MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo VI - Redes neurais para dados sequenciais

Exercícios (com soluções)

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

Recomenda-se fortemente que os exercícios sejam feitos sem consultar as respostas antecipadamente.

Exercício 1)

Considerando o treinamento de uma rede neural em que é utilizado um exemplo por vez para adaptar os pesos, qual das alternativas abaixo indica a diferença de uma unidade recorrente para uma não recorrente?

(a) Unidades recorrentes consideram que há dependência entre exemplos de cada iteração sucessiva e por isso mantêm uma memória relacionada unicamente ao exemplo imediamente anterior ao atual
(b) Unidades recorrentes possuem uma quantidade relativamente menor de parâmetros a serem computados quando comparada às não recorrentes
(c) Unidades recorrentes consideram que cada exemplo é independente dos demais e portanto apenas o exemplo atual é considerado na atualização de seus pesos
(d) Unidades recorrentes possuem uma memória interna que com base nos exemplos de iterações anteriores e que influencia na saída da iteração atual.

Justificativa: a memória ou estado interno contém informação de exemplos vistos anteriormente, não apenas do imediatamente anterior. Essa memória resulta na necessidade de aprender mais parâmetros para controlar a combinação entre os dados atuais e os vistos anteriormente.

Exercício 2)

Quais dos cenários abaixo não necessitam de redes recorrentes ou que consideram a dependência sequencial dos dados?

(a) Tradução de sentenças entre dois idiomas
(b) Descrição automática de cenas em um clipe de vídeo
(c) Estimar o valor de um imóvel colocado a venda, com base em suas características

(d) Predição de valor futuro de uma série temporal relacionada ao preço de uma ação na bolsa de valores

Justificativa: Das alternativas, a regressão para estimar o valor de um imóvel é a única que não necessita da dependencia sequencial dos dados pois esse tipo de problema tem características estáticas. Sendo um imóvel colocado a venda, não há dados anteriores que possam ajudar imediatamente a melhor modelar o problema

---

Exercício 3)

Qual a principal diferença entre o "output gate" da LSTM e o "update gate" da GRU?

(a) O update gate (GRU) filtra qual parte do sumário anterior será mantida e qual será descartada, e o output gate (LSTM) filtra qual parte do estado de célula anterior será mantida e qual será descartada
(b) O update gate (GRU) realiza uma combinação entre o sumário atual e o anterior enquanto o output gate (LSTM) faz uma combinação entre o estado de célula atual e o anterior
(c) O update gate (GRU) realiza uma combinação entre o sumário atual e o anterior enquanto o output gate (LSTM) utiliza do estado de célula para ponderar uma combinação entre a entrada atual e o sumário anterior
(d) O update gate (GRU) realiza uma combinação entre a entrada atual e o sumário anterior o sumário atual eenquanto o output gate (LSTM) faz uma combinação entre o estado de célula atual e o anterior

Justificativa: O output gate realiza a operação $h_t = f(W_o h_{t-1} + W_o x_t + b_o) * tanh(C_t)$ e portanto utiliza o estado de célula C_t para poderar a entrada atual x_t e o sumário anterior h_{t-1}, já o update gate da GRU não mantêm um estado de célula, apenas combinando um sumário candidato atual $\tilde{h}_t$ e o anterior h_{t-1}.

Exercício 4)

Representa uma opção menos adequada para o projeto de redes neurais aplicadas a dados sequenciais:

(a) O uso de convoluções para capturar o posicionamento local de dados sequenciais
(b) O uso de unidades densas para melhor aprender a relação entre dados correlacionados sequencialmente
(c) O uso de unidades do tipo recorrente para capturar dependências na ordem em que os exemplos são vistos pela rede neural
(d) O uso de mecanismos de atenção para identificar relações entre representações sequenciais

Justificativa: as unidades densas podem aprender relações entre dados sequenciais, mas é a menos adequada por não possuir um mecanismo explícito para capturar dependências,

em particular de mais longo prazo

Exercício 5)

O mecanismo de atenção no contexto de dados sequenciais é implementado para:

(a) Capturar a importância de características vistas anteriormente com relação a uma características do exemplo atual
(b) Simular um tipo de recorrência no aspecto espacial dos dados
(c) Ponderar as características de um exemplo de entrada atual com relação a exemplos futuros
(d) Realizar seleção de características numa base de dados difícil

Justificativa: a atenção é computada por meio do alinhamento de características vistas anteriormente com características de um exemplo atual. Ainda que isso substitua a recorrência não há um mecanismo explícito de recorrência, não é possível ponderar com relação a dados futuros, e não está relacionada a seleção de características.

Exercício 6)

Nesse exercício vamos juntar 4 dos modelos que já viram até agora e comparar num problema de regressão de séries temporais:

Carregue a base de dados starbucks.csv, com uma divisão hold-out utilizando os 80% exemplos iniciais para treinamento e os restantes para teste e normalize no intervalo 0-1 conforme visto em sala de aula:

Considerando 4 redes:

- densa: 2 camada densas com 16 e 8 neurônios respectivamente.
- LSTM: 2 camada lstm com 16 e 8 neurônios respectivamente.
- GRU: 2 camada GRU 16 e 8 neurônios respectivamente.

Nos 3 casos, adicione uma camada final densa com 1 neurônio (nossa predição). Isto é, todas as redes terminam com 3 camadas ao todo. Utilize ativação sigmoid em todas as camadas.

