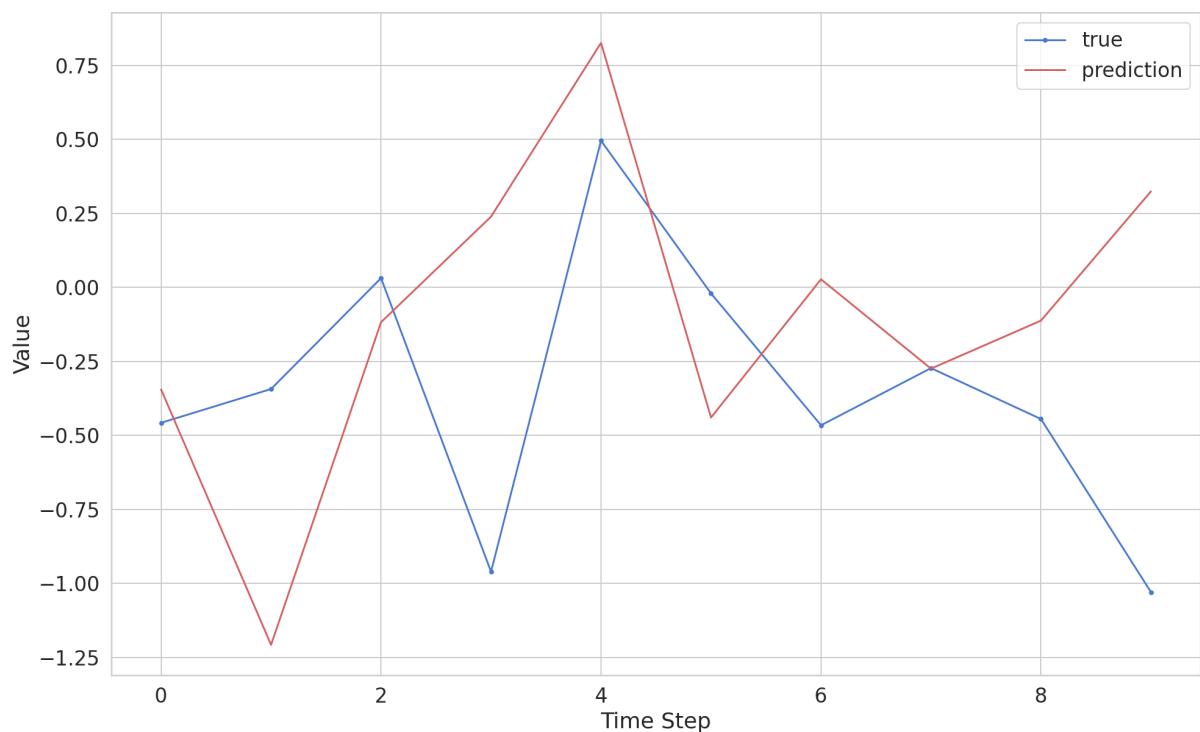
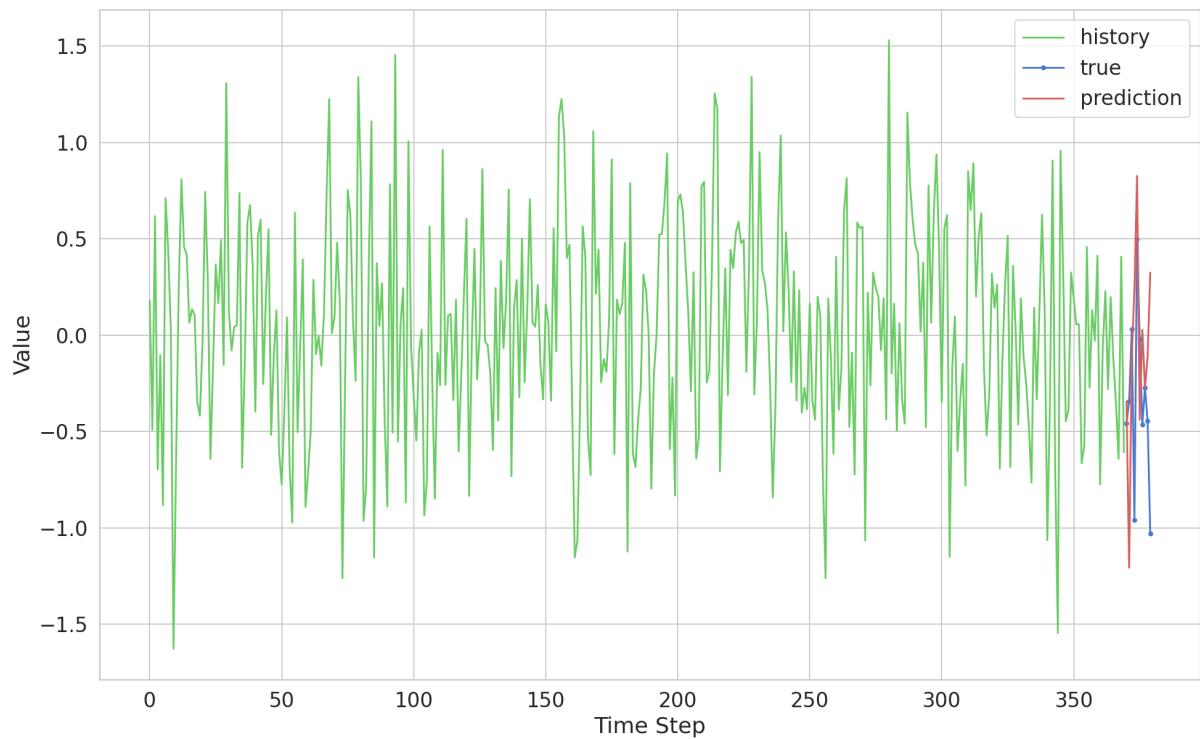
- 360 elementos para treino e predição dos últimos 40 elementos





