[PSI3472-2023. Aula 8. Início.]

Classificação de séries temporais 1D

1. Introdução

Uma série temporal é uma coleção de observações feitas sequencialmente ao longo do tempo, onde a ordem dos dados é fundamental. Vamos estudar o exemplo de classificação de série temporal de tutorial de Keras, usando camadas convolucionais 1D:

https://keras.io/examples/timeseries/timeseries_classification_from_scratch/
http://www.timeseriesclassification.com/description.php?Dataset=FordA

Artigo correspondente:

https://arxiv.org/abs/1611.06455

2. Descrição do conjunto de dados

O conjunto de dados que iremos usar é *FordA*. Este conjunto contém 3.601 instâncias de treinamento e outras 1.320 instâncias de teste. Cada série temporal corresponde a uma medição do ruído do motor capturada por um sensor. Para esta tarefa, o objetivo é detectar automaticamente a presença de um problema específico no motor. O problema é uma tarefa de classificação binária balanceada.

Os arquivos de treino e teste estão no formato ".tsv" (semelhante a ".csv"). Cada linha do arquivo se inicia com -1 (sem defeito) ou +1 (com defeito), seguido por 500 números em ponto flutuante.

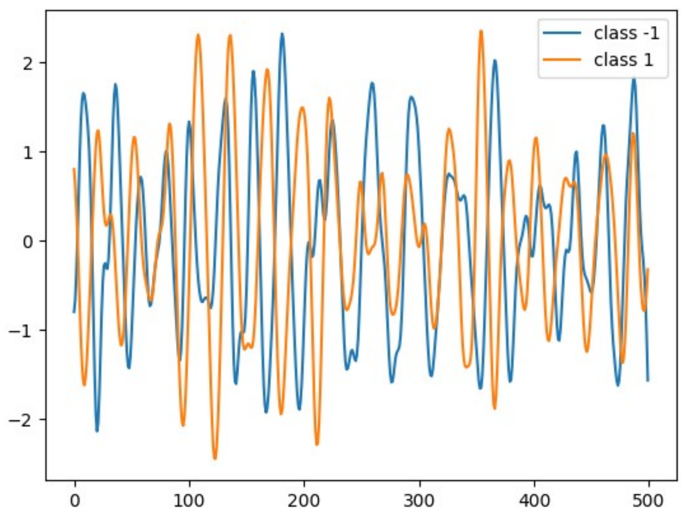


Figura 1: Exemplo de classe -1 (sem defeito) e classe +1 (com defeito).

Formato dos dados após leitura e reshape:

```
x_train (3601, 500, 1) float64
y_train (3601,) int64
x_test (1320, 500, 1) float64
y_test (1320,) int64
```

3. Rede neural convolucional 1D

```
# -*- coding: utf-8 -*-
#~/deep/keras/temporal/temporal2.py
# https://keras.io/examples/timeseries/timeseries_classification_from_scratch/
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import sys

def readucr(filename):
    data = np.loadtxt(filename, delimiter="\t")
    y = data[:, 0]; x = data[:, 1:]
    return x, y.astype(int)

root_url = "https://raw.githubusercontent.com/hfawaz/cd-diagram/master/FordA/"
x_train, y_train = readucr(root_url + "FordA_TRAIN.tsv")
x_test, y_test = readucr(root_url + "FordA_TEST.tsv")

classes = np.unique(np.concatenate((y_train, y_test), axis=0))

plt.figure()
for c in classes:
    c_x_train = x_train[y_train == c]
    plt.plot(c_x_train[0], label="class " + str(c))
plt.legend(loc="best"); plt.show()
plt.close()

x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))

num_classes = len(np.unique(y_train)) #2
idx = np.random.permutation(len(x_train))
x_train = x_train[idx]; y_train = y_train[idx]
y_train[y_train == -1] = 0; y_test[y_test == -1] = 0

def make_model(input_shape):
    input_layer = keras.layers.Input(input_shape) # (None, 500, 1)

    conv1 = keras.layers.Conv1D(filters=64, kernel_size=3, padding="same")(input_layer) # (None, 500, 3)
    conv1 = keras.layers.BatchNormalization()(conv1)
    conv1 = keras.layers.ReLU()(conv1) # (None, 500, 3)

    conv2 = keras.layers.Conv1D(filters=64, kernel_size=3, padding="same")(conv1)
    conv2 = keras.layers.BatchNormalization()(conv2)
    conv2 = keras.layers.ReLU()(conv2)

    conv3 = keras.layers.Conv1D(filters=64, kernel_size=3, padding="same")(conv2)
    conv3 = keras.layers.BatchNormalization()(conv3)
    conv3 = keras.layers.ReLU()(conv3)  # (None, 500, 64)

    gap = keras.layers.GlobalAveragePooling1D()(conv3) # (None, 64) - verificar
    output_layer = keras.layers.Dense(num_classes, activation="softmax")(gap)
    return keras.models.Model(inputs=input_layer, outputs=output_layer)

model = make_model(input_shape=x_train.shape[1:])
keras.utils.plot_model(model, to_file="temporal1.png", show_shapes=True)
model.summary()

epochs = 500; batch_size = 32
callbacks = [
    keras.callbacks.ModelCheckpoint(
        "best_model.h5", save_best_only=True, monitor="val_loss"
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor="val_loss", factor=0.5, patience=20, min_lr=0.0001
    ),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=50, verbose=1),
]
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["sparse_categorical_accuracy"],
)
history = model.fit(x_train, y_train, batch_size=batch_size,
    epochs=epochs, callbacks=callbacks, validation_split=0.2, verbose=2
)

model = keras.models.load_model("best_model.h5")
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy", test_acc); print("Test loss", test_loss)

metric = "sparse_categorical_accuracy"
plt.figure()
plt.plot(history.history[metric]); plt.plot(history.history["val_" + metric])
plt.title("model " + metric)
plt.ylabel(metric, fontsize="large"); plt.xlabel("epoch", fontsize="large")
plt.legend(["train", "val"], loc="best")
plt.show(); plt.close()
```