Observação: Nos casos da LSTM e GRU, para empilhar mais do que uma camada, essas devem ter o parâmetro `return_sequences=True` , exceto a última. Isso faz com que a saída da camada inclua seu estado/sumário. Exemplo com 4 camadas em sequência:

```
model.add(LSTM(dim, return_sequences=True, input_shape=(1,1)))
model.add(LSTM(dim, return_sequences=True))
model.add(LSTM(dim, return_sequences=True))
model.add(LSTM(dim))
```

Treine os modelos por 15 épocas, batch size 1, com o otimizador Adam, learning rate 0.001 e loss MSE. Configure a seed 2 antes de treinar cada modelo.

Qual a alternativa correta?

a) pela rede densa tem mais parâmetros e obtêm o pior resultado de todos por não conseguir aprender tendências temporais.
b) a rede LSTM consegue um MSE melhor, mesmo tendo a maior quantidade de parametros entre todas as redes.
c) as 3 redes recebem de entrada os dados organizados da mesma maneira e rede LSTM consegue o melhor MSE.
d) A rede GRU consegue o melhor MSE, mesmo tendo menos parâmetros que a rede LSTM.

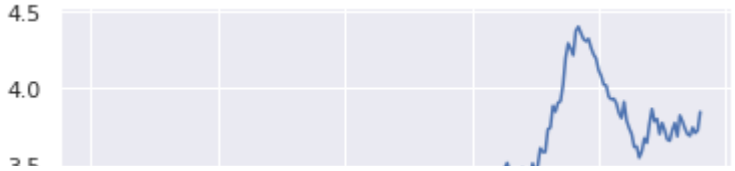
Justificativa

- a) Errado, a rede densa possui menos parametros.
- b) Errado, a LSTM tem um MSE inferior às demais;
- c) Errado, a LSTM tem um MSE inferior e os dados são organizados de uma maneira diferente para as redes RNNs com relação à densa.
- d) Correto.

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

from numpy.random import seed
from tensorflow.random import set_seed

from sklearn.metrics import mean_squared_error
from math import sqrt


import matplotlib.pyplot as plt
try:
    import seaborn as sns
    sns.set()
except:
    print("Seaborn não encontrado, pularemos o plot final.")
```

In [2]:
```
df = pd.read_csv("data/price_of_ground_chuck.csv")

# pega segunda coluna do dataframe
var = df.columns.values[1]
series = np.array(df[var])
plt.plot(series)
```

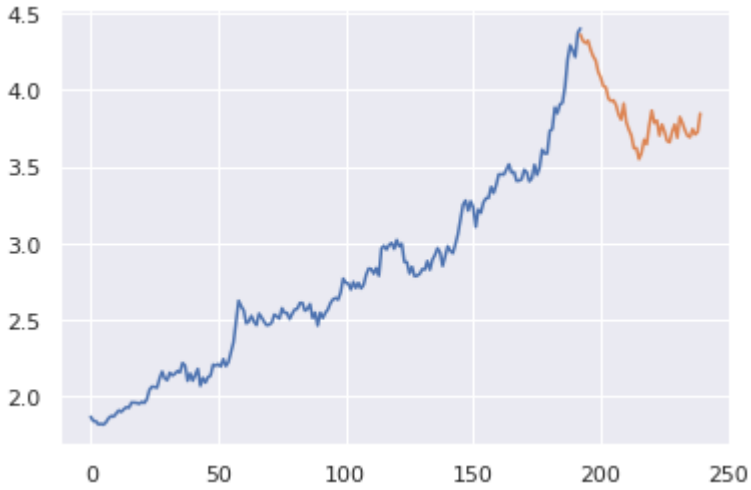
Out[2]:
```
[<matplotlib.lines.Line2D at 0x7f323f04b910>]
```



In [3]:
```
N = series.shape[0]
porc_treinamento = 80
# calcula tamanhos dos dados de treinamento (n1) e teste (n2)
n1 = int(series.shape[0]*(porc_treinamento/100.0))
n2 = int(series.shape[0]*(1-(porc_treinamento/100.0)))
# divide dados de treinamento e teste
train, test = series[0:-n2], series[-n2:]
print("Exemplos de Treinamento: ", n1)
print("Exemplos de Teste: ", n2)
plt.plot(np.arange(0, n1+1), train)
plt.plot(np.arange(n1,n1+n2), test)
```

```
Exemplos de Treinamento:  192
Exemplos de Teste:  48
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x7f323efb1430>]
```



In [4]:
```
# modifica uma serie temporal tornando-a
# um problema de aprendizado supervisionado
def timeseries_to_supervised(series, look_back=1):
    x = series[:-look_back]
    y = np.array(series[look_back:], copy=True)
    return x,y

def normalize(train, test):
    d_max = np.max(train)
    d_min = np.min(train)
    return ((train - d_min) / (d_max-d_min)), ((test - d_min) / (d_max-d_m
```

In [5]:
```
look_back = 1

n_train, n_test = normalize(train, test)

x_train, y_train = timeseries_to_supervised(n_train, look_back)
x_test, y_test = timeseries_to_supervised(n_test, look_back)