- 380 elementos para treino e predição dos últimos 20 elementos





Código:<https://colab.research.google.com/drive/1iF4U8D9M1lv6tjflh3N5m8BCC7lzNgXJ?usp=sharing>

5º)

5-) Pesquise sobre redes neurais recorrentes LSTM e variações como a rede GRU. Apresente neste estudo aplicações das LSTM deep learning. Seguem abaixo sugestões de aplicações. Escolha uma ou mais

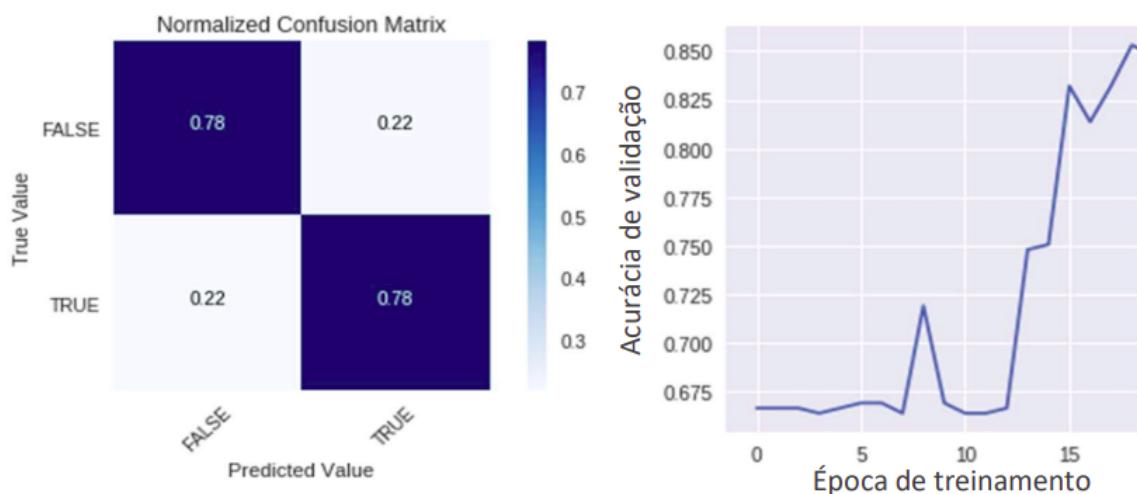
Resposta:

As Redes de Memória de Curto Prazo Longo (LSTMs) são um tipo especial de RNN, capaz de aprender dependências de longo prazo. Eles foram introduzidos por Hochreiter & Schmidhuber (1997) e foram refinados e popularizados e atualmente funcionam bem em uma grande variedade de problemas.

Os LSTMs são projetados para evitar o problema de dependência de longo prazo, uma vez que seu comportamento esperado já é o de “lembra” informações por longos períodos de tempo.