Programa 1 (temporal2): Classifica FordA usando redes convolucionais 1D.
https://colab.research.google.com/drive/1d-1zOwsX09D2gRKOBEjqem4Eqj1fldf3

Atinge, após 306 épocas (parou pelo callback EarlyStopping):

Tabela 1: Desempenho de CNN 1D para classificar FordA.

Acuracidade de treino	0.9851
Acuracidade de validação	0.9639
Acuracidade de teste	0.9720

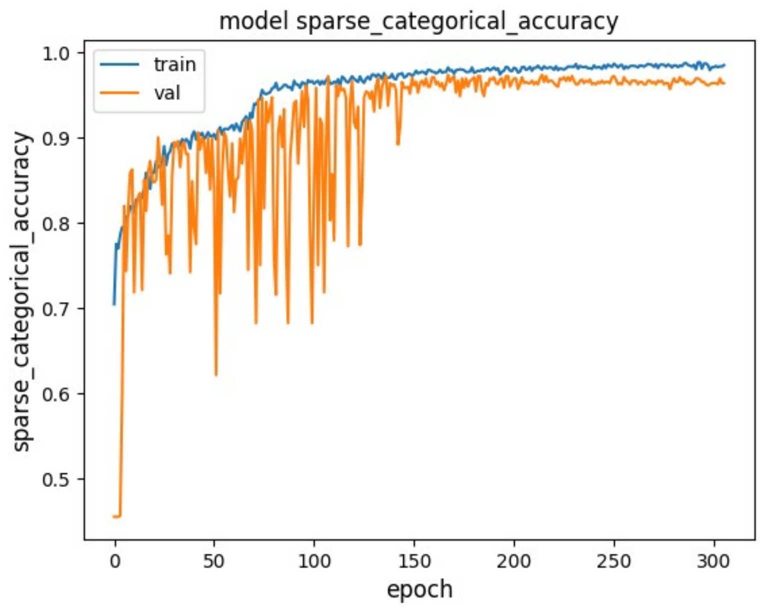
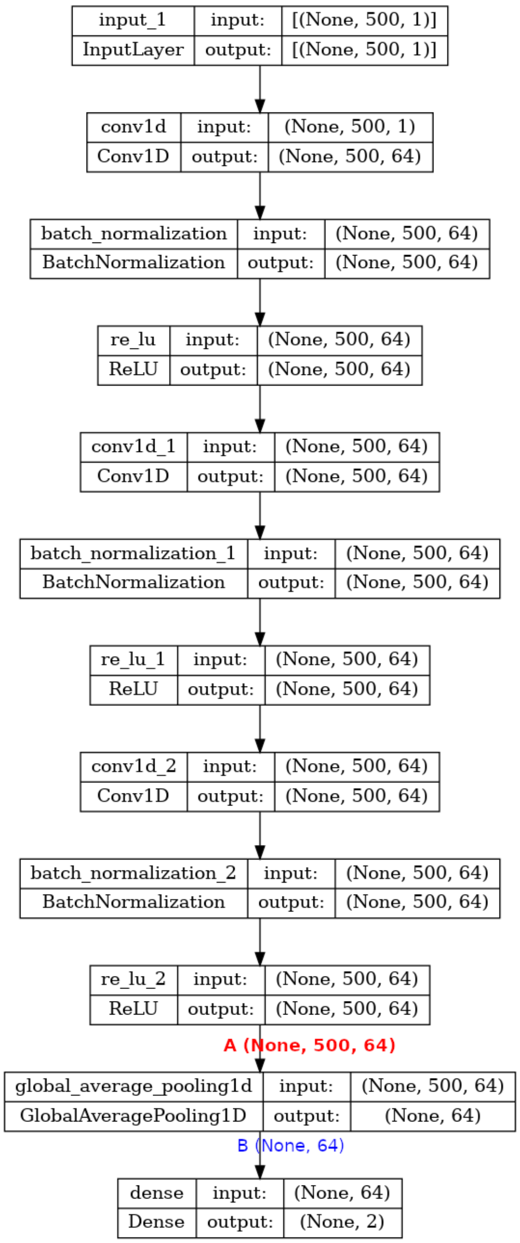


Figura 2: Histórico de treino do modelo CNN 1D.