# formato deve ser [samples, time steps, features]
rnn_train = np.reshape(x_train, (x_train.shape[0], 1, 1))
rnn_test = np.reshape(x_test, (x_test.shape[0], 1, 1))
```

In [6]:
```
seed(2)
set_seed(2)
modelDense = keras.models.Sequential()
modelDense.add(keras.layers.Dense(16, activation='sigmoid', input_shape=(1
modelDense.add(keras.layers.Dense(8, activation='sigmoid'))
modelDense.add(keras.layers.Dense(1, activation='sigmoid'))
modelDense.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1, 16)             32
_________________________________________________________________
dense_1 (Dense)              (None, 1, 8)              136
_________________________________________________________________
dense_2 (Dense)              (None, 1, 1)              9
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
_________________________________________________________________
```

In [7]:
```
seed(2)
set_seed(2)
modelLSTM = keras.models.Sequential()
modelLSTM.add(keras.layers.LSTM(16, return_sequences=True, activation='sig
modelLSTM.add(keras.layers.LSTM(8, activation='sigmoid'))
modelLSTM.add(keras.layers.Dense(1, activation='sigmoid'))
modelLSTM.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 16)             1152
_________________________________________________________________
lstm_1 (LSTM)                (None, 8)                 800
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 1,961
Trainable params: 1,961
Non-trainable params: 0
_________________________________________________________________
```

In [8]:
```
seed(2)
set_seed(2)
modelGRU = keras.models.Sequential()
modelGRU.add(keras.layers.GRU(16, return_sequences=True, activation='sigmo
modelGRU.add(keras.layers.GRU(8, activation='sigmoid'))
modelGRU.add(keras.layers.Dense(1, activation='sigmoid'))
modelGRU.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 1, 16)             912
_________________________________________________________________
gru_1 (GRU)                  (None, 8)                 624
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 9
=================================================================
```

```
Total params: 1,545
Trainable params: 1,545
Non-trainable params: 0
_________________________________________________________________
```

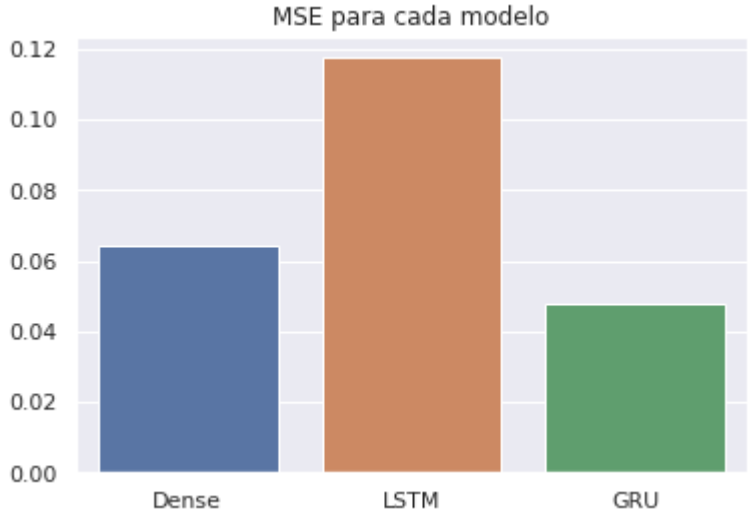
In [9]:
```
# Models é um dicionário com o primeiro elemento o conjunto de treino,
# o segundo o de teste e por último a rede em si.
# Perceba que a rede densa não usa o mesmo input que as RNNs.
models = {
    "Dense": (x_train, x_test, modelDense),
    "LSTM": (rnn_train, rnn_test, modelLSTM),
    "GRU": (rnn_train, rnn_test, modelGRU),
}

batch_size = 1
epochs = 15
learning_rate = 0.001

models_mse = {}
for name, (train_model_data, test_model_data, model) in models.items():
    print(name)
    seed(2)
    set_seed(2)
    model.compile(loss='mean_squared_error',
                  optimizer=keras.optimizers.Adam(lr=learning_rate),
                  metrics=['mae'])
    hist = model.fit(train_model_data, y_train, epochs=epochs,
                batch_size=batch_size,
                verbose=0, shuffle=False)

    pred = model.predict(test_model_data).reshape(-1)
    mse = mean_squared_error(y_test, pred)
    print(f"MSE de {name}: {mse:.4}\n\n\n")
    models_mse[name] = mse
```

```
Dense
MSE de Dense: 0.06415



LSTM
MSE de LSTM: 0.1175



GRU
MSE de GRU: 0.04773
```

In [10]:
```
models_mse
```

Out[10]:
```
{'Dense': 0.06414968948239423,
 'LSTM': 0.11754443812569394,
 'GRU': 0.04773285639581572}
```

In [11]:
```
try:
    results = pd.Series(models_mse)
    sns.barplot(y=results.values, x=results.index)
    plt.title("MSE para cada modelo")
except:
    print("Seaborn não instalado")
```



Exercício 7)

Carregue o arquivo `livro1.txt` conforme indicado abaixo, bem como o word embedding Glove com 50 dimensões em português utilizado em aula. Obtenha uma vetorização de texto com número máximo de tokens 5000 e tamanho da sentença com no máximo 30 tokens. A seguir obtenha a matriz de embeddings a partir das palavras desse arquivo e imprima palavras não convertidas. Qual a natureza da maior parte das palavras em que houveram falhas?

(a) Palavras com hífens, números e em outro idioma
(b) Nomes próprios e caracteres especiais isolados
(c) Palavras com caracteres não codificados corretamente no UTF-16
(d) Palavras em outro idioma e caracteres isolados

Justificativa: Veja abaixo. As palavras que usam hífens como em "queixava-se" deveriam ter sido pré-processadas e usando o vectorizer não foram reconhecidas. Também números e palavras em outros idiomas foram encontradas e não existem no vocabulário utilizado.

In [12]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import models
from numpy.random import seed
from tensorflow.random import set_seed
```