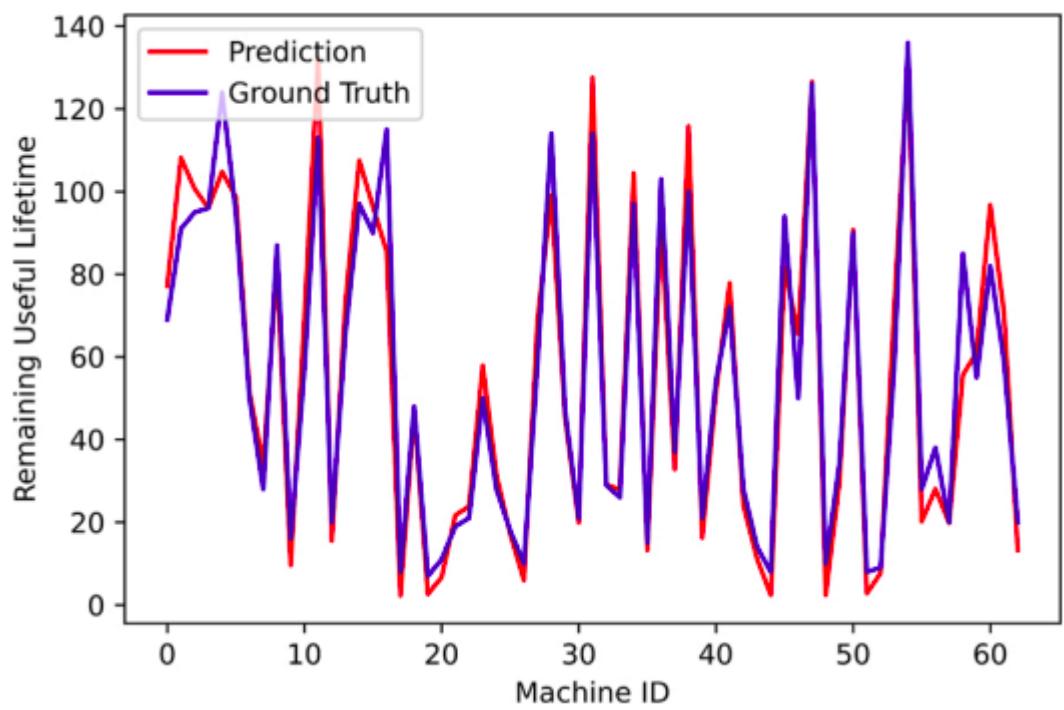
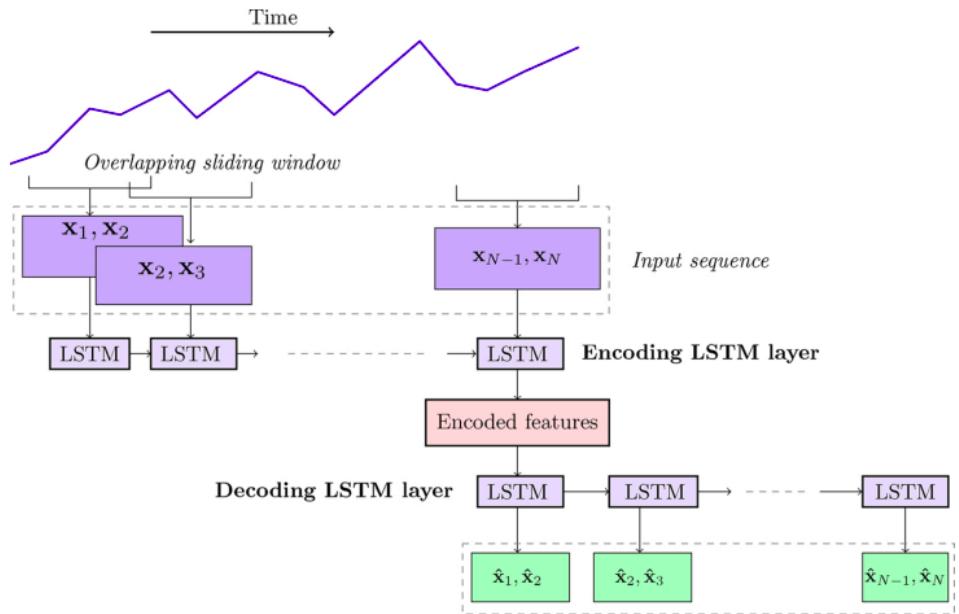
Uma variação no LSTM é a Unidade Recorrente Gated, ou GRU, que combina as portas de esquecimento e de entrada em uma única “porta de atualização”. Ele também mescla o estado da célula e o estado oculto, resultando em um modelo mais simples do que os modelos LSTM padrão e se tornando cada vez mais popular.