Algumas pontos interessantes deste programa Keras (que seria bom usar também em outros programas):

- Linhas 60-62: Salva o melhor modelo monitorando “validation loss”.
- Linhas 63-65: Reduz learning rate por 0.5 quando “validation loss” para de melhorar por 20 épocas.
- Linha 66: Termina o programa antecipadamente se “validation loss” não melhorar durante 50 épocas.
- Linhas 70-71: O uso de “sparse categorical crossentropy” e “sparse categorical accuracy” evita ter que calcular explicitamente one-hot-encoding.
- Linha 74: validation_split=0.2 reserva 20% dos dados de teste para validação.
- Linha 77: Carrega o melhor modelo (obtido monitorando “validation loss”) do disco para avaliar o desempenho do modelo nos dados de teste. A rede que está na memória possivelmente não é a melhor.




```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500, 1)]          0

 conv1d (Conv1D)             (None, 500, 64)           256

 batch_normalization (BatchN  (None, 500, 64)          256
 ormalization)

 re_lu (ReLU)                (None, 500, 64)           0

 conv1d_1 (Conv1D)           (None, 500, 64)           12352

 batch_normalization_1 (Batc  (None, 500, 64)          256
 hNormalization)

 re_lu_1 (ReLU)              (None, 500, 64)           0

 conv1d_2 (Conv1D)           (None, 500, 64)           12352

 batch_normalization_2 (Batc  (None, 500, 64)          256
 hNormalization)

 re_lu_2 (ReLU)              (None, 500, 64)           0

 global_average_pooling1d (G  (None, 64)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 2)                 130

=================================================================
Total params: 25,858
Trainable params: 25,474
Non-trainable params: 384
_________________________________________________________________
```

Figura 3: Arquitetura da rede com convoluções 1-D.

O sinal de entrada tem 500 pontos amostrais coletados em 500 instantes diferentes (dtype=float_64). Este sinal é representado por um tensor (500,1) – o número de colunas 1 indica que só tem um único sensor.

Este sinal (500,1) passa por uma camada com 64 convoluções de kernel 3, resultando em mapas de atributos (500, 64). Depois, há camadas de batch normalization e relu.

Seguem mais 2 conjuntos de camadas convolução, batch normalization e relu, resultando em (500, 64) atributos (ponto A vermelho da figura 3).

Global average pooling tira média no tempo dos 64 atributos, ficando com vetor de 64 elementos (ponto B azul da figura 3). Veja a figura 4. Pense por que é razoável tirar as médias no tempo dos 64 atributos mas não é razoável tirar média entre os 64 atributos.

Por fim, uma camada densa com 2 saídas termina a classificação. A figura 4 ilustra o funcionamento de global average pooling (entre os pontos A e B da figura 3).

Este programa chegou a acuracidade de teste de 96,97%.

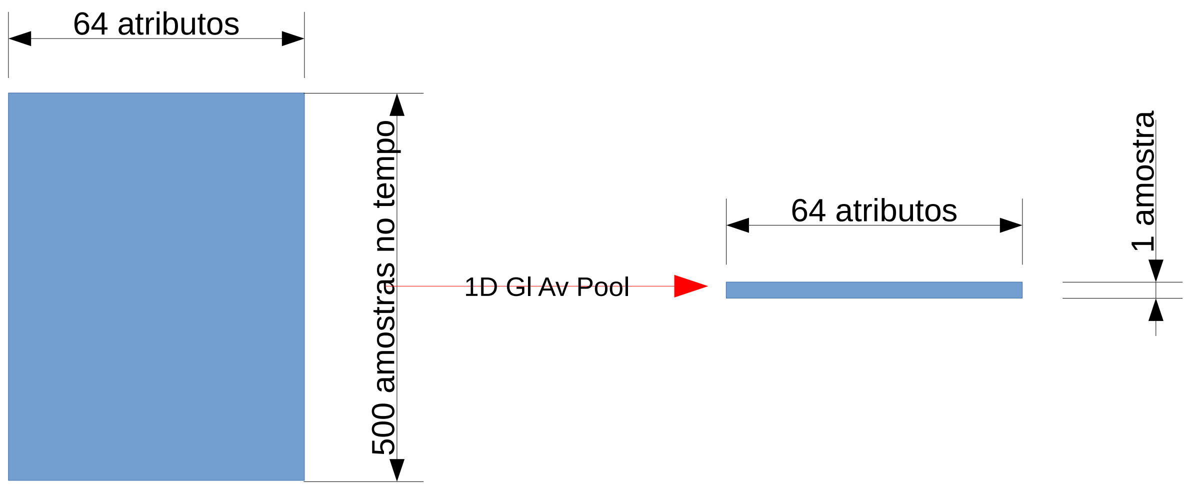


Figura 4: Uso de 1-D global average pooling na CNN 1D.

[PSI3472-2023. Aula 8. Fim.]

4. Rede neural recorrente para classificar série temporal

Nota: A parte teórica das redes neurais recorrentes (RNN - *SimpleRNN*, *LSTM* e *Bidirectional*) está na apostila *NLP*. Leia essa apostila antes de prosseguir.