In [13]:
```
path_to_glove_file = "./glove_s50.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Encontrados %s word vectors." % len(embeddings_index))
```

```
<ipython-input-13-ecfee5d4ff7a>:7: DeprecationWarning: string or file could
not be read to its end due to unmatched data; this will raise a ValueError
in the future.
  coefs = np.fromstring(coefs, "f", sep=" ")
Encontrados 929594 word vectors.
```

In [14]:
```
df = pd.read_csv("data/livro1.txt", delimiter='[\n\r]+', header=None, engi
df
```

Out[14]:

	0
0	Os curiosos acontecimentos que são o objeto de...
1	Segundo a opinião geral, estavam deslocados, j...
2	À primeira vista, Oran é, na verdade, uma cida...
3	A própria cidade, vamos admiti-lo, é feia
4	com seu aspecto tranquilo, é preciso algum tem...
...	...
538	Queimava, na verdade, mas nem mais nem menos d...
539	Toda a cidade estava com febre
540	Era essa pelo menos a impressão que perseguia ...
541	Mas essa impressão parecia-lhe insensata
542	Atribuía-a ao enervamento e às preocupações qu...

543 rows × 1 columns

In [15]:
```
texto = np.array(df)
print(texto[:5])
print(texto.shape)
```

```
[['Os curiosos acontecimentos que são o objeto desta crônica ocorreram em 1
94x, em Oran']
 ['Segundo a opinião geral, estavam deslocados, já que saíam um pouco do co
mum']
 ['À primeira vista, Oran é, na verdade, uma cidade comum e não passa de um
a prefeitura francesa na costa argelina']
 ['A própria cidade, vamos admiti-lo, é feia']
 ['com seu aspecto tranquilo, é preciso algum tempo para se perceber o que
a torna diferente de tantas outras cidades comerciais em todas as latitudes
']]
(543, 1)
```

In [16]:
```
seed(1)
set_seed(2)

from tensorflow.keras.layers.experimental.preprocessing import TextVectori

vectorizer = TextVectorization(max_tokens=5000, output_sequence_length=30)
text_ds = tf.data.Dataset.from_tensor_slices(texto).batch(16)
vectorizer.adapt(text_ds)

voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
```

In [17]:
```
num_tokens = len(voc) + 2
print("Número de tokens: ", num_tokens)
embedding_dim = 50
convertidas = 0
falhas = 0

embedding_matrix = np.zeros((num_tokens, embedding_dim))
print(embedding_matrix.shape)
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        if (embedding_vector.shape[0] != embedding_dim):
            falhas += 1
        else:
            embedding_matrix[i] = embedding_vector
            convertidas += 1
    else:
        falhas += 1
        print(word)

print("Palavras convertidas: %d / não convertidas: %d)" % (convertidas, fal
```

```
Número de tokens:  2194
(2194, 50)

[UNK]
à
É
tarrou
às
À
paneloux
cottard
vigiálo
tinhamno
respondilhe
ransdoc
queixavase
ocupouse
acompanhouo
18
voltavalhe
veiolhe
veemse
têlos
tratarao
tinhamlhe
sentilo
sentiamse
sentase
```

```
saintjust
ríchard
reúnemse
purgava
proíboo
preparavase
perdiase
percebese
pareceralhe
parecelhe
ouviamse
olhavaos
ocupamse
obrigavao
leválas
lançarse
jogouse
isolálo
interrogálo
interessamse
instalavaa
instalamse
inquietarse
inclinavase
gracejadores
fungosidades
frontdemer
fixarase
fechála
faziao
explicarlhe
exigiamse
esperouos
escarrara
encontrálos
encontrála
encontravamnos
empoleirarse
dãolhe
distraise
dirseia
detevese
desconhecese
desateio
demorouse
deitese
darmes
curvavamse
cuidese
contemplouo
consolálo
compreendiase
carroleito
boulomanes
bemeducados
bemeducada
aviseme
atribuíaa
atirálos
atirouse
assuou
apressavamse
apoiavase
apertoulhe
```

```
amicales
alastravamse
agravouse
agitouse
afligiase
afastase
adormecese
admitilo
acusavamse
acalmese
abraçoua
abateramse
30
28
25
194x
17
16
1
```

Exercício 8)

A matriz de embedding (`embedding_matrix`) contém a mesma representação para uma dada palavra do que o índice de embedding (`embedding_index`), mas a quantidade de elementos da matriz é limitada ao vocabulário da base de dados que estamos trabalhando. Imprima a quantidade de elementos em ambos para conferir a diferença.

Utilize a configuração obtida no exercício anterior (base de texto e word embedding carregados), e obtenha os índices das palavras: `rato` , `médico` , `cidade` e `febre` calculados pelo vetorizador.

A seguir, obtenha a soma da diferença absoluta (ou distância L1) entre as representações da palavra `rato` e aqueles obtidos por meio dos índices encontrados das palavras `médico` , `ouro` , `cidade` e `febre` na matriz de embedding (`embedding_matrix`), ou seja, dentre as palavras convertidas.