Para exemplificar o uso das LSTMs, podemos citar o estudo realizado por ALVES (2019) que implementou uma estrutura com fator multiplicativo em modelos com embeddings e redes recorrentes para detecção de fake news na língua portuguesa.



O autor obteve uma acurácia de cerca de 83% usando LSTM, enquanto usando uma RNN tradicional o resultado foi de 60%.

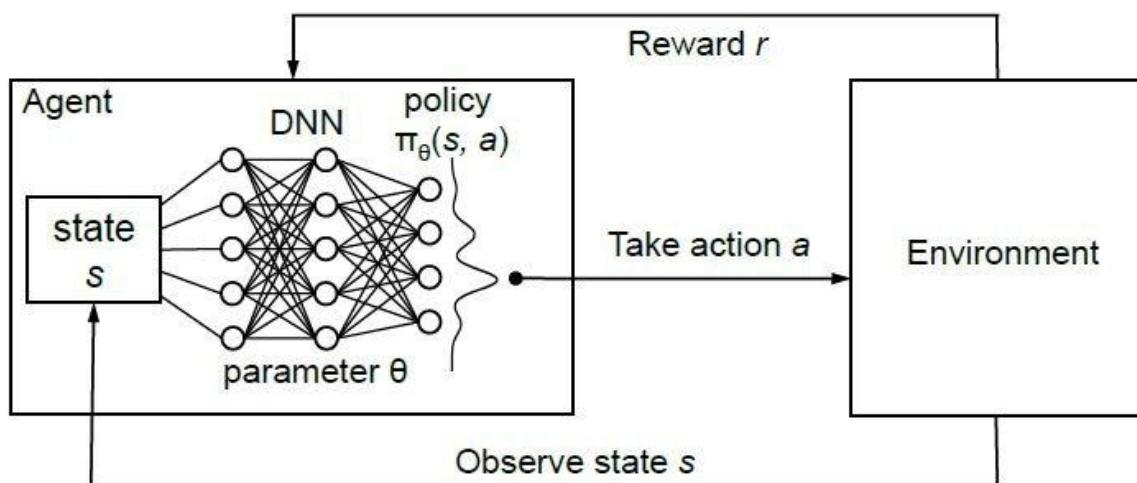
Um outro exemplo é o de Hamad (2021) que usou as LSTMs para realizar previsões do tempo.



7º)

Apresente os fundamentos da aprendizagem por reforço profundo. Exemplifique uma aplicação. Sugestão: Pesquise e aplique a técnica aprendizagem por reforço profundo em problema de livre escolha.

Sugestão: Aplique em um jogo (game) da sua escolha.



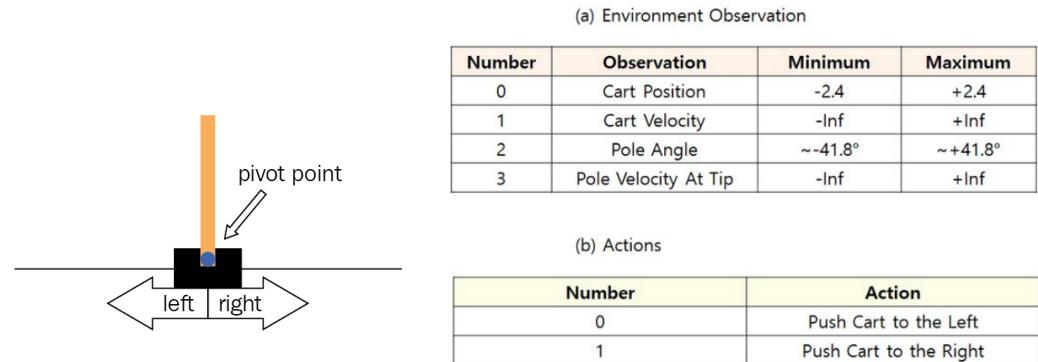
Deep reinforcement learning (aprendizado por reforço profundo) é uma técnica de IA (inteligência artificial), que combina reinforcement learning (aprendizado por reforço) e deep learning (aprendizado profundo). Nesta técnica um agente, composto por uma estrutura de deep learning (as entradas são os estados e as saídas são as ações) e uma política, interage com um ambiente, sendo recompensado ou penalizado quando, respectivamente, acerta ou erra na tomada de decisão.