4.1 RNN6 - LSTM bidirecional

Vamos classificar FordA usando LSTM bidirecional. Este modelo foi inspirado modelo do blog abaixo, tornando a primeira camada LSTM bidirecional e removendo uma das camadas LSTM.

https://medium.com/@prajjwalchauhan94017/stock-prediction-and-forecasting-using-lstm-long-short-term-memory-9ff56625de73

```python
# -*- coding: utf-8 -*-
# ~/deep/keras/temporal/rnn6.py
from tensorflow import keras
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM, SimpleRNN, Bidirectional
import numpy as np
import matplotlib.pyplot as plt
import sys

def readucr(filename):
    data = np.loadtxt(filename, delimiter="\t")
    y = data[:, 0]; x = data[:, 1:]
    return x, y.astype(int)

root_url = "https://raw.githubusercontent.com/hfawaz/cd-diagram/master/FordA/"
x_train, y_train = readucr(root_url + "FordA_TRAIN.tsv")
x_test, y_test = readucr(root_url + "FordA_TEST.tsv")
classes = np.unique(np.concatenate((y_train, y_test), axis=0))

x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))

num_classes = len(np.unique(y_train)) #2
idx = np.random.permutation(len(x_train))
x_train = x_train[idx]; y_train = y_train[idx]
y_train[y_train == -1] = 0; y_test[y_test == -1] = 0

def make_model(input_shape):
    input_layer = keras.layers.Input((x_train.shape[1],1)) # (None, 500, 1)
    lstm1 = Bidirectional( LSTM(80,return_sequences = True) )(input_layer)
    #lstm2 = LSTM(50,return_sequences = True)(lstm1)
    lstm3 = LSTM(80)(lstm1)
    output_layer = keras.layers.Dense(num_classes, activation="softmax")(lstm3)
    return keras.models.Model(inputs=input_layer, outputs=output_layer)

model = make_model(input_shape=x_train.shape[1:])
keras.utils.plot_model(model, to_file="rnn6.png", show_shapes=True)
model.summary()

epochs = 500; batch_size = 32
callbacks = [
    keras.callbacks.ModelCheckpoint(
        "rnn6_best_model.h5", save_best_only=True, monitor="val_loss"
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor="val_loss", factor=0.5, patience=20, min_lr=0.0001
    ),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=50, verbose=1),
]
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["sparse_categorical_accuracy"],
)
history = model.fit(x_train, y_train, batch_size=batch_size,
    epochs=epochs, callbacks=callbacks, validation_split=0.2, verbose=2
)

model = keras.models.load_model("rnn6_best_model.h5")
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy", test_acc); print("Test loss", test_loss)

metric = "sparse_categorical_accuracy"
plt.figure()
plt.plot(history.history[metric]); plt.plot(history.history["val_" + metric])
plt.title("model " + metric)
plt.ylabel(metric, fontsize="large"); plt.xlabel("epoch", fontsize="large")
plt.legend(["train", "val"], loc="best")
plt.savefig("Figure_rnn6.png")
plt.show(); plt.close()
```

Programa 2 (rnn6): Classifica FordA usando LSTM bidirecional.

Saída:

```
Epoch 187/500
90/90 - 5s - loss: 0.0864 - sparse_categorical_accuracy: 0.9726 - val_loss: 0.2216 - val_sparse_categorical_accuracy: 0.9293 - lr: 1.0000e-04 - 5s/epoch - 55ms/step
Epoch 188/500
90/90 - 5s - loss: 0.0519 - sparse_categorical_accuracy: 0.9882 - val_loss: 0.2557 - val_sparse_categorical_accuracy: 0.9293 - lr: 1.0000e-04 - 5s/epoch - 56ms/step
Epoch 188: early stopping
42/42 [==============================] - 2s 25ms/step - loss: 0.1761 - sparse_categorical_accuracy: 0.9348
Test accuracy 0.9348484873771667
Test loss 0.17611220479011536
```

Após 187 épocas (parou pelo callback EarlyStopping), atinge acuracidade de teste de 93,5%. Isto é bem pior do que rede convolucional que atingiu a acuracidade de teste de 97,2%.

Tabela 2: Comparação de CNN com LSTM para classificar FordA.

	temporal2 (CNN) programa da seção anterior	RNN6 (LSTM) programa desta seção
Acuracidade de treino	0.9851	0.9882
Acuracidade de validação	**0.9639**	0.9293
Acuracidade de teste	**0.9720**	0.9348