Qual palavra, das citadas acima dentro do vocabulário, é mais próxima de `rato` segundo o word embedding utilizado?

(a) Febre
(b) Ouro
(c) Médico
(d) Médico e Febre empatados

Justificativa: Veja abaixo, a palavra "ouro" não está no vocabulário, sendo atribuído o índice 1. Assim, a palavra mais próxima dentro do vocabulário é "Febre"

In [18]:
```
print("Tamanho do índice: ", len(embeddings_index))
print("Tamanho da matriz: ", embedding_matrix.shape[0])
```

```
Tamanho do índice:  929594
Tamanho da matriz:  2194
```

In [19]:
```
output = vectorizer([["rato médico ouro cidade febre"]])
output
```

Out[19]:
```
<tf.Tensor: shape=(1, 30), dtype=int64, numpy=
array([[ 82,  33,   1,  27, 110,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0]])>
```

In [20]:
```
print("Médico: ", np.sum(np.abs(embedding_matrix[33]-embedding_matrix[82])))
print("Cidade: ", np.sum(np.abs(embedding_matrix[27]-embedding_matrix[82])))
print("Ouro: ", np.sum(np.abs(embedding_matrix[1]-embedding_matrix[82])))
print("Febre: ", np.sum(np.abs(embedding_matrix[110]-embedding_matrix[82])))
```

```
Médico:  33.421877262182534
Cidade:  39.917805386241525
Ouro:  22.207107034511864
Febre:  31.358608989976346
```

Exercício 9)

Word2Vec com aprendizado de uma rede recorrente para classificação de sentenças.

Iremos treinar uma rede neural capaz de diferenciar frases de dois livros diferentes, contidos nos arquivos `livro1.txt` e `livro2.txt` .

Com base no processo realizado nos exercícios anteriores, carregue os arquivos citados. Gere o vetorizador com máximo de tokens 5000 e sequência com máximo de 60 elementos/tokens e realize a conversão com os textos de ambos os livros em conjunto.

Depois crie o conjunto de treinamento e teste da seguinte forma:

- o conjunto de treinamento terá as 75% primeiras frases do livro1 seguida das 75% primeiras frases do livro2
- o conjunto de teste terá as 25% frases restantes do livro1 seguida das 25% frases restantes do livro 2
- monte vetores com os rótulos de treinamento e teste de forma que livro 1 seja a classe 0 e livro 2 seja a classe 1.

Projete uma rede Convolucional-recorrente com as seguintes camadas (todas com ativação ReLU exceto especificado outra)

- Embedding layer
- Conv1D com 32 filtros de tamanho 2 e zeropadding
- GRU com 32 unidades
- Densa com 32 unidades
- Dropout com taxa 25%
- Densa com 1 unidade e ativação sigmoide

Configure as sementes com seed(1) e set_seed(2), depois compile com a função entropia cruzada binária, otimizador adam (com seus parâmetros padrão) e compute a acurácia. Treine por 20 épocas com batch size 16.

Após o treinamento, a acurácia no conjunto de teste (use o método `evaluate`) está em qual intervalo?

(a) [99, 100]
(b) [86, 91]
(c) [92, 98]
(d) [79, 85]

Justificativa: Veja abaixo o código

In [21]:
```
df2 = pd.read_csv("data/livro2.txt", delimiter='[\n\r]+', header=None, eng
df2
```

Out[21]:

	0
0	No princípio era o Verbo e o Verbo estava junt...
1	Ele estava no princípio junto a Deus e dever d...
2	Mas videmus nunc per speculum et in aenigmate ...
3	Chegando ao fim desta minha vida de pecador, e...
4	Conceda-me o Senhor a graça de ser testemunha ...
...	...
96	Pela mole, e pela forma, o Edifício me pareceu...
97	E é sorte que, sendo uma límpida manhã de inve...
98	Não direi de modo algum que ela sugerisse sent...
99	Trouxe-me espanto, e uma inquietação sutil
100	Deus sabe que não eram fantasmas de minha alma...

101 rows × 1 columns

In [22]:
```
texto2 = np.array(df2)
print(texto2[:5])
print(texto2.shape)
```

```
[['No princípio era o Verbo e o Verbo estava junto a Deus, e o Verbo era De
us']
 ['Ele estava no princípio junto a Deus e dever do monge fiel seria repetir
cada dia com salmodiante humildade o único evento imodificável do qual se p
ode confirmar a incontrovertível verdade']
 ['Mas videmus nunc per speculum et in aenigmate e a verdade, em vez de car
a a cara, manifesta-se deixando às vezes rastros (ai, quão ilegíveis) no er
ro do mundo, tanto que precisamos calculá-lo, soletrando os verdadeiros sin
ais, mesmo lá onde nos parecem obscuros e quase entremeados por uma vontade
totalmente voltada para o mal']
 ['Chegando ao fim desta minha vida de pecador, enquanto, encanecido, envel
heço como o mundo, à espera de perder-me no abismo sem fundo da divindade s
ilenciosa e deserta, participando da luz inconversível das inteligências an
gélicas, já entrevado com meu corpo pesado e doente nesta cela do caro most
eiro de Melk, apresto-me a deixar sobre este pergaminho o testemunho dos ev
entos miríficos e formidáveis a que na juventude me foi dado assistir, repe
tindo verbatim quanto vi e ouvi, sem me aventurar a tirar disso um desenho,
como a deixar aos que virão (se o Anticristo não os preceder) signos de sig
nos, para que sobre eles se exercite a prece da decifração']
 ['Conceda-me o Senhor a graça de ser testemunha transparente dos acontecim
entos que tiveram lugar na abadia da qual é bem e piedoso se cale também af
```

```
inal o nome, ao findar do ano do Senhor de 1327 em que o imperador Ludovico
entrou na Itália para reconstituir a dignidade do sagrado império romano, s
egundo os desígnios do Altíssimo e a confusão do infame usurpador simoníaco
e heresiarca que em Avignon lançou vergonha ao santo nome do apóstolo (falo
da alma pecadora de Jacques de Cahors, que os ímpios honraram como João XXI
I)']]
```