A política em deep reinforcement learning funciona como uma estratégia do agente que ajuda no processo de realizar ações que geram recompensas, ou seja, acertar o objetivo. Um exemplo de política, nesse contexto, pode ser apresentado através de um robô (agente) inserido em uma sala (environment) com o objetivo de alcançar uma posição predefinida (x, y). Nesse caso, a posição do robô indica o estado e a movimentação dele representa a ação. A política é a estratégia que o robô irá usar para chegar no objetivo, podendo existir algumas mais eficientes do que outras.

O reward é uma função de recompensa responsável por descrever como o agente precisa se comportar, baseado em quão bem ele está interagindo no environment em direção ao objetivo.

Resposta:

DRL - CartPole-v0 (OpenAI gym)



```
In [1]: !pip install tensorflow==2.3.0  
!pip install gym  
!pip install keras  
!pip install keras-rl2  
!pip install pyglet
```

Environment (CartPole-v0) with OpenAI Gym

```
In [4]: import gym  
import random
```

```
In [5]: env = gym.make('CartPole-v0')  
states = env.observation_space.shape[0]  
actions = env.action_space.n
```

```
In [6]: # Cart position  
# Cart velocity  
# Pole angle  
# Pole velocity at tip  
print('States:', states)
```

States: 4

```
In [7]: # push cart to the left  
# push cart to the right  
print('Actions:', actions)
```

Actions: 2

Test Before Use Deep Reinforcement Learning (DRL)

```
In [8]: episodes = 30 # match number
for episode in range(1, episodes+1):
    state = env.reset()
    done = False
    score = 0

    while not done:
        env.render()
        action = random.choice([0,1])
        n_state, reward, done, info = env.step(action)
        score+=reward
    print('Episode:{} Score:{}'.format(episode, score))
```

Antes do treinamento com DRL: <https://youtu.be/WScxZpmv8hA>

Deep Learning Model (MLP) with Keras

```
In [25]: import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
```

```
In [26]: # MLP => 2 hidden layers with 24 neurons in each one
#           4 neurons for each state in the input layer
#           2 neurons for each action in the output layer
#           activation function for all the hidden layers = relu
def build_model(states, actions):
    model = tf.keras.Sequential()
    model.add(Flatten(input_shape=(1,states)))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(actions, activation='linear'))
    return model
```

```
In [32]: # MLP structure
model = build_model(states, actions)
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 4)	0
dense_6 (Dense)	(None, 24)	120
dense_7 (Dense)	(None, 24)	600
dense_8 (Dense)	(None, 2)	50

Total params: 770
Trainable params: 770
Non-trainable params: 0

Build Agent with Keras-DRL (Deep Reinforcement Learning)

```
In [33]: from rl.agents import DQNAgent
from rl.policy import BoltzmannQPolicy
from rl.memory import SequentialMemory

In [34]: def build_agent(model, actions):
    policy = BoltzmannQPolicy()
    memory = SequentialMemory(limit=50000, window_length=1)
    dqn = DQNAgent(model=model, memory=memory, policy=policy,
                   nb_actions=actions, nb_steps_warmup=10, target_model_update=1e-2)
    return dqn
```

DRL Training

```
In [35]: dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])
dqn.fit(env, nb_steps=50000, visualize=False, verbose=1)
```

Durante o treinamento com DRL: <https://youtu.be/B6e1GC2PKzw>

DEL Test

```
In [36]: scores = dqn.test(env, nb_episodes=100, visualize=False)
print(np.mean(scores.history['episode_reward']))
```

```
In [37]: _ = dqn.test(env, nb_episodes=15, visualize=True)
```

Após o treinamento com DRL: <https://youtu.be/NoZHYuzhT2o>

Reloading Agent from Memory (Saving Weights)