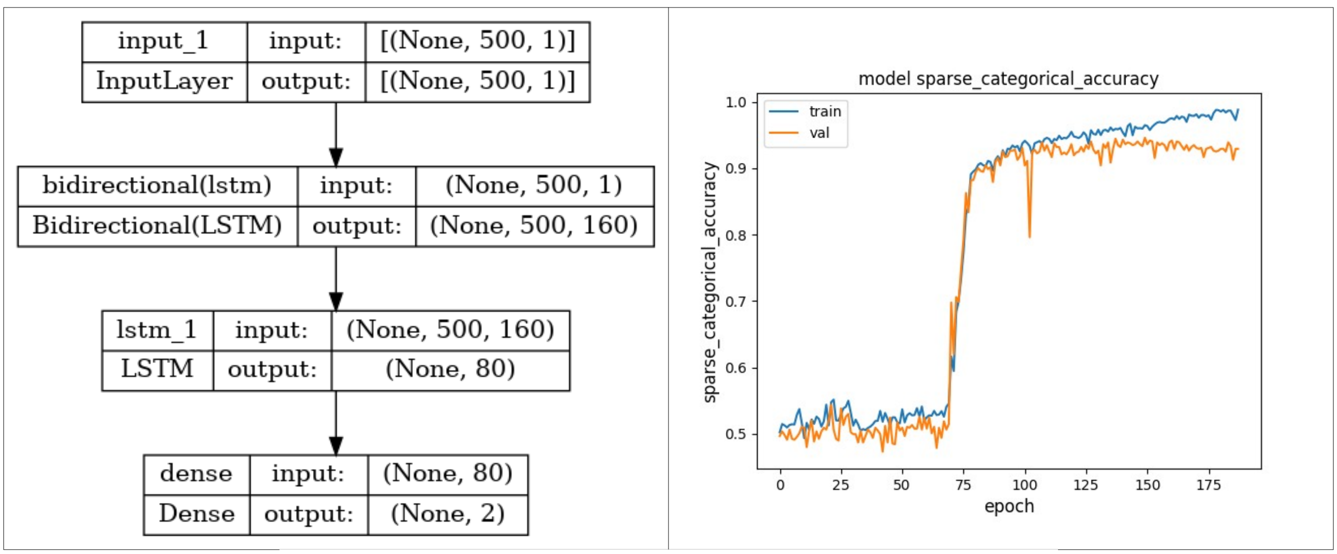


Figura 5: Arquitetura e histórico de treino do modelo LSTM.

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500, 1)]          0
 bidirectional (Bidirectiona  (None, 500, 160)         52480
 l)
 lstm_1 (LSTM)               (None, 80)                77120
 dense (Dense)               (None, 2)                 162
=================================================================
Total params: 129,762
Trainable params: 129,762
Non-trainable params: 0
```

4.2 RNN7 – Trocar LSTM por SimpleRNN

Vamos trocar camadas LSTM do programa anterior (RNN6) por SimpleRNN.

```
# -*- coding: utf-8 -*-
# ~/deep/keras/temporal/rnn7.py
from tensorflow import keras
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM, SimpleRNN, Bidirectional
import numpy as np
import matplotlib.pyplot as plt
import sys

def readucr(filename):
    data = np.loadtxt(filename, delimiter="\t")
    y = data[:, 0]; x = data[:, 1:]
    return x, y.astype(int)

root_url = "https://raw.githubusercontent.com/hfawaz/cd-diagram/master/FordA/"
x_train, y_train = readucr(root_url + "FordA_TRAIN.tsv")
x_test, y_test = readucr(root_url + "FordA_TEST.tsv")

classes = np.unique(np.concatenate((y_train, y_test), axis=0))

x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))

num_classes = len(np.unique(y_train)) #2
idx = np.random.permutation(len(x_train))
x_train = x_train[idx]; y_train = y_train[idx]
y_train[y_train == -1] = 0; y_test[y_test == -1] = 0

def make_model(input_shape):
    input_layer = keras.layers.Input((x_train.shape[1],1)) # (None, 500, 1)
    lstm1 = Bidirectional( SimpleRNN(80,return_sequences = True) )(input_layer)
    #lstm2 = LSTM(50,return_sequences = True)(lstm1)
    lstm3 = SimpleRNN(80)(lstm1)
    output_layer = keras.layers.Dense(num_classes, activation="softmax")(lstm3)
    return keras.models.Model(inputs=input_layer, outputs=output_layer)

model = make_model(input_shape=x_train.shape[1:])
keras.utils.plot_model(model, to_file="rnn7.png", show_shapes=True)
model.summary()

epochs = 500; batch_size = 32
callbacks = [
    keras.callbacks.ModelCheckpoint(
        "rnn7_best_model.h5", save_best_only=True, monitor="val_loss"
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor="val_loss", factor=0.5, patience=20, min_lr=0.0001
    ),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=50, verbose=1),
]
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["sparse_categorical_accuracy"],
)
history = model.fit(x_train, y_train, batch_size=batch_size,
    epochs=epochs, callbacks=callbacks, validation_split=0.2, verbose=2
)

model = keras.models.load_model("rnn7_best_model.h5")
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy", test_acc); print("Test loss", test_loss)

metric = "sparse_categorical_accuracy"
plt.figure()
plt.plot(history.history[metric]); plt.plot(history.history["val_" + metric])
plt.title("model " + metric)
plt.ylabel(metric, fontsize="large"); plt.xlabel("epoch", fontsize="large")
plt.legend(["train", "val"], loc="best")
plt.savefig("Figure_rnn7.png")
plt.show(); plt.close()
```

Programa 3 (rnn7): Programa para classificar FordA usando SimpleRNN bidirecional.

Saída:

```
Epoch 52/500
90/90 - 46s - loss: 0.6903 - sparse_categorical_accuracy: 0.5295 - val_loss: 0.6969 - val_sparse_categorical_accuracy: 0.4910 - lr: 2.5000e-04 - 46s/epoch - 508ms/step
Epoch 53/500
90/90 - 44s - loss: 0.6909 - sparse_categorical_accuracy: 0.5281 - val_loss: 0.6974 - val_sparse_categorical_accuracy: 0.4840 - lr: 2.5000e-04 - 44s/epoch - 485ms/step
Epoch 53: early stopping
42/42 [==============================] - 4s 97ms/step - loss: 0.6667 - sparse_categorical_accuracy: 0.5409
Test accuracy 0.5409091114997864
Test loss 0.666744589805603
```

O modelo não converge, provavelmente porque RNN simples não consegue associar bem eventos distantes no tempo.

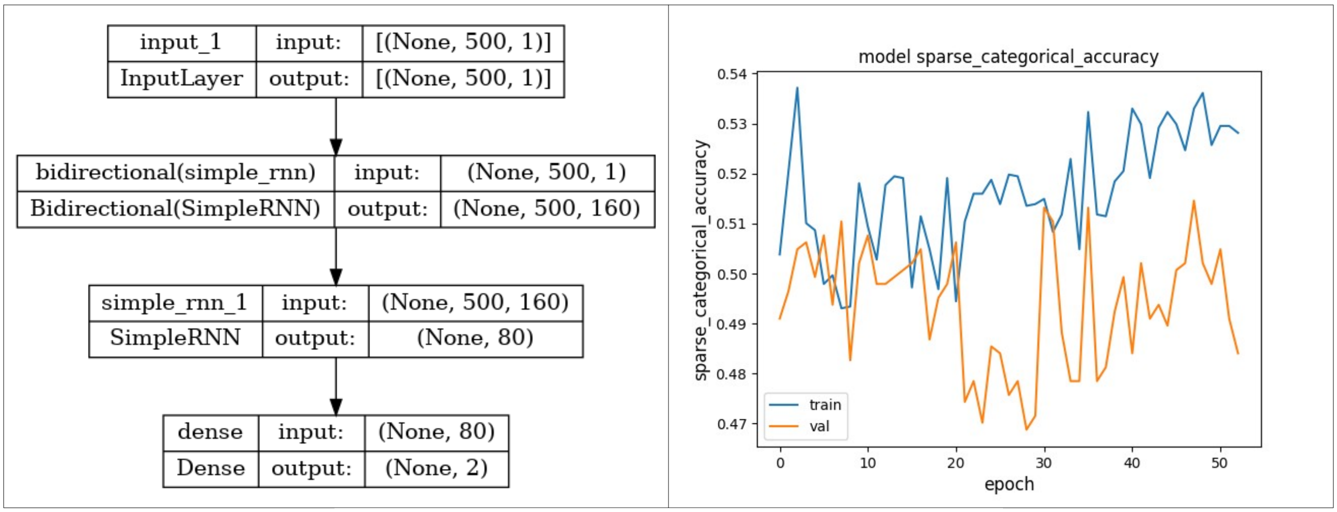


Figura 6: Usando SimpleRNN, a rede não converge.

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500, 1)]          0
 bidirectional (Bidirectiona  (None, 500, 160)         13120
 l)
 simple_rnn_1 (SimpleRNN)    (None, 80)                19280
 dense (Dense)               (None, 2)                 162
=================================================================
Total params: 32,562
Trainable params: 32,562
Non-trainable params: 0
```

5. Transformer para classificar série temporal

O seguinte tutorial de Keras mostra como classificar FordA usando Transformer.

https://keras.io/examples/timeseries/timeseries_classification_transformer/

```
# https://colab.research.google.com/drive/170PiIdRbTnNhhG_VTYTOELRX3zXJsPCC#scrollTo=G8wqHj1IqeLU
# ~/deep/keras/temporal/transf2.py
# Este programa nao roda no meu computador por falta de memoria
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

def readucr(filename):
    data = np.loadtxt(filename, delimiter="\t")
    y = data[:, 0]; x = data[:, 1:]
    return x, y.astype(int)

root_url = "https://raw.githubusercontent.com/hfawaz/cd-diagram/master/FordA/"
x_train, y_train = readucr(root_url + "FordA_TRAIN.tsv")
x_test, y_test = readucr(root_url + "FordA_TEST.tsv")
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))
n_classes = len(np.unique(y_train))

idx = np.random.permutation(len(x_train))
x_train = x_train[idx]; y_train = y_train[idx]
y_train[y_train == -1] = 0; y_test[y_test == -1] = 0

def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    x = layers.LayerNormalization(epsilon=1e-6)(inputs)
    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = layers.Dropout(dropout)(x)
    res = x + inputs

    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation="relu")(x)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res

def build_model(input_shape, head_size, num_heads, ff_dim,
    num_transformer_blocks, mlp_units, dropout=0, mlp_dropout=0):
    inputs = keras.Input(shape=input_shape)
    x = inputs
    for _ in range(num_transformer_blocks):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)
    x = layers.GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x = layers.Dense(dim, activation="relu")(x)
        x = layers.Dropout(mlp_dropout)(x)
    outputs = layers.Dense(n_classes, activation="softmax")(x)
    return keras.Model(inputs, outputs)

input_shape = x_train.shape[1:]

model = build_model(input_shape, head_size=256, num_heads=4, ff_dim=4,
    num_transformer_blocks=4, mlp_units=[128], mlp_dropout=0.4, dropout=0.25)

model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=1e-4),
    metrics=["sparse_categorical_accuracy"],
)
keras.utils.plot_model(model, to_file="transf2.png", show_shapes=True)
model.summary()

callbacks = [keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)]

history = model.fit(x_train, y_train, validation_split=0.2, epochs=200,
    batch_size=64, callbacks=callbacks
)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=1)
print("Test accuracy", test_acc); print("Test loss", test_loss)

metric = "sparse_categorical_accuracy"
plt.figure()
plt.plot(history.history[metric]); plt.plot(history.history["val_" + metric])
plt.title("model " + metric)
plt.ylabel(metric, fontsize="large"); plt.xlabel("epoch", fontsize="large")
plt.legend(["train", "val"], loc="best")
plt.savefig("Figure_rnn6.png")
plt.show(); plt.close()
```