In [23]:
```
texto_integrado = np.vstack((texto, texto2))
```

In [24]:
```
seed(1)
set_seed(2)

from tensorflow.keras.layers.experimental.preprocessing import TextVectori

vectorizer2 = TextVectorization(max_tokens=5000, output_sequence_length=60
text_ds2 = tf.data.Dataset.from_tensor_slices(texto_integrado).batch(16)
vectorizer2.adapt(text_ds2)

voc2 = vectorizer2.get_vocabulary()
word_index2 = dict(zip(voc2, range(len(voc2))))
```

In [25]:
```
num_tokens = len(voc2) + 2
print("Número de tokens: ", num_tokens)
embedding_dim = 50
convertidas = 0
falhas = 0

embedding_matrix2 = np.zeros((num_tokens, embedding_dim))
print(embedding_matrix2.shape)
for word, i in word_index2.items():
    embedding_vector2 = embeddings_index.get(word)
    if embedding_vector2 is not None:
        if (embedding_vector2.shape[0] != embedding_dim):
            falhas += 1
        else:
            embedding_matrix2[i] = embedding_vector2
            convertidas += 1
    else:
        falhas += 1

print("Palavras convertidas: %d / não convertidas: %d)" % (convertidas, fal
```

```
Número de tokens:  3200
(3200, 50)
Palavras convertidas: 3025 / não convertidas: 173)
```

In [26]:
```
from tensorflow.keras.layers import Embedding

embedding_layer = Embedding(
    num_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix2),
    trainable=False,
)
```

In [27]:
```
validation_split = 0.25

# texto 1
num_validation1 = int(validation_split * len(texto))
num_train1 = int((1-validation_split) * len(texto))

x_train1 = texto[:-num_validation1]
y_train1 = np.zeros(num_train1+1)
x_val1 = texto[-num_validation1:]
y_val1 = np.zeros(num_validation1)
print("Texto 1:")
print("train (x,y): %d,%d" % (x_train1.shape[0], y_train1.shape[0]))
print("val (x,y): %d,%d" % (x_val1.shape[0], y_val1.shape[0]))

# texto 2
num_validation2 = int(validation_split * len(texto2))
num_train2 = int((1-validation_split) * len(texto2))
x_train2 = texto2[:-num_validation2]
y_train2 = np.ones(num_train2+1)
x_val2 = texto2[-num_validation2:]
y_val2 = np.ones(num_validation2)
print("Texto 2:")
print("train (x,y): %d,%d" % (x_train2.shape[0], y_train2.shape[0]))
print("val (x,y): %d,%d" % (x_val2.shape[0], y_val2.shape[0]))
```

```
Texto 1:
train (x,y): 408,408
val (x,y): 135,135
Texto 2:
train (x,y): 76,76
val (x,y): 25,25
```

In [28]:
```
# juntando textos
x_train = np.vstack((x_train1, x_train2))
x_val = np.vstack((x_val1, x_val2))
print(x_train.shape)
print(x_val.shape)

y_train = np.concatenate((y_train1, y_train2))
y_val = np.concatenate((y_val1, y_val2))
```

```
(484, 1)
(160, 1)
```

In [29]:
```
# vetorizacao dos dados
x_train_net = vectorizer(np.array([s for s in x_train])).numpy()
x_val_net = vectorizer(np.array([s for s in x_val])).numpy()

print(x_train_net.shape)
print(x_val_net.shape)
```

```
(484, 30)
(160, 30)
```

In [30]:
```
def GRU_texto():
    int_sequences_input = keras.Input(shape=(None,), dtype="int64")
    embedded_sequences = embedding_layer(int_sequences_input)
    x = layers.Conv1D(32, 2, activation="relu", padding="same")(embedded_s
    x = layers.GRU(32, input_shape=(1, 1), activation="relu")(x)
    x = layers.Dense(32, activation="relu")(x)
    x = layers.Dropout(0.25)(x)
    predsGRUout = layers.Dense(1, activation="sigmoid")(x)
    modelGRU = keras.Model(inputs=int_sequences_input, outputs=predsGRUout
    return modelGRU
```

In [31]:
```
modelGRU = GRU_texto()
modelGRU.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding (Embedding)        (None, None, 50)          160000
_________________________________________________________________
conv1d (Conv1D)              (None, None, 32)          3232
_________________________________________________________________
gru_2 (GRU)                  (None, 32)                6336
_________________________________________________________________
dense_5 (Dense)              (None, 32)                1056
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 33
=================================================================
Total params: 170,657
Trainable params: 10,657
Non-trainable params: 160,000
_________________________________________________________________
```