```
In [38]: dqn.save_weights('dqn_weights.h5f', overwrite=True)
```

```
In [39]: del model
del dqn
del env
```

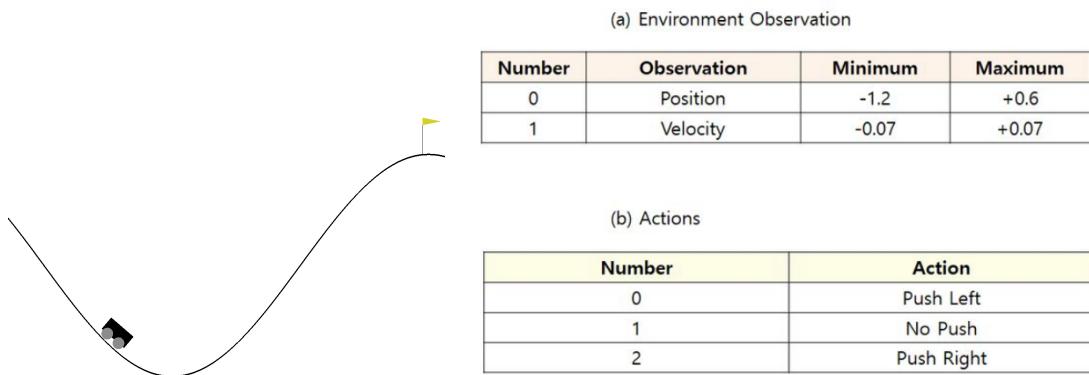
```
In [40]: env = gym.make('CartPole-v0')
actions = env.action_space.n
states = env.observation_space.shape[0]
model = build_model(states, actions)
dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])
```

```
In [41]: dqn.load_weights('dqn_weights.h5f')
```

```
In [42]: _ = dqn.test(env, nb_episodes=15, visualize=True)
```

Código:https://github.com/pedro048/EEC1505-REDES-NEURAIS/blob/main/CartPole-v0_DRL.ipynb

DRL - MountainCar-v0 (OpenAI gym)



```
In [1]: !pip install tensorflow==2.3.0  
!pip install gym  
!pip install keras  
!pip install keras-rl2  
!pip install pyglet|
```

Environment (MountainCar-v0) with OpenAI Gym

```
In [3]: import gym  
import random
```

```
In [4]: env = gym.make('MountainCar-v0')  
states = env.observation_space.shape[0]  
actions = env.action_space.n
```

```
In [5]: # Position  
# Velocity  
print('States:', states)
```

States: 2

```
In [6]: # Push left  
# Push right  
# No push  
print('actions:', actions)
```

actions: 3

Test Before Use Deep Reinforcement Learning (DRL)

```
In [7]: episodes = 15 # match number
for episode in range(1, episodes+1):
    state = env.reset()
    done = False
    score = 0

    while not done:
        env.render()
        action = random.choice([0,1])
        n_state, reward, done, info = env.step(action)
        score+=reward
    print('Episode:{} Score:{}'.format(episode, score))
```

Antes do treinamento com DRL: <https://youtu.be/lc4M7PLjFPo>

Deep Learning Model (MLP) with Keras

```
In [10]: import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
```

```
In [23]: # MLP => 3 hidden layers (1º = 128 neurons, 2º = 64 neurons, 3º = 32 neurons)
#           2 neurons for each state in the input layer
#           3 neurons for each action in the output layer
#           activation function for all the hidden layers = relu
def build_model(states, actions):
    model = Sequential()
    model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(env.action_space.n, activation='linear'))
    return model
```

```
In [24]: # MLP structure
model = build_model(states, actions)
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
<hr/>		
flatten_2 (Flatten)	(None, 2)	0
dense_8 (Dense)	(None, 128)	384
dense_9 (Dense)	(None, 64)	8256
dense_10 (Dense)	(None, 32)	2080
dense_11 (Dense)	(None, 3)	99
<hr/>		
Total params: 10,819		
Trainable params: 10,819		
Non-trainable params: 0		

Build Agent with Keras-DRL (Deep Reinforcement Learning)

```
In [25]: from rl.agents import DQNAgent
from rl.policy import BoltzmannQPolicy
from rl.memory import SequentialMemory

In [26]: def build_agent(model, actions):
    policy = BoltzmannQPolicy()
    memory = SequentialMemory(limit=50000, window_length=1)
    dqn = DQNAgent(model=model, memory=memory, policy=policy,
                   nb_actions=env.action_space.n, nb_steps_warmup=10, target_model_update=1e-2)
    return dqn
```