Programa 4 (transf2): Classificar FordA usando Transformer.

https://colab.research.google.com/drive/170PiIdRbTnNhhG_VTYTOELRX3zXJsPCC#scrollTo=G8wqHj1IqeLU

Saída:

```
Epoch 116/200
45/45 [==============================] - 24s 545ms/step - loss: 0.1610 - sparse_categorical_accuracy:
0.9462 - val_loss: 0.3743 - val_sparse_categorical_accuracy: 0.8377
42/42 [==============================] - 4s 85ms/step - loss: 0.3488 - sparse_categorical_accuracy:
0.8417
Test accuracy 0.8416666388511658 Test loss 0.3487994074821472
```

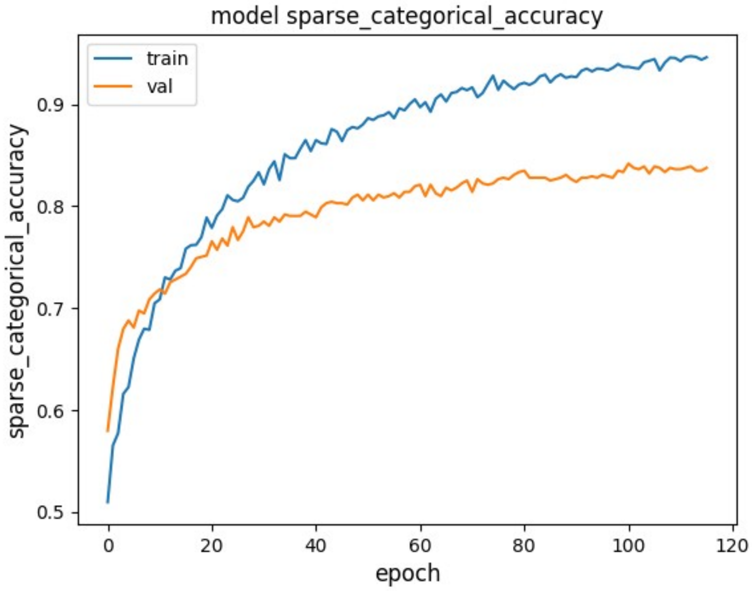


Figura 7: Histórico de treino da rede Transformer.

Após 116 épocas (parou pelo callback EarlyStopping), atinge acuracidade de teste de 84,2%. Isto é bem pior do que rede convolucional que atingiu acuracidade de 97,2% ou LSTM bidirecional que atingiu acuracidade de 93,5%. Parece que Transformer necessita de um conjunto de treino grande para que funcione bem.

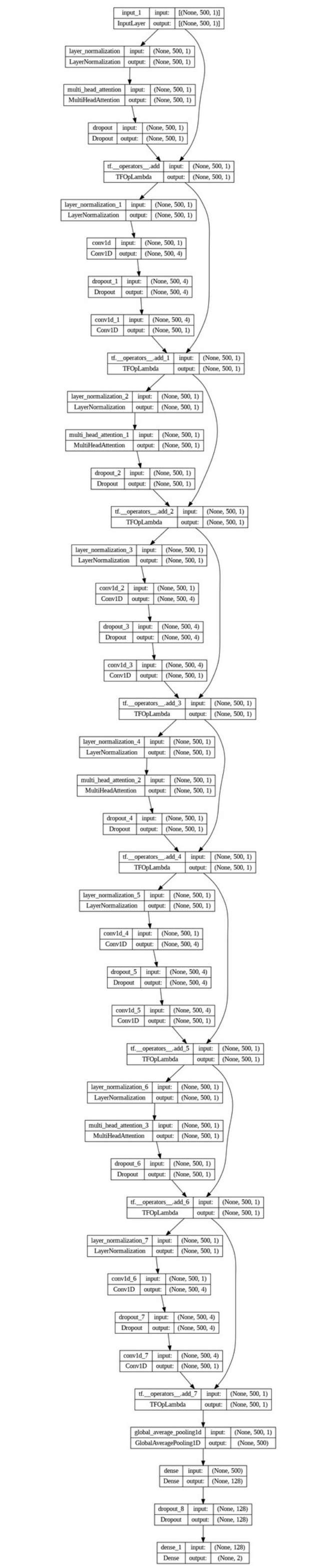
Um ponto interessante deste programa é:

```
callbacks = [keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)]
```

Onde o melhor modelo é restaurado sem precisar armazenar o melhor modelo no disco.

Tabela 3: Comparação de CNN, LSTM e Transformer para classificar FordA.

	temporal2 (CNN) programa anterior	RNN6 (LSTM) programa anterior	Transf2 (Transformer) programa desta seção
Acuracidade de treino	0.9851	0.9882	0.9462
Acuracidade de validação	**0.9639**	0.9293	0.8377
Acuracidade de teste	**0.9720**	0.9348	0.8417



```
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 500, 1)]     0           []
 layer_normalization (LayerNorm  (None, 500, 1)      2           ['input_1[0][0]']
 alization)
 multi_head_attention (MultiHea  (None, 500, 1)      7169        ['layer_normalization[0][0]',
 dAttention)                                                      'layer_normalization[0][0]']
 dropout (Dropout)              (None, 500, 1)       0           ['multi_head_attention[0][0]']
 tf.__operators__.add (TFOpLamb  (None, 500, 1)      0           ['dropout[0][0]',
 da)                                                              'input_1[0][0]']
 layer_normalization_1 (LayerNo  (None, 500, 1)      2           ['tf.__operators__.add[0][0]']
 rmalization)
 conv1d (Conv1D)                (None, 500, 4)       8           ['layer_normalization_1[0][0]']
 dropout_1 (Dropout)            (None, 500, 4)       0           ['conv1d[0][0]']
 conv1d_1 (Conv1D)              (None, 500, 1)       5           ['dropout_1[0][0]']
 tf.__operators__.add_1 (TFOpLa  (None, 500, 1)      0           ['conv1d_1[0][0]',
 mbda)                                                            'tf.__operators__.add[0][0]']
 layer_normalization_2 (LayerNo  (None, 500, 1)      2           ['tf.__operators__.add_1[0][0]']
 rmalization)
 multi_head_attention_1 (MultiH  (None, 500, 1)      7169        ['layer_normalization_2[0][0]',
 eadAttention)                                                    'layer_normalization_2[0][0]']
 dropout_2 (Dropout)            (None, 500, 1)       0           ['multi_head_attention_1[0][0]']
 tf.__operators__.add_2 (TFOpLa  (None, 500, 1)      0           ['dropout_2[0][0]',
 mbda)                                                            'tf.__operators__.add_1[0][0]']
 layer_normalization_3 (LayerNo  (None, 500, 1)      2           ['tf.__operators__.add_2[0][0]']
 rmalization)
 conv1d_2 (Conv1D)              (None, 500, 4)       8           ['layer_normalization_3[0][0]']
 dropout_3 (Dropout)            (None, 500, 4)       0           ['conv1d_2[0][0]']
 conv1d_3 (Conv1D)              (None, 500, 1)       5           ['dropout_3[0][0]']
 tf.__operators__.add_3 (TFOpLa  (None, 500, 1)      0           ['conv1d_3[0][0]',
 mbda)                                                            'tf.__operators__.add_2[0][0]']
 layer_normalization_4 (LayerNo  (None, 500, 1)      2           ['tf.__operators__.add_3[0][0]']
 rmalization)
 multi_head_attention_2 (MultiH  (None, 500, 1)      7169        ['layer_normalization_4[0][0]',
 eadAttention)                                                    'layer_normalization_4[0][0]']
 dropout_4 (Dropout)            (None, 500, 1)       0           ['multi_head_attention_2[0][0]']
 tf.__operators__.add_4 (TFOpLa  (None, 500, 1)      0           ['dropout_4[0][0]',
 mbda)                                                            'tf.__operators__.add_3[0][0]']
 layer_normalization_5 (LayerNo  (None, 500, 1)      2           ['tf.__operators__.add_4[0][0]']
 rmalization)
 conv1d_4 (Conv1D)              (None, 500, 4)       8           ['layer_normalization_5[0][0]']
 dropout_5 (Dropout)            (None, 500, 4)       0           ['conv1d_4[0][0]']
 conv1d_5 (Conv1D)              (None, 500, 1)       5           ['dropout_5[0][0]']
 tf.__operators__.add_5 (TFOpLa  (None, 500, 1)      0           ['conv1d_5[0][0]',
 mbda)                                                            'tf.__operators__.add_4[0][0]']
 layer_normalization_6 (LayerNo  (None, 500, 1)      2           ['tf.__operators__.add_5[0][0]']
 rmalization)
 multi_head_attention_3 (MultiH  (None, 500, 1)      7169        ['layer_normalization_6[0][0]',
 eadAttention)                                                    'layer_normalization_6[0][0]']
 dropout_6 (Dropout)            (None, 500, 1)       0           ['multi_head_attention_3[0][0]']
 tf.__operators__.add_6 (TFOpLa  (None, 500, 1)      0           ['dropout_6[0][0]',
 mbda)                                                            'tf.__operators__.add_5[0][0]']
 layer_normalization_7 (LayerNo  (None, 500, 1)      2           ['tf.__operators__.add_6[0][0]']
 rmalization)
 conv1d_6 (Conv1D)              (None, 500, 4)       8           ['layer_normalization_7[0][0]']
 dropout_7 (Dropout)            (None, 500, 4)       0           ['conv1d_6[0][0]']
 conv1d_7 (Conv1D)              (None, 500, 1)       5           ['dropout_7[0][0]']
 tf.__operators__.add_7 (TFOpLa  (None, 500, 1)      0           ['conv1d_7[0][0]',
 mbda)                                                            'tf.__operators__.add_6[0][0]']
 global_average_pooling1d (Glob  (None, 500)         0           ['tf.__operators__.add_7[0][0]']
 alAveragePooling1D)
 dense (Dense)                  (None, 128)          64128       ['global_average_pooling1d[0][0]'
                                                                  ]
 dropout_8 (Dropout)            (None, 128)          0           ['dense[0][0]']
 dense_1 (Dense)                (None, 2)            258         ['dropout_8[0][0]']
==================================================================================================
Total params: 93,130
Trainable params: 93,130
Non-trainable params: 0
```

Figura 8: Arquitetura da rede Transformer para classificar FordA.