In [32]:
```
seed(1)
set_seed(2)

modelGRU.compile(
    loss="binary_crossentropy", optimizer="adam", metrics=["acc"]
)
modelGRU.fit(x_train_net, y_train, batch_size=16, epochs=20, validation_da
```

```
Epoch 1/20
31/31 [==============================] - 0s 15ms/step - loss: 0.6776 - acc:
0.8202 - val_loss: 0.6477 - val_acc: 0.8438
Epoch 2/20
31/31 [==============================] - 0s 6ms/step - loss: 0.6073 - acc:
0.8450 - val_loss: 0.5494 - val_acc: 0.8438
Epoch 3/20
31/31 [==============================] - 0s 6ms/step - loss: 0.4788 - acc:
0.8533 - val_loss: 0.4188 - val_acc: 0.8438
Epoch 4/20
31/31 [==============================] - 0s 6ms/step - loss: 0.3522 - acc:
0.8595 - val_loss: 0.3764 - val_acc: 0.8687
Epoch 5/20
31/31 [==============================] - 0s 6ms/step - loss: 0.2689 - acc:
0.9132 - val_loss: 0.3008 - val_acc: 0.9125
Epoch 6/20
31/31 [==============================] - 0s 6ms/step - loss: 0.1611 - acc:
0.9463 - val_loss: 0.1885 - val_acc: 0.9312
```

```
Epoch 7/20
31/31 [==============================] - 0s 6ms/step - loss: 0.1645 - acc:
0.9421 - val_loss: 0.1846 - val_acc: 0.9125
Epoch 8/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0760 - acc:
0.9835 - val_loss: 0.1470 - val_acc: 0.9438
Epoch 9/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0640 - acc:
0.9835 - val_loss: 0.1333 - val_acc: 0.9563
Epoch 10/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0407 - acc:
0.9917 - val_loss: 0.1275 - val_acc: 0.9625
Epoch 11/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0374 - acc:
0.9897 - val_loss: 0.1746 - val_acc: 0.9500
Epoch 12/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0275 - acc:
0.9917 - val_loss: 0.1545 - val_acc: 0.9625
Epoch 13/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0181 - acc:
0.9959 - val_loss: 0.2392 - val_acc: 0.9625
Epoch 14/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0187 - acc:
0.9897 - val_loss: 0.1578 - val_acc: 0.9563
Epoch 15/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0266 - acc:
0.9917 - val_loss: 0.2501 - val_acc: 0.9500
Epoch 16/20
31/31 [==============================] - 0s 6ms/step - loss: 0.1032 - acc:
0.9649 - val_loss: 0.2604 - val_acc: 0.9312
Epoch 17/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0287 - acc:
0.9917 - val_loss: 0.1780 - val_acc: 0.9625
Epoch 18/20
31/31 [==============================] - 0s 6ms/step - loss: 0.0509 - acc:
0.9876 - val_loss: 0.1147 - val_acc: 0.9500
Epoch 19/20
31/31 [==============================] - 0s 7ms/step - loss: 0.0091 - acc:
0.9979 - val_loss: 0.1770 - val_acc: 0.9688
Epoch 20/20
31/31 [==============================] - 0s 6ms/step - loss: 9.6773e-04 - a
cc: 1.0000 - val_loss: 0.2046 - val_acc: 0.9688
```

Out[32]: `<tensorflow.python.keras.callbacks.History at 0x7f31cde8bd30>`

In [33]:
```
scores = modelGRU.evaluate(x_val_net,y_val, verbose=0)
print("Perda validação: %f" % (scores[0]))
print("Acurácia validação: %f" % (scores[1]))
```

```
Perda validação: 0.204649
Acurácia validação: 0.968750
```

Exercício 10)

Word2Vec com aprendizado de uma rede totalmente convolucional, com convoluções dilatadas para classificação de sentenças.

Considere o mesmo problema anteriormente abordado no exercício 9. Agora vamos utilizar uma rede Convolucional com convoluções dilatadas que é um método para reduzir dimensionalidade enquanto mantêm o campo receptivo local, utilizada por exemplo para

aplicações com áudio (como a WaveNet), mas também aplicável para outros dados sequenciais. Veja uma explicação em: https://www.paperswithcode.com/method/dilated-convolution

A rede deve ter as seguintes camadas

- Embedding layer
- Conv1D com 32 filtros de tamanho 2, zeropadding e parâmetro `dilation_rate=2`
- Conv1D com 32 filtros de tamanho 2, zeropadding e parâmetro `dilation_rate=3`
- Conv1D com 64 filtros de tamanho 3, zeropadding e parâmetro `dilation_rate=4`
- Dropout com taxa 25%
- Densa com 1 unidade e ativação sigmoide

Configure as sementes com seed(1) e set_seed(2), depois compile com a função entropia cruzada binária, otimizador adam (com seus parâmetros padrão) e compute a acurácia. Treine por 20 épocas com batch size 16.

A acurácia no conjunto de teste (use o método `evaluate`) está em qual intervalo?