DRL Training

```
In [27]: dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])
dqn.fit(env, nb_steps=150000, visualize=False, verbose=2)
```

Durante o treinamento com DRL: <https://youtu.be/lzPJWmoUnvw>

DEL Test

```
In [40]: scores = dqn.test(env, nb_episodes=100, visualize=False)
print(np.mean(scores.history['episode_reward']))

In [42]: _ = dqn.test(env, nb_episodes=15, visualize=True)
```

Após o treinamento com DRL: <https://youtu.be/iddK0PvEoHY>

Reloading Agent from Memory (Saving Weights)

```
In [30]: dqn.save_weights('dqn_weights_1.h5f', overwrite=True)

In [31]: del model
del dqn
del env

In [32]: env = gym.make('MountainCar-v0')
actions = env.action_space.n
states = env.observation_space.shape[0]
model = build_model(states, actions)
dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])

In [34]: dqn.load_weights('dqn_weights_1.h5f')

In [35]: _ = dqn.test(env, nb_episodes=15, visualize=True)
```

Código:https://github.com/pedro048/EEC1505-REDES-NEURAIS/blob/main/MountainCar-v0_DRL.ipynb

Trabalho:

INTRODUÇÃO

As redes neurais artificiais (RNA) são um método de aprendizado de máquina criado a partir da ideia de modelar a forma de funcionamento do cérebro humano, simulando um conjunto de neurônios conectados, geralmente organizados em camadas, por onde são transmitidas as informações pela rede neural (HAYKIN et al., 2009).

Uma RNA é formada por múltiplos neurônios espalhados em camadas, nessas camadas os neurônios possuem parâmetros a serem estimados, chamados pesos sinápticos, são os pesos sinápticos que serão ajustados conforme as camadas extraem as características a partir das informações de entrada, podendo assim criar sua própria representação interna mais adequada do problema. Em um problema de classificação, essa representação é a que será usada como modelo para indicar as classes dos elementos da entrada. Devido muitos dos problemas do mundo real não serem linearmente separáveis, necessitamos de uma arquitetura com maior capacidade de representar essas informações, é aí que se insere o conceito de aprendizagem profunda.

Métodos de aprendizado profundo são métodos de aprendizagem com múltiplas camadas de representação, obtidos pela composição de módulos simples, mas não-lineares. A aprendizagem profunda permite modelos computacionais que compõem múltiplas camadas de processamento aprender representações de dados com múltiplos níveis de abstração. Esses métodos melhoraram drasticamente o estado da arte no reconhecimento de objetos visuais, detecção de objetos e muitos outros domínios (LECUN; BENGIO; HINTON, 2015).

TRANSFERÊNCIA DO CONHECIMENTO

Visando reduzir o tempo gasto treinando grandes modelos de aprendizado profundo do zero além de tentar suprir a falta de grandes conjuntos de dados para o treinamento desses modelos a solução encontrada foi a de reciclar o conhecimento de modelos previamente treinados com grande volume de dados (MENEGOLA et al., 2017). Essa reciclagem de conhecimento dos modelos é conhecida como transferência de conhecimento.

A transferência de conhecimento para a classificação de imagens consiste em usar os vetores de características gerados por uma rede neural profunda previamente treinada com seus pesos sinápticos configurados para reconhecer e extrair características de um determinado conjunto de dados para o qual foi previamente treinado, esses vetores gerados serão os descritores de cada imagem presente na base de dados. Após gerados esses vetores de características do novo conjunto do qual se deseja fazer a classificação, esses vetores podem ser utilizados como entrada para um novo classificador para que seja treinado com esses vetores ou podem ser dados como entrada para a camada completamente conectada de classificação da própria arquitetura que foi utilizada para extração das características (YOSINSKI et al., 2014).

Podemos exemplificar o processo de transferência de conhecimento como uma pessoa que está vendendo um objeto e começa a descrever os elementos presentes neste objeto para outra pessoa, onde esta segunda pessoa não está vendendo o objeto descrito, a partir das características ditas pela primeira pessoa o ouvinte irá traçar um perfil dos elementos que ele conhece e das características recebidas da primeira pessoa e associar a um objeto que ele conhece e que possua características semelhantes.

Como observado por (MENEGOLA et al., 2017) a transferência de conhecimento tem mostrado bons resultados e é uma alternativa muito vantajosa, pois dispensa o tempo que seria gasto treinando uma rede neural robusta do zero para um domínio específico.

A DATASET FOR BREAST CANCER HISTOPATHOLOGICAL IMAGE CLASSIFICATION

Em (SPANHOL et al., 2016), o autor nos apresenta a base BreaKHis, uma base de imagens histopatológicas geradas a partir da análise microscópica feita por patologistas de nódulos mamários. Tendo como objetivo mostrar a dificuldade do problema de classificação desse tipo de imagem os autores realizam testes sobre a base de dados utilizando diferentes algoritmos de extração de características de imagens e classificadores baseados em modelos de aprendizado de máquina.

Considerando duas principais formas de extrair características de imagens sendo uma delas a segmentação que é um processo que não é considerado trivial para imagens histopatológicas e características globais baseadas na textura, o autor optou por utilizar descritores baseados em textura sendo eles local binary patterns (LBP), completed LBP (CLBP), local phase quantization (LPQ), gray-level co-occurrence matrix (GLCM), parameter-free threshold adjacency statistics (PFTAS) e Oriented FAST and Rotated BRIEF (ORB) a Tabela 1 apresenta a quantidade de características geradas por cada um dos algoritmos listados:

RESUMO DOS DESCRIPTORES

Método	Quantidade de Características
CLBP	1352
GLCM	13
LBP	10
LPQ	256
ORB	32
PFTAS	162

Os algoritmos de extração de características das imagens foram testado com diferentes tipos de classificadores sendo eles 1-Nearest Neighbour (1-NN), quadratic linear analysis (QDA), support vector machine (SVM) e Random Forest of decision trees (RF). Dentre os classificadores utilizados o 1-NN foi utilizado com a finalidade de discriminar a relevância das características utilizadas, o SVM foi testado com vários kerneis, sendo o Gaussiano o que obteve melhores resultados.

Na etapa de teste (SPANHOL et al., 2016) dividiu o conjunto e 70% para treino e 30% para teste, a métrica utilizada para o resultado obtido consistiu na média do percentual de acertos de 5 testes realizados com os classificadores.

Ao apresentar os resultados obtidos o autor faz uma análise de como a magnitude das imagens estão influenciando nos testes foi observado que as diferentes magnitudes apresentadas na base de dados não carregam a mesma quantidade de informações relevantes para o aprendizado dos classificadores, no caso das imagens com magnitude 40x e 200x apresentaram bons resultados dos classificadores, sendo elas foco para outras investigações do trabalho.

Um outro ponto analisado pelo autor foi a influência dos descritores das imagens e como eles influenciaram nos resultados obtidos, no geral os resultados pareceram estáveis e com valores próximos, mas o descritor PFTAS teve a melhor performance entre os demais apesar de pouca diferença entre os resultados mesmo sabendo que esses resultados também possuem um pouco de influência dos classificadores utilizados.

Tendo feito essas observações (SPANHOL et al., 2016) explorou seus melhores resultados, que foi a combinação do classificador SVM e o descritor PFTAS no conjunto de magnitude 200x. Para entender melhor este resultado foi gerada uma matriz de confusão que teve como resposta a ocorrência de muitos falsos negativos que estavam ocorrendo por conta de imagens do conjunto benigno do subconjunto do tipo Fibroadenoma faziam com que o classificador não obtivesse uma melhor performance.

REFERÊNCIAS

- HAYKIN, S. S.; HAYKIN, S. S.; HAYKIN, S. S.; HAYKIN, S. S. Neural networks and learning machines. [S.I.]: Pearson Upper Saddle River, NJ, USA: 2009. v. 3.
- LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. Nature, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.
- MENEGOLA, A.; FORNACIALI, M.; PIRES, R.; BITTENCOURT, F. V.; AVILA, S.; VALLE, E. Knowledge transfer for melanoma screening with deep learning. arXiv preprint arXiv:1703.07479, 2017.
- YOSINSKI, J.; CLUNE, J.; BENGIO, Y.; LIPSON, H. How transferable are features in deep neural networks? In: Advances in neural information processing systems. [S.I.: s.n.], 2014. p. 3320–3328.
- SPANHOL, F. A.; OLIVEIRA, L. S.; PETITJEAN, C.; HEUTTE, L. A dataset for breast cancer histopathological image classification. IEEE Transactions on Biomedical Engineering, IEEE, v. 63, n. 7, p. 1455–1462, 2016.