(a) [99, 100]
(b) [86, 91]
(c) [92, 98]
(d) [79, 85]

Justificativa: Veja abaixo o código

In [34]:
```
def Conv_texto():
    int_sequences_input = keras.Input(shape=(None,), dtype="int64")
    embedded_sequences = embedding_layer(int_sequences_input)
    x = layers.Conv1D(32, 2, activation="relu", dilation_rate=2, padding="
    x = layers.Conv1D(32, 2, activation="relu", dilation_rate=3, padding="
    x = layers.Conv1D(64, 3, activation="relu", dilation_rate=4, padding="
    x = layers.Dropout(0.25)(x)
    predsConvout = layers.Dense(1, activation="sigmoid")(x)
    modelConv = keras.Model(inputs=int_sequences_input, outputs=predsConvou
    return modelConv

modelConv = Conv_texto()
modelConv.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding (Embedding)        (None, None, 50)          160000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, None, 32)          3232
_________________________________________________________________
conv1d_2 (Conv1D)            (None, None, 32)          2080
_________________________________________________________________
conv1d_3 (Conv1D)            (None, None, 64)          6208
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 64)          0
_________________________________________________________________
dense_7 (Dense)              (None, None, 1)           65
=================================================================
```

```
Total params: 171,585
Trainable params: 11,585
Non-trainable params: 160,000
```

In [35]:
```
seed(1)
set_seed(2)

modelConv.compile(
    loss="binary_crossentropy", optimizer="adam", metrics=["acc"]
)
modelConv.fit(x_train_net, y_train, batch_size=16, epochs=20, validation_da
```

```
Epoch 1/20
31/31 [==============================] - 0s 5ms/step - loss: 0.6017 - acc:
0.8377 - val_loss: 0.5461 - val_acc: 0.8435
Epoch 2/20
31/31 [==============================] - 0s 3ms/step - loss: 0.4931 - acc:
0.8431 - val_loss: 0.4190 - val_acc: 0.8435
Epoch 3/20
31/31 [==============================] - 0s 3ms/step - loss: 0.3844 - acc:
0.8449 - val_loss: 0.3886 - val_acc: 0.8435
Epoch 4/20
31/31 [==============================] - 0s 3ms/step - loss: 0.3517 - acc:
0.8521 - val_loss: 0.3652 - val_acc: 0.8533
Epoch 5/20
31/31 [==============================] - 0s 2ms/step - loss: 0.3225 - acc:
0.8684 - val_loss: 0.3470 - val_acc: 0.8681
Epoch 6/20
31/31 [==============================] - 0s 2ms/step - loss: 0.2946 - acc:
0.8871 - val_loss: 0.3279 - val_acc: 0.8781
Epoch 7/20
31/31 [==============================] - 0s 2ms/step - loss: 0.2773 - acc:
0.9003 - val_loss: 0.3649 - val_acc: 0.8433
Epoch 8/20
31/31 [==============================] - 0s 2ms/step - loss: 0.2639 - acc:
0.9024 - val_loss: 0.3127 - val_acc: 0.8883
Epoch 9/20
31/31 [==============================] - 0s 2ms/step - loss: 0.2388 - acc:
0.9174 - val_loss: 0.3406 - val_acc: 0.8821
Epoch 10/20
31/31 [==============================] - 0s 3ms/step - loss: 0.2288 - acc:
0.9203 - val_loss: 0.3108 - val_acc: 0.8842
Epoch 11/20
31/31 [==============================] - 0s 3ms/step - loss: 0.2195 - acc:
0.9245 - val_loss: 0.3048 - val_acc: 0.8906
Epoch 12/20
31/31 [==============================] - 0s 2ms/step - loss: 0.2117 - acc:
0.9258 - val_loss: 0.3081 - val_acc: 0.8919
Epoch 13/20
31/31 [==============================] - 0s 2ms/step - loss: 0.2026 - acc:
0.9304 - val_loss: 0.3011 - val_acc: 0.8883
Epoch 14/20
31/31 [==============================] - 0s 2ms/step - loss: 0.1899 - acc:
0.9351 - val_loss: 0.3048 - val_acc: 0.8904
Epoch 15/20
31/31 [==============================] - 0s 2ms/step - loss: 0.1949 - acc:
0.9336 - val_loss: 0.3102 - val_acc: 0.8854
Epoch 16/20
31/31 [==============================] - 0s 2ms/step - loss: 0.1760 - acc:
0.9424 - val_loss: 0.3082 - val_acc: 0.8923
Epoch 17/20
31/31 [==============================] - 0s 2ms/step - loss: 0.1814 - acc:
0.9387 - val_loss: 0.3068 - val_acc: 0.8910
```

```
Epoch 18/20
31/31 [==============================] - 0s 2ms/step - loss: 0.1679 - acc:
0.9449 - val_loss: 0.3109 - val_acc: 0.8929
Epoch 19/20
31/31 [==============================] - 0s 3ms/step - loss: 0.1610 - acc:
0.9485 - val_loss: 0.3242 - val_acc: 0.8975
Epoch 20/20
31/31 [==============================] - 0s 3ms/step - loss: 0.1575 - acc:
```

Out[35]: <tensorflow.python.keras.callbacks.History at 0x7f31cdb2b6a0>

In [36]:
```
scores = modelConv.evaluate(x_val_net,y_val, verbose=0)
print("Perda validação: %f" % (scores[0]))
print("Acurácia validação: %f" % (scores[1]))
```

```
Perda validação: 0.338944
Acurácia validação: 0.895625
```

In [ ]: