Computational Intelligence Lab

Assignment 7

Karolina Kotlowska IO czw. 9:30

7.1 Load and preprocess IMDB reviews

7.1.1 Load IMDB reviews dataset

In [1]:
```
import tensorflow_datasets as tfds
import tensorflow as tf

ds_train = tfds.load('imdb_reviews', split='train', as_supervised=True, s
ds_test = tfds.load('imdb_reviews', split='test', as_supervised=True, shu
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-p
ackages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/
user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
import pandas as pd
data = [(text.numpy().decode('UTF8'),label.numpy()) for text,label in ds_
df_train = pd.DataFrame(data,columns=['text','label'])
df_train.head()
```

```
2023-04-20 17:46:59.998542: W tensorflow/tsl/platform/profile_utils/cpu_
utils.cc:128] Failed to get CPU frequency: 0 Hz
```

Out[2]:

	text	label
0	This was an absolutely terrible movie. Don't b...	0
1	I have been known to fall asleep during films,...	0
2	Mann photographs the Alberta Rocky Mountains i...	0
3	This is the kind of film for a snowy Sunday af...	1
4	As others have mentioned, all the women that g...	1

In [3]:
```
data = [(text.numpy().decode('UTF8'),label.numpy()) for text,label in ds_
df_test = pd.DataFrame(data,columns=['text','label'])
df_test.head()
```

Out[3]:

	text	label
0	There are films that make careers. For George ...	1
1	A blackly comic tale of a down-trodden priest,...	1
2	Scary Movie 1-4, Epic Movie, Date Movie, Meet ...	0
3	Poor Shirley MacLaine tries hard to lend some ...	0
4	As a former Erasmus student I enjoyed this fil...	1

7.1.2 Basic preprocessing

We define `clean_text` function, which removes separators, stopwords and optionally aplies lemmatization (conversion of words to basic forms)

In [4]:
```
import re
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
stopwords = stopwords.words('english')
nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer


def clean_text(text,stopwords,lemmatize=False):
  text = text.lower()
  text = re.sub(r"<br />", "", text)
  # text = re.sub(r"[,;`^\.\"\'!?:_()%&{}*+\#\$-/\\<>=@|~\]\[]", " ", tex
  text = re.sub(r"[,;`^\.\"!?:_()%&{}*+\#\$-/\\<>=@|~\]\[]", " ", text)
  words = text.split()
  words = list(filter(lambda w:not w in stopwords, words))
  if lemmatize:
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(token) for token in words]
    words = [lemmatizer.lemmatize(token, "v") for token in words]
  return ' '.join(words)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/spoton/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/spoton/nltk_data...
```

Small test

In [5]:
```
clean_text("he's running for a president",stopwords)
```

Out[5]:
```
'running president'
```

In [6]:
```
df_train['text_preprocessed'] = df_train.text.apply(lambda t:clean_text(t
df_test['text_preprocessed'] = df_test.text.apply(lambda t:clean_text(t,s
```

In [7]:
```
df_train.head()
```

Out[7]:

	text	label	text_preprocessed
0	This was an absolutely terrible movie. Don't b...	0	absolutely terrible movie lured christopher wa...
1	I have been known to fall asleep during films,...	0	known fall asleep films usually due combinatio...
2	Mann photographs the Alberta Rocky Mountains i...	0	mann photographs alberta rocky mountains super...
3	This is the kind of film for a snowy Sunday af...	1	kind film snowy sunday afternoon rest world go...
4	As others have mentioned, all the women that g...	1	others mentioned women go nude film mostly abs...

7.1.3 Conversion to sequences

In further experiments we will use `text_preprocessed` column

In [8]:
```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=10_000)
tokenizer.fit_on_texts(df_train.text_preprocessed)
seq_train = tokenizer.texts_to_sequences(df_train.text_preprocessed)
seq_test = tokenizer.texts_to_sequences(df_test.text_preprocessed)
```

In [9]:
```
# from keras import preprocessing
# maxlen = 500
# seq_train_padded = preprocessing.sequence.pad_sequences(seq_train, maxl
# seq_test_padded = preprocessing.sequence.pad_sequences(seq_test, maxlen

# FIXME

# from keras import preprocessing
import keras
maxlen = 500
seq_train_padded = keras.utils.pad_sequences(seq_train, maxlen=maxlen)
seq_test_padded = keras.utils.pad_sequences(seq_test, maxlen=maxlen)
```

7.2 Simple RNN

Simple RNN accepts a sequence of inputs $[x(0), x(1), \ldots, x(maxlen-1)]$ and applies the following formula for updating internal state h:

$$h(t+1) = f(W \cdot x(t) + U \cdot h(t) + b)$$

- $x(t)$ - input sequence $[x(0), x(1), \ldots x(maxlen-1)]$
- $h(t)$ hidden state passed between iterations

Finally, at the end of sequence it yields $h(maxlen-1)$

In [10]:
```
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.layers import Embedding, SimpleRNN
from keras.layers import GlobalMaxPool1D, MaxPool1D, Dropout
```

```
batch_size = 256
max_features=10_000

model = Sequential()
model.add(Embedding(input_dim=10_000, output_dim=64, input_length=maxlen)
model.add(SimpleRNN(units = 32,return_sequences=True))
model.add(GlobalMaxPool1D())
model.add(Dense(20, activation="relu"))
model.add(Dropout(0.05))
model.add(Dense(2, activation='softmax'))
# model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy'
# model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 500, 64)           640000    
                                                                 
 simple_rnn (SimpleRNN)      (None, 500, 32)           3104      
                                                                 
 global_max_pooling1d (Globa  (None, 32)               0         
 lMaxPooling1D)                                                  
                                                                 
 dense (Dense)               (None, 20)                660       
                                                                 
 dropout (Dropout)           (None, 20)                0         
                                                                 
 dense_1 (Dense)             (None, 2)                 42        
                                                                 
=================================================================
Total params: 643,806
Trainable params: 643,806
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
epochs = 5
batch_size = 512
hist = model.fit(seq_train_padded, df_train.label, epochs = epochs, batch
```

```
Epoch 1/5
40/40 [==============================] - 13s 320ms/step - loss: 0.6880 - accuracy: 0.5361 - val_loss: 0.6788 - val_accuracy: 0.5262
Epoch 2/5
40/40 [==============================] - 13s 334ms/step - loss: 0.5842 - accuracy: 0.7293 - val_loss: 0.6368 - val_accuracy: 0.6134
Epoch 3/5
40/40 [==============================] - 13s 333ms/step - loss: 0.4302 - accuracy: 0.8201 - val_loss: 0.3977 - val_accuracy: 0.8256
Epoch 4/5
40/40 [==============================] - 14s 347ms/step - loss: 0.3299 - accuracy: 0.8711 - val_loss: 0.3461 - val_accuracy: 0.8518
Epoch 5/5
40/40 [==============================] - 14s 343ms/step - loss: 0.2617 - accuracy: 0.9015 - val_loss: 0.3985 - val_accuracy: 0.8150
```

TODO 7.2.1 Predict probabilities, then labels and print the classification report

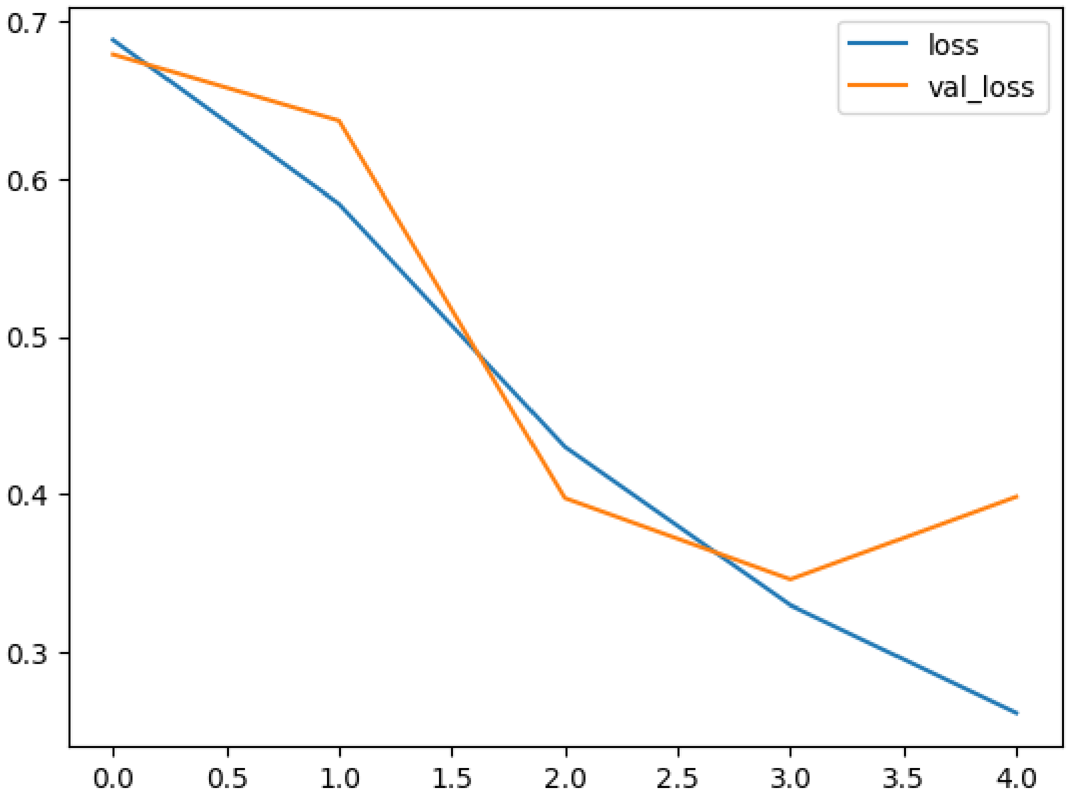
In [12]:
```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

y_test = [label.numpy() for img, label in ds_test]
y_test = np.array(y_test)
probs = model.predict(seq_test_padded)
y_pred = np.argmax(probs, axis=1)

print(classification_report(y_test, y_pred))
```

```
782/782 [==============================] - 10s 12ms/step
              precision    recall  f1-score   support

           0       0.93      0.67      0.77     12500
           1       0.74      0.95      0.83     12500

    accuracy                           0.81     25000
   macro avg       0.83      0.81      0.80     25000
weighted avg       0.83      0.81      0.80     25000
```

TODO 7.2.2 Plot the data collected in the history

In [14]:
```
import matplotlib.pyplot as plt

plt.plot(hist.history['loss'],label='loss')
plt.plot(hist.history['val_loss'],label='val_loss')
plt.legend()
```

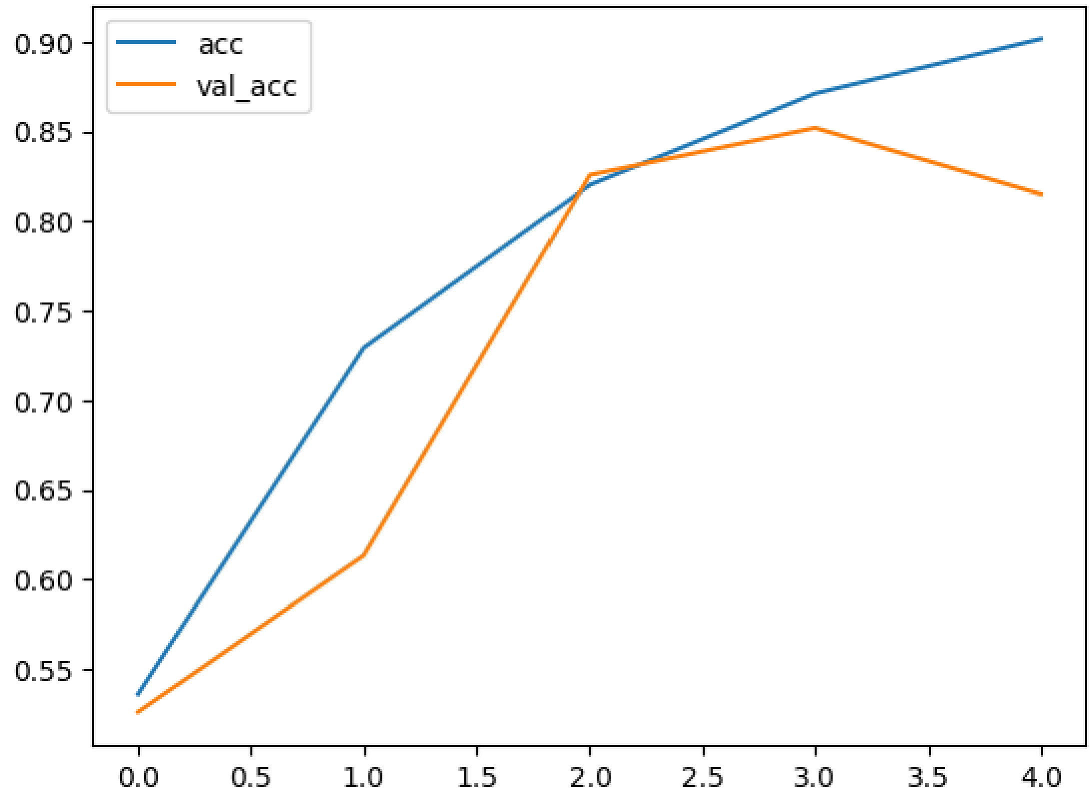
Out[14]: <matplotlib.legend.Legend at 0x2cd4e6fb0>



In [15]:
```
plt.plot(hist.history['accuracy'],label='acc')
plt.plot(hist.history['val_accuracy'],label='val_acc')
```

```
plt.legend()
```

Out[15]: <matplotlib.legend.Legend at 0x2d7f2e3b0>



7.3 Other recurrent layers implemented in keras

- LSTM
- Bidirectional(LSTM)
- GRU

Long-Term Short Term Memory LSTM networks

You may read about LSTM networks You may read about LSTM networks in this article.

LSTM is a special kind of RNN which is mainly useful for learning long-term dependencies. The name refers to the idea that the activations of a network correspond to short-term memory, while the weights correspond to long-term memory. If the activations can preserve information over long distances, that makes them long-term short-term memory.

Bidirectional

The idea of Bidirectional Recurrent Neural Networks (RNNs) is straightforward. It involves duplicating the first recurrent layer in the network so that there are now two layers side-by-side, then providing the input sequence as-is as input to the first layer and providing a reversed copy of the input sequence to the second.

Open the artice...

Gated Recurrent Unit - GRU Citing Wikipedia

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al. The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate.GRU's performance on certain tasks of polyphonic music modeling, speech signal modeling and natural language processing was found to be similar to that of LSTM. GRUs have been shown to exhibit better performance on certain smaller and less frequent datasets

We will define a few sample configurations and a a function `build_recurrent_model` that will build a model inserting a recurrent layer passed as a parameter

In [19]:
```
from keras.layers import LSTM
from keras.layers import Bidirectional
from keras.layers import GRU

configurations=[
    {'recurrent_layer':SimpleRNN(units = 20,return_sequences=True),'embed
    {'recurrent_layer':LSTM(units = 20,return_sequences=True),'embedding_
    {'recurrent_layer':Bidirectional(LSTM(units = 8,return_sequences=True
    {'recurrent_layer':GRU(units = 40,return_sequences=True),'embedding_d
]

def build_recurrent_model(embedding_dim, recurrent_layer):
  model= Sequential()
  model.add(Embedding(input_dim=10_000, output_dim=embedding_dim, input_l
  model.add(recurrent_layer)
  model.add(GlobalMaxPool1D())
  model.add(Dense(20, activation="relu"))
  model.add(Dropout(0.05))
  model.add(Dense(2, activation='softmax'))
  # model.add(Dense(1, activation='sigmoid'))
  model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentrop
  # model.compile(optimizer='rmsprop', loss='binary_crossentropy', metric
  model.summary()
  return model
```

TODO 7.3.1 Prepare at least four similar configurations using `SimpleRNN` , `LSTM` , `Bidirectional` and `GRU` layers. Each of basic configurations should be tested (or cross-validated). As a rule - you may use a configuration with smaller number of epochs for CV (cross-validation) and copy it entering greater number of epochs (for tetsing).

7.3.1 Using scikit learn compatiblie keras wrapper

We will try KerasClassifier, which can be used within scikit-learn pipelines and functions for hyperparameter selection (e.g. `GridSearchCV`)

In [17]:
```
from keras.wrappers.scikit_learn import KerasClassifier
#or
#!pip install scikeras tensorflow
#from scikeras.wrappers import KerasClassifier
```

In [20]:
```
config = configurations[2]

classifier = KerasClassifier(build_fn = lambda : build_recurrent_model(
    embedding_dim=config['embedding_dim'],
    recurrent_layer=config['recurrent_layer']),
    epochs=config['epochs'],
    batch_size=256)

classifier.fit(seq_train_padded,df_train.label)
y_pred = classifier.predict(seq_test_padded)
print(classification_report(df_test.label,y_pred))
```

```
/var/folders/db/mk99b90s0f95rb60mwwyfc9m0000gq/T/ipykernel_84877/3045942
352.py:3: DeprecationWarning: KerasClassifier is deprecated, use Sci-Ker
as (https://github.com/adriangb/scikeras) instead. See https://www.adria
ngb.com/scikeras/stable/migration.html for help migrating.
  classifier = KerasClassifier(build_fn = lambda : build_recurrent_model
(
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 500, 100)          1000000   
                                                                 
 bidirectional_1 (Bidirectio  (None, 500, 16)          6976      
 nal)                                                            
                                                                 
 global_max_pooling1d_2 (Glo  (None, 16)               0         
 balMaxPooling1D)                                                
                                                                 
 dense_4 (Dense)             (None, 20)                340       
                                                                 
 dropout_2 (Dropout)         (None, 20)                0         
                                                                 
 dense_5 (Dense)             (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,007,358
Trainable params: 1,007,358
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
98/98 [==============================] - 28s 262ms/step - loss: 0.6114 - 
accuracy: 0.6996
Epoch 2/3
98/98 [==============================] - 26s 261ms/step - loss: 0.3088 - 
accuracy: 0.8768
Epoch 3/3
98/98 [==============================] - 26s 264ms/step - loss: 0.2218 - 
accuracy: 0.9154
782/782 [==============================] - 13s 16ms/step
              precision    recall  f1-score   support

           0       0.88      0.88      0.88     12500
           1       0.88      0.88      0.88     12500

    accuracy                           0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000
```

7.3.2 Applying cross-validation

We will demonstrate cross-vaildation using `cross_vaildate` function.

TODO 7.3.2 Apply CV for 4 basic configurations with a small number of epochs (e.g.3) and number of folds (also 3). Use training dataset in cross validation.

In [21]:
```
import sklearn
sorted(sklearn.metrics.SCORERS.keys())
```

Out[21]:
```
['accuracy',
 'adjusted_mutual_info_score',
 'adjusted_rand_score',
 'average_precision',
 'balanced_accuracy',
 'completeness_score',
 'explained_variance',
 'f1',
 'f1_macro',
 'f1_micro',
 'f1_samples',
 'f1_weighted',
 'fowlkes_mallows_score',
 'homogeneity_score',
 'jaccard',
 'jaccard_macro',
 'jaccard_micro',
 'jaccard_samples',
 'jaccard_weighted',
 'matthews_corrcoef',
 'max_error',
 'mutual_info_score',
 'neg_brier_score',
 'neg_log_loss',
 'neg_mean_absolute_error',
 'neg_mean_absolute_percentage_error',
 'neg_mean_gamma_deviance',
 'neg_mean_poisson_deviance',
 'neg_mean_squared_error',
 'neg_mean_squared_log_error',
 'neg_median_absolute_error',
 'neg_negative_likelihood_ratio',
 'neg_root_mean_squared_error',
 'normalized_mutual_info_score',
 'positive_likelihood_ratio',
 'precision',
 'precision_macro',
 'precision_micro',
 'precision_samples',
 'precision_weighted',
 'r2',
 'rand_score',
 'recall',
 'recall_macro',
 'recall_micro',
 'recall_samples',
 'recall_weighted',
 'roc_auc',
 'roc_auc_ovo',
 'roc_auc_ovo_weighted',
 'roc_auc_ovr',
 'roc_auc_ovr_weighted',
 'top_k_accuracy',
 'v_measure_score']
```

In [107…
```
from sklearn.model_selection import cross_validate

config = configurations[0]

classifier = KerasClassifier(build_fn = lambda : build_recurrent_model(
```

```
    embedding_dim=config['embedding_dim'],
    recurrent_layer=config['recurrent_layer']),
    epochs=config['epochs'],
    batch_size=256)

n_cv_folds = 3
scoring = ['accuracy','precision_macro','recall_macro','f1_macro','roc_au
results = cross_validate(classifier, seq_train_padded,df_train.label,
                        cv=n_cv_folds, scoring=scoring,return_train_score
```

```
Model: "sequential_28"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_21 (Embedding)    (None, 500, 8)            80000     
                                                                 
 simple_rnn_2 (SimpleRNN)    (None, 500, 20)           580       
                                                                 
 global_max_pooling1d_21 (Gl  (None, 20)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_55 (Dense)            (None, 20)                420       
                                                                 
 dropout_19 (Dropout)        (None, 20)                0         
                                                                 
 dense_56 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 81,042
Trainable params: 81,042
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
```

```
/var/folders/db/mk99b90s0f95rb60mwwyfc9m0000gq/T/ipykernel_84877/3030849
369.py:5: DeprecationWarning: KerasClassifier is deprecated, use Sci-Ker
as (https://github.com/adriangb/scikeras) instead. See https://www.adria
ngb.com/scikeras/stable/migration.html for help migrating.
  classifier = KerasClassifier(build_fn = lambda : build_recurrent_model
(
```

```
66/66 [==============================] - 4s 52ms/step - loss: 0.6093 - a
ccuracy: 0.6783
Epoch 2/10
66/66 [==============================] - 3s 52ms/step - loss: 0.3962 - a
ccuracy: 0.8429
Epoch 3/10
66/66 [==============================] - 3s 52ms/step - loss: 0.2970 - a
ccuracy: 0.8868
Epoch 4/10
66/66 [==============================] - 3s 52ms/step - loss: 0.2338 - a
ccuracy: 0.9123
Epoch 5/10
66/66 [==============================] - 3s 52ms/step - loss: 0.1888 - a
ccuracy: 0.9317
Epoch 6/10
66/66 [==============================] - 3s 52ms/step - loss: 0.1578 - a
ccuracy: 0.9459
Epoch 7/10
66/66 [==============================] - 3s 52ms/step - loss: 0.1299 - a
ccuracy: 0.9544
Epoch 8/10
66/66 [==============================] - 3s 52ms/step - loss: 0.1113 - a
ccuracy: 0.9618
Epoch 9/10
66/66 [==============================] - 3s 52ms/step - loss: 0.0911 - a
ccuracy: 0.9702
Epoch 10/10
66/66 [==============================] - 3s 52ms/step - loss: 0.0823 - a
ccuracy: 0.9745
261/261 [==============================] - 2s 6ms/step
261/261 [==============================] - 2s 6ms/step
521/521 [==============================] - 3s 6ms/step
521/521 [==============================] - 3s 6ms/step
Model: "sequential_29"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_22 (Embedding)    (None, 500, 8)            80000     
                                                                 
 simple_rnn_2 (SimpleRNN)    (None, 500, 20)           580       
                                                                 
 global_max_pooling1d_22 (Gl  (None, 20)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_57 (Dense)            (None, 20)                420       
                                                                 
 dropout_20 (Dropout)        (None, 20)                0         
                                                                 
 dense_58 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 81,042
Trainable params: 81,042
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
66/66 [==============================] - 5s 54ms/step - loss: 0.6253 - a
ccuracy: 0.6460
Epoch 2/10
66/66 [==============================] - 4s 54ms/step - loss: 0.4167 - a
```

```
ccuracy: 0.8318
Epoch 3/10
66/66 [==============================] - 4s 53ms/step - loss: 0.3015 - a
ccuracy: 0.8838
Epoch 4/10
66/66 [==============================] - 3s 53ms/step - loss: 0.2389 - a
ccuracy: 0.9110
Epoch 5/10
66/66 [==============================] - 3s 53ms/step - loss: 0.1872 - a
ccuracy: 0.9345
Epoch 6/10
66/66 [==============================] - 4s 53ms/step - loss: 0.1565 - a
ccuracy: 0.9473
Epoch 7/10
66/66 [==============================] - 3s 52ms/step - loss: 0.1321 - a
ccuracy: 0.9572
Epoch 8/10
66/66 [==============================] - 3s 53ms/step - loss: 0.1135 - a
ccuracy: 0.9644
Epoch 9/10
66/66 [==============================] - 4s 54ms/step - loss: 0.0920 - a
ccuracy: 0.9719
Epoch 10/10
66/66 [==============================] - 3s 53ms/step - loss: 0.0798 - a
ccuracy: 0.9765
261/261 [==============================] - 2s 6ms/step
261/261 [==============================] - 2s 6ms/step
521/521 [==============================] - 3s 6ms/step
521/521 [==============================] - 4s 7ms/step
Model: "sequential_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_23 (Embedding)    (None, 500, 8)            80000

 simple_rnn_2 (SimpleRNN)    (None, 500, 20)           580

 global_max_pooling1d_23 (Gl  (None, 20)               0
 obalMaxPooling1D)

 dense_59 (Dense)            (None, 20)                420

 dropout_21 (Dropout)        (None, 20)                0

 dense_60 (Dense)            (None, 2)                 42

=================================================================
Total params: 81,042
Trainable params: 81,042
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
66/66 [==============================] - 4s 53ms/step - loss: 0.5661 - a
ccuracy: 0.7022
Epoch 2/10
66/66 [==============================] - 4s 54ms/step - loss: 0.3717 - a
ccuracy: 0.8477
Epoch 3/10
66/66 [==============================] - 4s 55ms/step - loss: 0.2646 - a
ccuracy: 0.9008
```

```
Epoch 4/10
66/66 [==============================] - 4s 53ms/step - loss: 0.2073 - a
ccuracy: 0.9263
Epoch 5/10
66/66 [==============================] - 4s 54ms/step - loss: 0.1671 - a
ccuracy: 0.9427
Epoch 6/10
66/66 [==============================] - 4s 54ms/step - loss: 0.1304 - a
ccuracy: 0.9563
Epoch 7/10
66/66 [==============================] - 3s 52ms/step - loss: 0.1050 - a
ccuracy: 0.9663
Epoch 8/10
66/66 [==============================] - 3s 52ms/step - loss: 0.0840 - a
ccuracy: 0.9738
Epoch 9/10
66/66 [==============================] - 3s 52ms/step - loss: 0.0723 - a
ccuracy: 0.9776
Epoch 10/10
66/66 [==============================] - 3s 52ms/step - loss: 0.0554 - a
ccuracy: 0.9842
261/261 [==============================] - 2s 7ms/step
261/261 [==============================] - 2s 6ms/step
521/521 [==============================] - 4s 7ms/step
521/521 [==============================] - 3s 6ms/step
fit_time                 35.691582
score_time                3.534172
test_accuracy             0.855040
train_accuracy            0.991620
test_precision_macro      0.855443
train_precision_macro     0.991628
test_recall_macro         0.855106
train_recall_macro        0.991617
test_f1_macro             0.855011
train_f1_macro            0.991620
test_roc_auc              0.931138
train_roc_auc             0.997963
dtype: float64
-------------
```

In [109…

```
df_results = pd.DataFrame(results)
df_mean = df_results.mean()
print(df_mean)
```


```
fit_time                 35.691582
score_time                3.534172
test_accuracy             0.855040
train_accuracy            0.991620
test_precision_macro      0.855443
train_precision_macro     0.991628
test_recall_macro         0.855106
train_recall_macro        0.991617
test_f1_macro             0.855011
train_f1_macro            0.991620
test_roc_auc              0.931138
train_roc_auc             0.997963
dtype: float64
```

In [110…

```
config = configurations[1]

classifier = KerasClassifier(build_fn = lambda : build_recurrent_model(
    embedding_dim=config['embedding_dim'],
    recurrent_layer=config['recurrent_layer']),
    epochs=config['epochs'],
    batch_size=256)

n_cv_folds = 3
scoring = ['accuracy','precision_macro','recall_macro','f1_macro','roc_au
results = cross_validate(classifier, seq_train_padded,df_train.label,
                        cv=n_cv_folds, scoring=scoring,return_train_score
```

```
Model: "sequential_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_24 (Embedding)    (None, 500, 100)          1000000   
                                                                 
 lstm_2 (LSTM)               (None, 500, 20)           9680      
                                                                 
 global_max_pooling1d_24 (Gl  (None, 20)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_61 (Dense)            (None, 20)                420       
                                                                 
 dropout_22 (Dropout)        (None, 20)                0         
                                                                 
 dense_62 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,010,142
Trainable params: 1,010,142
Non-trainable params: 0
_________________________________________________________________
```

```
/var/folders/db/mk99b90s0f95rb60mwwyfc9m0000gq/T/ipykernel_84877/3799207
936.py:3: DeprecationWarning: KerasClassifier is deprecated, use Sci-Ker
as (https://github.com/adriangb/scikeras) instead. See https://www.adria
ngb.com/scikeras/stable/migration.html for help migrating.
  classifier = KerasClassifier(build_fn = lambda : build_recurrent_model
(
```

```
Epoch 1/10
66/66 [==============================] - 22s 318ms/step - loss: 0.5017 -
accuracy: 0.7629
Epoch 2/10
66/66 [==============================] - 19s 291ms/step - loss: 0.2857 -
accuracy: 0.8890
Epoch 3/10
66/66 [==============================] - 19s 292ms/step - loss: 0.2119 -
accuracy: 0.9210
Epoch 4/10
66/66 [==============================] - 19s 293ms/step - loss: 0.1631 -
accuracy: 0.9429
Epoch 5/10
66/66 [==============================] - 19s 294ms/step - loss: 0.1263 -
accuracy: 0.9590
Epoch 6/10
66/66 [==============================] - 20s 301ms/step - loss: 0.1018 -
accuracy: 0.9677
Epoch 7/10
66/66 [==============================] - 19s 287ms/step - loss: 0.0812 -
accuracy: 0.9755
Epoch 8/10
66/66 [==============================] - 19s 286ms/step - loss: 0.0681 -
accuracy: 0.9800
Epoch 9/10
66/66 [==============================] - 20s 296ms/step - loss: 0.0538 -
accuracy: 0.9849
Epoch 10/10
66/66 [==============================] - 20s 298ms/step - loss: 0.0417 -
accuracy: 0.9881
261/261 [==============================] - 4s 16ms/step
261/261 [==============================] - 4s 16ms/step
521/521 [==============================] - 9s 17ms/step
521/521 [==============================] - 8s 16ms/step
Model: "sequential_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_25 (Embedding)    (None, 500, 100)          1000000

 lstm_2 (LSTM)               (None, 500, 20)           9680

 global_max_pooling1d_25 (Gl  (None, 20)               0
 obalMaxPooling1D)

 dense_63 (Dense)            (None, 20)                420

 dropout_23 (Dropout)        (None, 20)                0

 dense_64 (Dense)            (None, 2)                 42

=================================================================
Total params: 1,010,142
Trainable params: 1,010,142
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
66/66 [==============================] - 22s 308ms/step - loss: 0.6366 -
accuracy: 0.6226
Epoch 2/10
```

```
66/66 [==============================] - 20s 303ms/step - loss: 0.3830 -
accuracy: 0.8514
Epoch 3/10
66/66 [==============================] - 20s 302ms/step - loss: 0.2526 -
accuracy: 0.9067
Epoch 4/10
66/66 [==============================] - 20s 304ms/step - loss: 0.1834 -
accuracy: 0.9363
Epoch 5/10
66/66 [==============================] - 20s 306ms/step - loss: 0.1372 -
accuracy: 0.9540
Epoch 6/10
66/66 [==============================] - 19s 292ms/step - loss: 0.1091 -
accuracy: 0.9651
Epoch 7/10
66/66 [==============================] - 20s 306ms/step - loss: 0.0839 -
accuracy: 0.9747
Epoch 8/10
66/66 [==============================] - 21s 320ms/step - loss: 0.0659 -
accuracy: 0.9810
Epoch 9/10
66/66 [==============================] - 21s 314ms/step - loss: 0.0499 -
accuracy: 0.9860
Epoch 10/10
66/66 [==============================] - 20s 311ms/step - loss: 0.0419 -
accuracy: 0.9888
261/261 [==============================] - 5s 17ms/step
261/261 [==============================] - 4s 17ms/step
521/521 [==============================] - 9s 16ms/step
521/521 [==============================] - 8s 16ms/step
Model: "sequential_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_26 (Embedding)    (None, 500, 100)          1000000   
                                                                 
 lstm_2 (LSTM)               (None, 500, 20)           9680      
                                                                 
 global_max_pooling1d_26 (Gl  (None, 20)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_65 (Dense)            (None, 20)                420       
                                                                 
 dropout_24 (Dropout)        (None, 20)                0         
                                                                 
 dense_66 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,010,142
Trainable params: 1,010,142
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
66/66 [==============================] - 20s 280ms/step - loss: 0.6144 -
accuracy: 0.6685
Epoch 2/10
66/66 [==============================] - 19s 287ms/step - loss: 0.4088 -
accuracy: 0.8374
Epoch 3/10
66/66 [==============================] - 18s 275ms/step - loss: 0.3006 -
```

```
accuracy: 0.8865
Epoch 4/10
66/66 [==============================] - 19s 293ms/step - loss: 0.2322 -
accuracy: 0.9164
Epoch 5/10
66/66 [==============================] - 19s 283ms/step - loss: 0.1804 -
accuracy: 0.9378
Epoch 6/10
66/66 [==============================] - 19s 288ms/step - loss: 0.1428 -
accuracy: 0.9534
Epoch 7/10
66/66 [==============================] - 20s 299ms/step - loss: 0.1110 -
accuracy: 0.9641
Epoch 8/10
66/66 [==============================] - 19s 289ms/step - loss: 0.0864 -
accuracy: 0.9741
Epoch 9/10
66/66 [==============================] - 19s 293ms/step - loss: 0.0664 -
accuracy: 0.9803
Epoch 10/10
66/66 [==============================] - 20s 303ms/step - loss: 0.0528 -
accuracy: 0.9848
261/261 [==============================] - 5s 17ms/step
261/261 [==============================] - 4s 17ms/step
521/521 [==============================] - 9s 17ms/step
521/521 [==============================] - 8s 16ms/step
```

In [111…

```
df_results = pd.DataFrame(results)
df_mean = df_results.mean()
print(df_mean)
```

```
fit_time                 197.578003
score_time                 8.937461
test_accuracy              0.845320
train_accuracy             0.987680
test_precision_macro       0.850614
train_precision_macro      0.987884
test_recall_macro          0.845260
train_recall_macro         0.987690
test_f1_macro              0.844686
train_f1_macro             0.987679
test_roc_auc               0.927810
train_roc_auc              0.998345
dtype: float64
```

In [112…

```
config = configurations[2]

classifier = KerasClassifier(build_fn = lambda : build_recurrent_model(
    embedding_dim=config['embedding_dim'],
    recurrent_layer=config['recurrent_layer']),
    epochs=config['epochs'],
    batch_size=256)

n_cv_folds = 3
scoring = ['accuracy','precision_macro','recall_macro','f1_macro','roc_au
results = cross_validate(classifier, seq_train_padded,df_train.label,
                        cv=n_cv_folds, scoring=scoring,return_train_score
df_results = pd.DataFrame(results)
df_mean = df_results.mean()
print(df_mean, "\n-------------\n\n")
```

```
/var/folders/db/mk99b90s0f95rb60mwwyfc9m0000gq/T/ipykernel_84877/4089919
225.py:3: DeprecationWarning: KerasClassifier is deprecated, use Sci-Ker
as (https://github.com/adriangb/scikeras) instead. See https://www.adria
ngb.com/scikeras/stable/migration.html for help migrating.
  classifier = KerasClassifier(build_fn = lambda : build_recurrent_model
(
```

```
Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_27 (Embedding)    (None, 500, 100)          1000000   
                                                                 
 bidirectional_1 (Bidirectio  (None, 500, 16)          6976      
 nal)                                                            
                                                                 
 global_max_pooling1d_27 (Gl  (None, 16)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_67 (Dense)            (None, 20)                340       
                                                                 
 dropout_25 (Dropout)        (None, 20)                0         
                                                                 
 dense_68 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,007,358
Trainable params: 1,007,358
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
66/66 [==============================] - 19s 243ms/step - loss: 0.5947 -
accuracy: 0.7069
Epoch 2/3
66/66 [==============================] - 16s 239ms/step - loss: 0.4637 -
accuracy: 0.8233
Epoch 3/3
66/66 [==============================] - 16s 246ms/step - loss: 0.3919 -
accuracy: 0.8610
261/261 [==============================] - 4s 15ms/step
261/261 [==============================] - 4s 15ms/step
521/521 [==============================] - 8s 14ms/step
521/521 [==============================] - 7s 14ms/step
Model: "sequential_35"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_28 (Embedding)    (None, 500, 100)          1000000   
                                                                 
 bidirectional_1 (Bidirectio  (None, 500, 16)          6976      
 nal)                                                            
                                                                 
 global_max_pooling1d_28 (Gl  (None, 16)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_69 (Dense)            (None, 20)                340       
                                                                 
 dropout_26 (Dropout)        (None, 20)                0         
                                                                 
 dense_70 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,007,358
Trainable params: 1,007,358
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
```

```
66/66 [==============================] - 17s 231ms/step - loss: 0.6022 -
accuracy: 0.6867
Epoch 2/3
66/66 [==============================] - 16s 240ms/step - loss: 0.3798 -
accuracy: 0.8595
Epoch 3/3
66/66 [==============================] - 16s 243ms/step - loss: 0.2658 -
accuracy: 0.9021
261/261 [==============================] - 4s 14ms/step
261/261 [==============================] - 4s 14ms/step
521/521 [==============================] - 7s 14ms/step
521/521 [==============================] - 7s 14ms/step
Model: "sequential_36"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_29 (Embedding)    (None, 500, 100)          1000000

 bidirectional_1 (Bidirectio  (None, 500, 16)          6976
 nal)

 global_max_pooling1d_29 (Gl  (None, 16)               0
 obalMaxPooling1D)

 dense_71 (Dense)            (None, 20)                340

 dropout_27 (Dropout)        (None, 20)                0

 dense_72 (Dense)            (None, 2)                 42

=================================================================
Total params: 1,007,358
Trainable params: 1,007,358
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
66/66 [==============================] - 18s 246ms/step - loss: 0.5258 -
accuracy: 0.7523
Epoch 2/3
66/66 [==============================] - 16s 239ms/step - loss: 0.2964 -
accuracy: 0.8940
Epoch 3/3
66/66 [==============================] - 16s 239ms/step - loss: 0.2117 -
accuracy: 0.9288
261/261 [==============================] - 4s 15ms/step
261/261 [==============================] - 4s 14ms/step
521/521 [==============================] - 7s 14ms/step
521/521 [==============================] - 7s 14ms/step
fit_time                 50.421065
score_time                8.113526
test_accuracy             0.822559
train_accuracy            0.876180
test_precision_macro      0.840712
train_precision_macro     0.890158
test_recall_macro         0.822043
train_recall_macro        0.876384
test_f1_macro             0.819648
train_f1_macro            0.874736
test_roc_auc              0.918331
train_roc_auc             0.960108
```

```
dtype: float64
-------------
```

In [113…

```
df_results = pd.DataFrame(results)
df_mean = df_results.mean()
print(df_mean)
```

```
fit_time                  50.421065
score_time                 8.113526
test_accuracy              0.822559
train_accuracy             0.876180
test_precision_macro       0.840712
train_precision_macro      0.890158
test_recall_macro          0.822043
train_recall_macro         0.876384
test_f1_macro              0.819648
train_f1_macro             0.874736
test_roc_auc               0.918331
train_roc_auc              0.960108
dtype: float64
```

In [114…

```
config = configurations[3]

classifier = KerasClassifier(build_fn = lambda : build_recurrent_model(
    embedding_dim=config['embedding_dim'],
    recurrent_layer=config['recurrent_layer']),
    epochs=config['epochs'],
    batch_size=256)

n_cv_folds = 3
scoring = ['accuracy','precision_macro','recall_macro','f1_macro','roc_au
results = cross_validate(classifier, seq_train_padded,df_train.label,
                        cv=n_cv_folds, scoring=scoring,return_train_score
df_results = pd.DataFrame(results)
df_mean = df_results.mean()
print(df_mean, "\n-------------\n\n")
```

```
/var/folders/db/mk99b90s0f95rb60mwwyfc9m0000gq/T/ipykernel_84877/6794986
52.py:3: DeprecationWarning: KerasClassifier is deprecated, use Sci-Kera
s (https://github.com/adriangb/scikeras) instead. See https://www.adrian
gb.com/scikeras/stable/migration.html for help migrating.
  classifier = KerasClassifier(build_fn = lambda : build_recurrent_model
(
```

```
Model: "sequential_37"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_30 (Embedding)    (None, 500, 100)          1000000   
                                                                 
 gru_1 (GRU)                 (None, 500, 40)           17040     
                                                                 
 global_max_pooling1d_30 (Gl  (None, 40)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_73 (Dense)            (None, 20)                820       
                                                                 
 dropout_28 (Dropout)        (None, 20)                0         
                                                                 
 dense_74 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,017,902
Trainable params: 1,017,902
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
66/66 [==============================] - 37s 535ms/step - loss: 0.6492 - 
accuracy: 0.6787
Epoch 2/3
66/66 [==============================] - 37s 557ms/step - loss: 0.4455 - 
accuracy: 0.8078
Epoch 3/3
66/66 [==============================] - 35s 525ms/step - loss: 0.3024 - 
accuracy: 0.8756
261/261 [==============================] - 6s 21ms/step
261/261 [==============================] - 5s 21ms/step
521/521 [==============================] - 11s 21ms/step
521/521 [==============================] - 11s 21ms/step
Model: "sequential_38"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_31 (Embedding)    (None, 500, 100)          1000000   
                                                                 
 gru_1 (GRU)                 (None, 500, 40)           17040     
                                                                 
 global_max_pooling1d_31 (Gl  (None, 40)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_75 (Dense)            (None, 20)                820       
                                                                 
 dropout_29 (Dropout)        (None, 20)                0         
                                                                 
 dense_76 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,017,902
Trainable params: 1,017,902
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
66/66 [==============================] - 35s 521ms/step - loss: 0.5785 - 
accuracy: 0.7061
```

```
Epoch 2/3
66/66 [==============================] - 36s 547ms/step - loss: 0.3477 -
accuracy: 0.8629
Epoch 3/3
66/66 [==============================] - 38s 570ms/step - loss: 0.2577 -
accuracy: 0.9013
261/261 [==============================] - 6s 21ms/step
261/261 [==============================] - 5s 21ms/step
521/521 [==============================] - 11s 21ms/step
521/521 [==============================] - 11s 21ms/step
Model: "sequential_39"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_32 (Embedding)    (None, 500, 100)          1000000   
                                                                 
 gru_1 (GRU)                 (None, 500, 40)           17040     
                                                                 
 global_max_pooling1d_32 (Gl  (None, 40)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_77 (Dense)            (None, 20)                820       
                                                                 
 dropout_30 (Dropout)        (None, 20)                0         
                                                                 
 dense_78 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,017,902
Trainable params: 1,017,902
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
66/66 [==============================] - 39s 574ms/step - loss: 0.5762 -
accuracy: 0.7109
Epoch 2/3
66/66 [==============================] - 37s 563ms/step - loss: 0.3716 -
accuracy: 0.8498
Epoch 3/3
66/66 [==============================] - 37s 563ms/step - loss: 0.2781 -
accuracy: 0.8942
261/261 [==============================] - 6s 21ms/step
261/261 [==============================] - 5s 21ms/step
521/521 [==============================] - 11s 21ms/step
521/521 [==============================] - 11s 21ms/step
fit_time                 110.479637
score_time                11.182815
test_accuracy              0.839399
train_accuracy             0.898480
test_precision_macro       0.856232
train_precision_macro      0.909356
test_recall_macro          0.839242
train_recall_macro         0.898543
test_f1_macro              0.837125
train_f1_macro             0.897651
test_roc_auc               0.939744
train_roc_auc              0.976934
dtype: float64
-------------
```

In [115…
```
df_results = pd.DataFrame(results)
df_mean = df_results.mean()
print(df_mean)
```

```
fit_time                 110.479637
score_time                11.182815
test_accuracy              0.839399
train_accuracy             0.898480
test_precision_macro       0.856232
train_precision_macro      0.909356
test_recall_macro          0.839242
train_recall_macro         0.898543
test_f1_macro              0.837125
train_f1_macro             0.897651
test_roc_auc               0.939744
train_roc_auc              0.976934
dtype: float64
```

TODO 7.3.3 For each configuration tested - collect scores, compute mean values and finaly prepare a table summarizing the results. Expected four rows (results for a configuration in one row).

	fit_time	score_time	test_accuracy	train_accuracy	test_precision_macro	tra
Model 0	35.691582	3.534172	0.855040	0.991620	0.855443	0.9
Model 1	197.578003	8.937461	0.845320	0.987680	0.850614	0.9
Model 2	50.421065	8.113526	0.822559	0.876180	0.840712	0.8
Model 3	110.479637	11.182815	0.839399	0.898480	0.856232	0.9

TODO 7.3.4 Select two most promising configurations for testing (based on accuracy or F1). Apply more epochs. During thesting use the test dataset. Gather the results in a table (two rows), giving:

- name of the configuration
- recurrent network type
- number of epochs
- main scores

In [ ]:
```
config = configurations[0]

classifier = KerasClassifier(build_fn = lambda : build_recurrent_model(
    embedding_dim=config['embedding_dim'],
    recurrent_layer=config['recurrent_layer']),
    epochs=config['epochs'],
    batch_size=256)

n_cv_folds = 3
scoring = ['accuracy','precision_macro','recall_macro','f1_macro','roc_au
results = cross_validate(classifier, seq_train_padded,df_train.label,
```

```
                        cv=n_cv_folds, scoring=scoring,return_train_score
df_results = pd.DataFrame(results)
df_mean = df_results.mean()
print(df_mean, "\n-------------\n\n")
```

In [ ]:
```
config = configurations[3]

classifier = KerasClassifier(build_fn = lambda : build_recurrent_model(
    embedding_dim=config['embedding_dim'],
    recurrent_layer=config['recurrent_layer']),
    epochs=config['epochs'],
    batch_size=256)

n_cv_folds = 3
scoring = ['accuracy','precision_macro','recall_macro','f1_macro','roc_au
results = cross_validate(classifier, seq_train_padded,df_train.label,
                        cv=n_cv_folds, scoring=scoring,return_train_score
df_results = pd.DataFrame(results)
df_mean = df_results.mean()
print(df_mean, "\n-------------\n\n")
```

	number of epochs	fit_time	score_time	test_accuracy	train_accuracy	test_preci
SimpleRNN	9	35.691582	3.534172	0.855040	0.991620	0.855443
Biderectional	3	110.479637	11.182815	0.839399	0.898480	0.856232

7.3.3 Conv1D - sequence procesing based on 1D convolutions

In [26]:
```
from keras.models import Sequential
from keras import layers
from tensorflow.keras.optimizers import RMSprop



model = Sequential()
model.add(layers.Embedding(input_dim=10_000, output_dim=128, input_length
model.add(layers.Conv1D(filters=32, kernel_size=7, activation='relu'))
model.add(layers.MaxPooling1D(pool_size=5))
model.add(layers.Conv1D(filters=32, kernel_size=7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.summary()

model.compile(optimizer=RMSprop(learning_rate=1e-4),
                                loss='binary_crossentropy',
                                metrics=['accuracy'])
hist = model.fit(seq_train_padded,df_train.label, epochs=20, batch_size=1
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 500, 128)          1280000

 conv1d_2 (Conv1D)           (None, 494, 32)           28704

 max_pooling1d_1 (MaxPooling  (None, 98, 32)           0
 1D)

 conv1d_3 (Conv1D)           (None, 92, 32)            7200

 global_max_pooling1d_10 (Gl  (None, 32)               0
 obalMaxPooling1D)

 dense_19 (Dense)            (None, 1)                 33

=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
157/157 [==============================] - 10s 58ms/step - loss: 0.7279
- accuracy: 0.5273 - val_loss: 0.6888 - val_accuracy: 0.5298
Epoch 2/20
157/157 [==============================] - 9s 57ms/step - loss: 0.6662 -
accuracy: 0.6780 - val_loss: 0.6555 - val_accuracy: 0.7018
Epoch 3/20
157/157 [==============================] - 9s 59ms/step - loss: 0.5948 -
accuracy: 0.7871 - val_loss: 0.5436 - val_accuracy: 0.7800
Epoch 4/20
157/157 [==============================] - 9s 59ms/step - loss: 0.4488 -
accuracy: 0.8278 - val_loss: 0.4391 - val_accuracy: 0.8224
Epoch 5/20
157/157 [==============================] - 10s 64ms/step - loss: 0.3699
- accuracy: 0.8661 - val_loss: 0.4193 - val_accuracy: 0.8444
Epoch 6/20
157/157 [==============================] - 9s 57ms/step - loss: 0.3207 -
accuracy: 0.8874 - val_loss: 0.4377 - val_accuracy: 0.8562
Epoch 7/20
157/157 [==============================] - 9s 59ms/step - loss: 0.2814 -
accuracy: 0.9036 - val_loss: 0.4592 - val_accuracy: 0.8632
Epoch 8/20
157/157 [==============================] - 9s 60ms/step - loss: 0.2519 -
accuracy: 0.9158 - val_loss: 0.5229 - val_accuracy: 0.8648
Epoch 9/20
157/157 [==============================] - 9s 60ms/step - loss: 0.2276 -
accuracy: 0.9260 - val_loss: 0.5098 - val_accuracy: 0.8704
Epoch 10/20
157/157 [==============================] - 9s 60ms/step - loss: 0.2099 -
accuracy: 0.9334 - val_loss: 0.5392 - val_accuracy: 0.8716
Epoch 11/20
157/157 [==============================] - 9s 59ms/step - loss: 0.1895 -
accuracy: 0.9416 - val_loss: 0.5690 - val_accuracy: 0.8716
Epoch 12/20
157/157 [==============================] - 10s 61ms/step - loss: 0.1740
- accuracy: 0.9491 - val_loss: 0.6050 - val_accuracy: 0.8708
Epoch 13/20
```

```
157/157 [==============================] - 9s 57ms/step - loss: 0.1592 -
accuracy: 0.9570 - val_loss: 0.7103 - val_accuracy: 0.8642
Epoch 14/20
157/157 [==============================] - 9s 57ms/step - loss: 0.1457 -
accuracy: 0.9633 - val_loss: 0.6752 - val_accuracy: 0.8682
Epoch 15/20
157/157 [==============================] - 9s 57ms/step - loss: 0.1318 -
accuracy: 0.9692 - val_loss: 0.7176 - val_accuracy: 0.8654
Epoch 16/20
157/157 [==============================] - 9s 57ms/step - loss: 0.1208 -
accuracy: 0.9758 - val_loss: 0.7512 - val_accuracy: 0.8650
Epoch 17/20
157/157 [==============================] - 9s 57ms/step - loss: 0.1102 -
accuracy: 0.9797 - val_loss: 0.8072 - val_accuracy: 0.8650
Epoch 18/20
157/157 [==============================] - 9s 58ms/step - loss: 0.1008 -
accuracy: 0.9844 - val_loss: 0.8529 - val_accuracy: 0.8598
Epoch 19/20
157/157 [==============================] - 9s 57ms/step - loss: 0.0927 -
accuracy: 0.9882 - val_loss: 0.9020 - val_accuracy: 0.8614
Epoch 20/20
157/157 [==============================] - 9s 57ms/step - loss: 0.0862 -
accuracy: 0.9906 - val_loss: 0.9356 - val_accuracy: 0.8584
```

TODO Compute and display scores for the test set

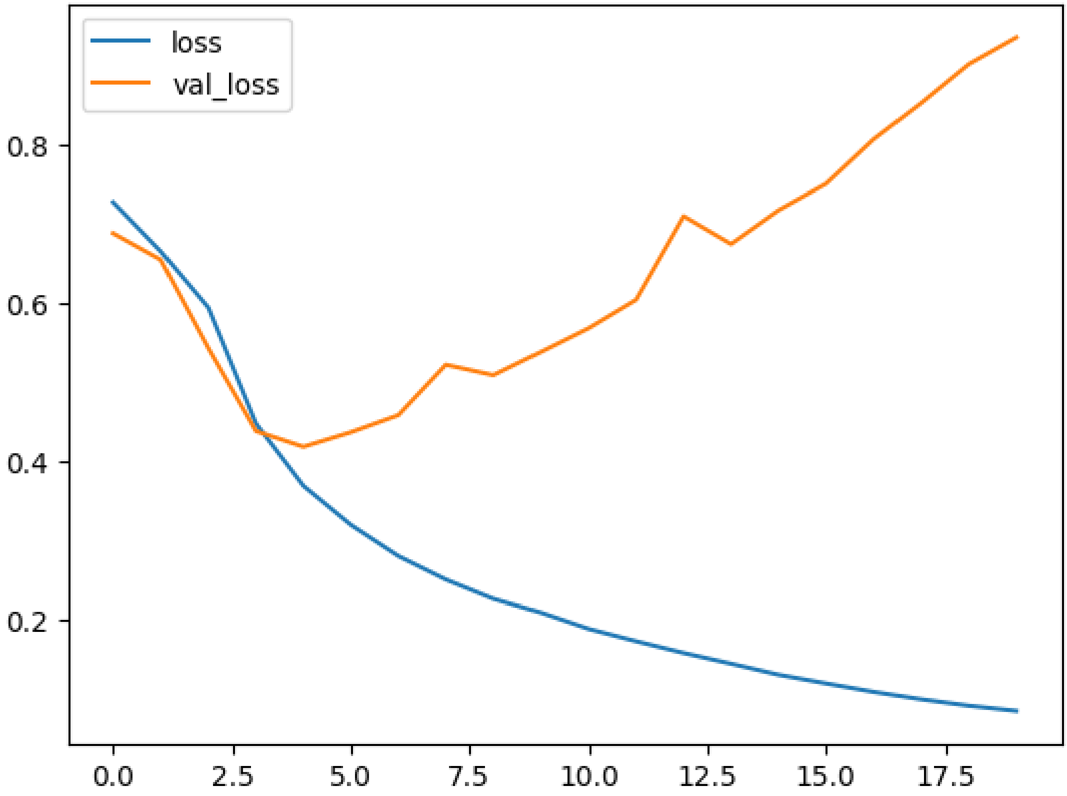
In [27]:
```
classifier.fit(seq_train_padded,df_train.label)
y_pred = classifier.predict(seq_test_padded)
print(classification_report(y_pred,df_test.label))
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_11 (Embedding)    (None, 500, 100)          1000000   
                                                                 
 bidirectional_1 (Bidirectio  (None, 500, 16)          6976      
 nal)                                                            
                                                                 
 global_max_pooling1d_11 (Gl  (None, 16)               0         
 obalMaxPooling1D)                                               
                                                                 
 dense_20 (Dense)            (None, 20)                340       
                                                                 
 dropout_9 (Dropout)         (None, 20)                0         
                                                                 
 dense_21 (Dense)            (None, 2)                 42        
                                                                 
=================================================================
Total params: 1,007,358
Trainable params: 1,007,358
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
98/98 [==============================] - 30s 283ms/step - loss: 0.5842 -
accuracy: 0.6901
Epoch 2/3
98/98 [==============================] - 27s 276ms/step - loss: 0.3785 -
accuracy: 0.8578
Epoch 3/3
98/98 [==============================] - 27s 277ms/step - loss: 0.3377 -
accuracy: 0.8860
782/782 [==============================] - 13s 16ms/step
              precision    recall  f1-score   support

           0       0.85      0.86      0.86     12329
           1       0.87      0.85      0.86     12671

    accuracy                           0.86     25000
   macro avg       0.86      0.86      0.86     25000
weighted avg       0.86      0.86      0.86     25000
```

TODO 7.3.5 Plot values in the history

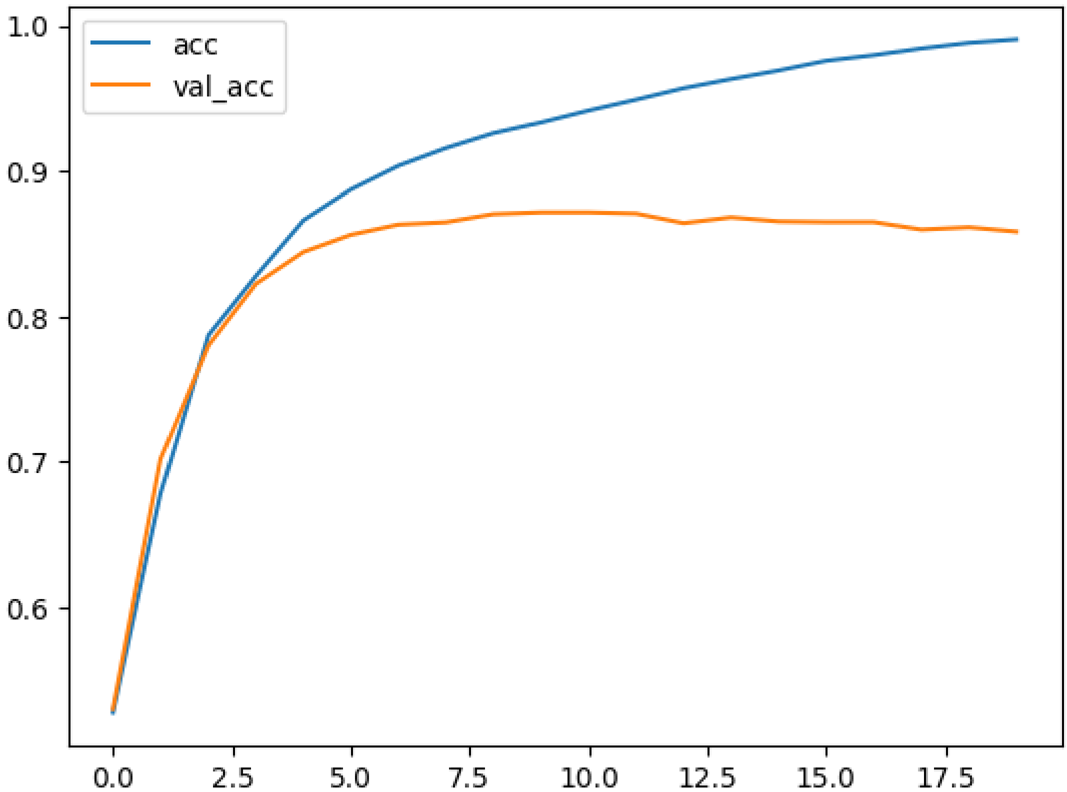
In [28]:
```
plt.plot(hist.history['loss'],label='loss')
plt.plot(hist.history['val_loss'],label='val_loss')
plt.legend()
```

Out[28]:
```
<matplotlib.legend.Legend at 0x30525b640>
```



In [29]:
```
plt.plot(hist.history['accuracy'],label='acc')
plt.plot(hist.history['val_accuracy'],label='val_acc')
plt.legend()
```

Out[29]: <matplotlib.legend.Legend at 0x2e76db1c0>



7.4 Regression - bike sharing dataset

We will use a dataset containing information on bike rentals. We will focus on the attribute `cnt` collected every hour.

In [61]:
```
!wget https://dysk.agh.edu.pl/s/G6ZNziBRbEEcMeN/download -O Bike-Sharing-
!unzip Bike-Sharing-Dataset.zip
!cat Readme.txt
```

```
--2023-04-20 18:28:17--  https://dysk.agh.edu.pl/s/G6ZNziBRbEEcMeN/downl
oad
Resolving dysk.agh.edu.pl (dysk.agh.edu.pl)... 2001:6d8:10:1060::6004, 1
49.156.96.4
Connecting to dysk.agh.edu.pl (dysk.agh.edu.pl)|2001:6d8:10:1060::6004|:
443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 279992 (273K) [application/zip]
Saving to: ‘Bike-Sharing-Dataset.zip’

Bike-Sharing-Datase 100%[===================>] 273,43K  --.-KB/s    in
0,05s

2023-04-20 18:28:18 (4,90 MB/s) - ‘Bike-Sharing-Dataset.zip’ saved [2799
92/279992]

Archive:  Bike-Sharing-Dataset.zip
replace Readme.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: ^C
==========================================
Bike Sharing Dataset
==========================================

Hadi Fanaee-T

Laboratory of Artificial Intelligence and Decision Support (LIAAD), Univ
ersity of Porto
INESC Porto, Campus da FEUP
Rua Dr. Roberto Frias, 378
4200 - 465 Porto, Portugal


=========================================
Background
=========================================

Bike sharing systems are new generation of traditional bike rentals wher
e whole process from membership, rental and return
back has become automatic. Through these systems, user is able to easily
rent a bike from a particular position and return
back at another position. Currently, there are about over 500 bike-shari
ng programs around the world which is composed of
over 500 thousands bicycles. Today, there exists great interest in these
systems due to their important role in traffic,
environmental and health issues.

Apart from interesting real world applications of bike sharing systems,
the characteristics of data being generated by
these systems make them attractive for the research. Opposed to other tr
ansport services such as bus or subway, the duration
of travel, departure and arrival position is explicitly recorded in thes
e systems. This feature turns bike sharing system into
a virtual sensor network that can be used for sensing mobility in the ci
ty. Hence, it is expected that most of important
events in the city could be detected via monitoring these data.

=========================================
Data Set
=========================================
Bike-sharing rental process is highly correlated to the environmental an
d seasonal settings. For instance, weather conditions,
```

```
precipitation, day of week, season, hour of the day, etc. can affect the
rental behaviors. The core data set is related to
the two-year historical log corresponding to years 2011 and 2012 from Ca
pital Bikeshare system, Washington D.C., USA which is
publicly available in http://capitalbikeshare.com/system-data. We aggreg
ated the data on two hourly and daily basis and then
extracted and added the corresponding weather and seasonal information.
Weather information are extracted from http://www.freemeteo.com.

=========================================
Associated tasks
=========================================

        - Regression:
                Predication of bike rental count hourly or daily based o
n the environmental and seasonal settings.

        - Event and Anomaly Detection:
                Count of rented bikes are also correlated to some events
in the town which easily are traceable via search engines.
                For instance, query like "2012-10-30 washington d.c." in
Google returns related results to Hurricane Sandy. Some of the important
events are
                identified in [1]. Therefore the data can be used for va
lidation of anomaly or event detection algorithms as well.


=========================================
Files
=========================================

        - Readme.txt
        - hour.csv : bike sharing counts aggregated on hourly basis. Rec
ords: 17379 hours
        - day.csv - bike sharing counts aggregated on daily basis. Recor
ds: 731 days


=========================================
Dataset characteristics
=========================================
Both hour.csv and day.csv have the following fields, except hr which is
not available in day.csv

        - instant: record index
        - dteday : date
        - season : season (1:springer, 2:summer, 3:fall, 4:winter)
        - yr : year (0: 2011, 1:2012)
        - mnth : month ( 1 to 12)
        - hr : hour (0 to 23)
        - holiday : weather day is holiday or not (extracted from htt
p://dchr.dc.gov/page/holiday-schedule)
        - weekday : day of the week
        - workingday : if day is neither weekend nor holiday is 1, other
wise is 0.
        + weathersit :
                - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
                - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clo
uds, Mist
                - 3: Light Snow, Light Rain + Thunderstorm + Scattered c
```

```
louds, Light Rain + Scattered clouds
                - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Sno
w + Fog
        - temp : Normalized temperature in Celsius. The values are divid
ed to 41 (max)
        - atemp: Normalized feeling temperature in Celsius. The values a
re divided to 50 (max)
        - hum: Normalized humidity. The values are divided to 100 (max)
        - windspeed: Normalized wind speed. The values are divided to 67
(max)
        - casual: count of casual users
        - registered: count of registered users
        - cnt: count of total rental bikes including both casual and reg
istered

=========================================
License
=========================================
Use of this dataset in publications must be cited to the following publi
cation:

[1] Fanaee-T, Hadi, and Gama, Joao, "Event labeling combining ensemble d
etectors and background knowledge", Progress in Artificial Intelligence
(2013): pp. 1-15, Springer Berlin Heidelberg, doi:10.1007/s13748-013-004
0-3.

@article{
        year={2013},
        issn={2192-6352},
        journal={Progress in Artificial Intelligence},
        doi={10.1007/s13748-013-0040-3},
        title={Event labeling combining ensemble detectors and backgroun
d knowledge},
        url={http://dx.doi.org/10.1007/s13748-013-0040-3},
        publisher={Springer Berlin Heidelberg},
        keywords={Event labeling; Event detection; Ensemble learning; Ba
ckground knowledge},
        author={Fanaee-T, Hadi and Gama, Joao},
        pages={1-15}
}

=========================================
Contact
=========================================

For further information about this dataset please contact Hadi Fanaee-T
(hadi.fanaee@fe.up.pt)
```

In [62]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('hour.csv',parse_dates=['dteday'])
df.head(len(df))
```

Out[62]:

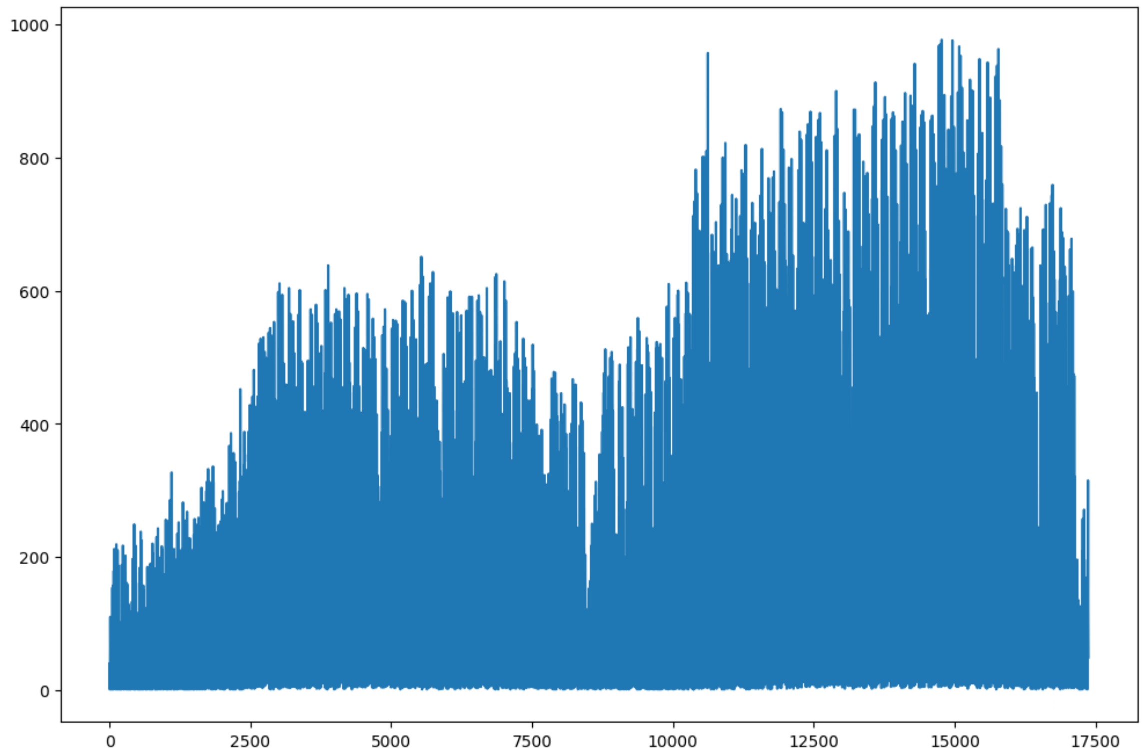
	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersi
0	1	2011-01-01	1	0	1	0	0	6	0	
1	2	2011-01-01	1	0	1	1	0	6	0	
2	3	2011-01-01	1	0	1	2	0	6	0	
3	4	2011-01-01	1	0	1	3	0	6	0	
4	5	2011-01-01	1	0	1	4	0	6	0	
...	...	...	...	...	...	...	...	...	...	...
17374	17375	2012-12-31	1	1	12	19	0	1	1	
17375	17376	2012-12-31	1	1	12	20	0	1	1	
17376	17377	2012-12-31	1	1	12	21	0	1	1	
17377	17378	2012-12-31	1	1	12	22	0	1	1	
17378	17379	2012-12-31	1	1	12	23	0	1	1	

17379 rows × 17 columns

In [63]:

```
plt.rcParams["figure.figsize"] = (12,8)
plt.plot(df.cnt)
```

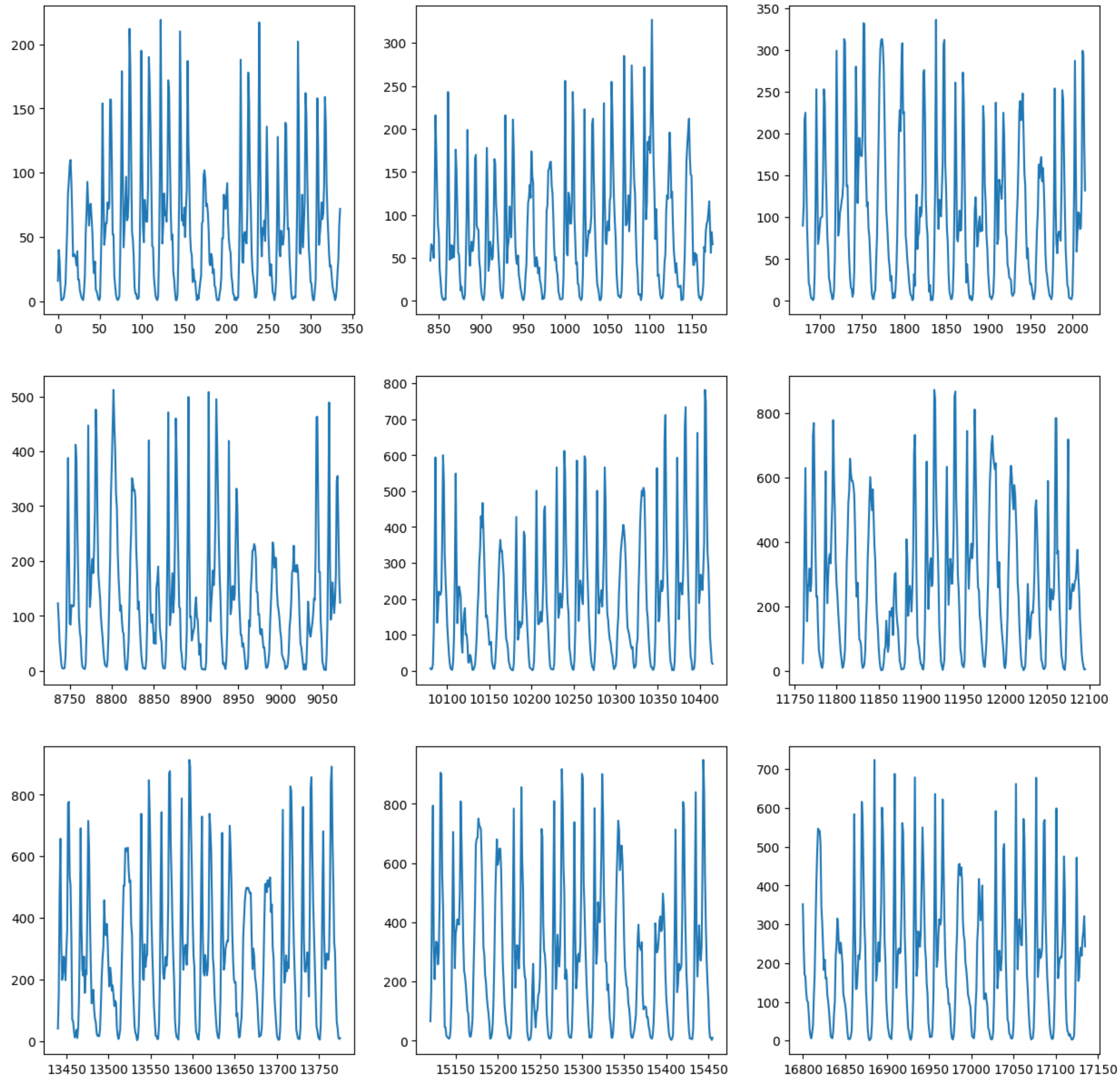
Out[63]: [<matplotlib.lines.Line2D at 0x2ee6fc040>]



Let us plot some examples for time intervals corresponding to two weeks. Visible fluctuations with 7 days and 24 periods.

In [64]:

```
import numpy as np

starts = np.array([0,5*7*24,10*7*24, 52*7*24, 60*7*24,70*7*24,80*7*24,90*
subplots = [df.cnt[starts[i]:starts[i]+14*24] for i in range(len(starts))

fig = plt.figure(figsize=(15, 15))
for i in range(9):
  ax = fig.add_subplot(3,3, 1 + i)
  ax.plot(subplots[i])
  # if title is not None:
  #   plt.suptitle(title)
plt.show()
```



Problem statement

We will use only a sequence of `cnt` values - let us dentoe it as $[x(0), x(1), \ldots, x(n)]$.

Our idea is to predict the target value in the feature based on a sequence registered recently in the past.

Let t be a current discrete time.

- $x(t+h)$ is a target value to be predicted. $h >= 1$ stands for *horizon*
- $[x(t-w), x(t-w+1), \ldots, x(t)]$ is a subsequence used for making predictions. It starts at $x(t-w)$ and ends at the current value $x(t)$. The parameter w stands for *window size*.

In our problem statement we select:

- h (horizon) equals to one week
- w (window size) equals to two weeks

So, we predict one week ahead.

7.4.1 Generator

The generator fiunction will yield a tuple `(samples, targets)` where `samples` is one batch of input data and `targets` is the corresponding array of target temperatures. It takes the following arguments:

- `data` : The original array of floating point data, which we just normalized in the code snippet above.
- `window_size` : How many timesteps back should our input data go.
- `horizon` : How many timesteps in the future should our target be.
- `min_index` and `max_index` : Indices in the `data` array that delimit which timesteps to draw from. This is useful for keeping a segment of the data for validation and another one for testing.
- `shuffle` : Whether to shuffle our samples or draw them in chronological order.
- `batch_size` : The number of samples per batch.
- `step` : The period, in timesteps, at which we sample data. We will set it 6 in order to draw one data point every hour.
- `expand_dims` wrap 2D arrays as 3D tensors.

In [65]:
```
import sklearn

def generator(data, window_size, horizon, min_index, max_index, shuffle=F
  if max_index is None:
    max_index = len(data) - 1
  cursor = min_index + window_size
  while 1:
      if cursor + batch_size + horizon > max_index:
          cursor = min_index + window_size

      rows = np.arange(cursor, min(cursor + batch_size, max_index))
      cursor += len(rows)

      samples = np.zeros((len(rows), window_size // step))
      targets = np.zeros((len(rows),))
      for j, row in enumerate(rows):
          indices = np.arange(rows[j] - window_size, rows[j], step,dtype=
          samples[j] = data[indices]
```

```
            targets[j] = data[rows[j] + horizon-1]
        if shuffle:
          samples,targets = sklearn.utils.shuffle(samples,targets)
        if expand_dims:
          samples = np.expand_dims(samples,axis=-1)
          # targets = np.expand_dims(targets,axis=-1)
        yield samples, targets
```

Some tests for demonstration

In [66]:
```
x=np.arange(3000)
gen = generator(x,10,10,min_index=0, max_index=70,batch_size = 5,shuffle=
for i,(x,y) in enumerate(gen):
  print(i,x,y)
  if i== 10: break
```

```
0 [[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
 [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
 [ 2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
 [ 3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]
 [ 4.  5.  6.  7.  8.  9. 10. 11. 12. 13.]] [19. 20. 21. 22. 23.]
1 [[ 5.  6.  7.  8.  9. 10. 11. 12. 13. 14.]
 [ 6.  7.  8.  9. 10. 11. 12. 13. 14. 15.]
 [ 7.  8.  9. 10. 11. 12. 13. 14. 15. 16.]
 [ 8.  9. 10. 11. 12. 13. 14. 15. 16. 17.]
 [ 9. 10. 11. 12. 13. 14. 15. 16. 17. 18.]] [24. 25. 26. 27. 28.]
2 [[10. 11. 12. 13. 14. 15. 16. 17. 18. 19.]
 [11. 12. 13. 14. 15. 16. 17. 18. 19. 20.]
 [12. 13. 14. 15. 16. 17. 18. 19. 20. 21.]
 [13. 14. 15. 16. 17. 18. 19. 20. 21. 22.]
 [14. 15. 16. 17. 18. 19. 20. 21. 22. 23.]] [29. 30. 31. 32. 33.]
3 [[15. 16. 17. 18. 19. 20. 21. 22. 23. 24.]
 [16. 17. 18. 19. 20. 21. 22. 23. 24. 25.]
 [17. 18. 19. 20. 21. 22. 23. 24. 25. 26.]
 [18. 19. 20. 21. 22. 23. 24. 25. 26. 27.]
 [19. 20. 21. 22. 23. 24. 25. 26. 27. 28.]] [34. 35. 36. 37. 38.]
4 [[20. 21. 22. 23. 24. 25. 26. 27. 28. 29.]
 [21. 22. 23. 24. 25. 26. 27. 28. 29. 30.]
 [22. 23. 24. 25. 26. 27. 28. 29. 30. 31.]
 [23. 24. 25. 26. 27. 28. 29. 30. 31. 32.]
 [24. 25. 26. 27. 28. 29. 30. 31. 32. 33.]] [39. 40. 41. 42. 43.]
5 [[25. 26. 27. 28. 29. 30. 31. 32. 33. 34.]
 [26. 27. 28. 29. 30. 31. 32. 33. 34. 35.]
 [27. 28. 29. 30. 31. 32. 33. 34. 35. 36.]
 [28. 29. 30. 31. 32. 33. 34. 35. 36. 37.]
 [29. 30. 31. 32. 33. 34. 35. 36. 37. 38.]] [44. 45. 46. 47. 48.]
6 [[30. 31. 32. 33. 34. 35. 36. 37. 38. 39.]
 [31. 32. 33. 34. 35. 36. 37. 38. 39. 40.]
 [32. 33. 34. 35. 36. 37. 38. 39. 40. 41.]
 [33. 34. 35. 36. 37. 38. 39. 40. 41. 42.]
 [34. 35. 36. 37. 38. 39. 40. 41. 42. 43.]] [49. 50. 51. 52. 53.]
7 [[35. 36. 37. 38. 39. 40. 41. 42. 43. 44.]
 [36. 37. 38. 39. 40. 41. 42. 43. 44. 45.]
 [37. 38. 39. 40. 41. 42. 43. 44. 45. 46.]
 [38. 39. 40. 41. 42. 43. 44. 45. 46. 47.]
 [39. 40. 41. 42. 43. 44. 45. 46. 47. 48.]] [54. 55. 56. 57. 58.]
8 [[40. 41. 42. 43. 44. 45. 46. 47. 48. 49.]
 [41. 42. 43. 44. 45. 46. 47. 48. 49. 50.]
 [42. 43. 44. 45. 46. 47. 48. 49. 50. 51.]
 [43. 44. 45. 46. 47. 48. 49. 50. 51. 52.]
 [44. 45. 46. 47. 48. 49. 50. 51. 52. 53.]] [59. 60. 61. 62. 63.]
9 [[45. 46. 47. 48. 49. 50. 51. 52. 53. 54.]
 [46. 47. 48. 49. 50. 51. 52. 53. 54. 55.]
 [47. 48. 49. 50. 51. 52. 53. 54. 55. 56.]
 [48. 49. 50. 51. 52. 53. 54. 55. 56. 57.]
 [49. 50. 51. 52. 53. 54. 55. 56. 57. 58.]] [64. 65. 66. 67. 68.]
10 [[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
 [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
 [ 2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
 [ 3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]
 [ 4.  5.  6.  7.  8.  9. 10. 11. 12. 13.]] [19. 20. 21. 22. 23.]
```

Data preparation and configuration of generators

- We selected that an inital sequence o 12000 readings wil be used for training and the rest for testing
- Data are normalized with `StandardScaler`

Note in further experiments we will repat cells initializing the generators. They start at the first index and then they wrap over the end to the beginning.

In [67]:
```
x_orig = df.cnt.to_numpy()
from sklearn.preprocessing import StandardScaler
x=x_orig.reshape(-1, 1)
scaler = StandardScaler()
scaler.fit(x[:12_000])
x_scaled=scaler.transform(x)
x_scaled=x_scaled.reshape(-1)
print(x_scaled)
```

```
[-0.94185161 -0.78331641 -0.83616147 ... -0.45303472 -0.6445981
 -0.7238657 ]
```

In [68]:
```
window_size = 14*24
horizon = 7*24
batch_size=128
train_gen = generator(x_scaled,
                      window_size=window_size,
                      horizon=horizon,
                      min_index=0,
                      max_index=12_000,
                      shuffle=False,
                      step=1,
                      batch_size=batch_size)
val_gen = generator(x_scaled,
                     window_size=window_size,
                     horizon=horizon,
                     min_index=12_000,
                     max_index=None,
                     shuffle=False,
                     step=1,
                     batch_size=batch_size)

train_steps = (12_000-window_size-horizon-batch_size)//window_size
print(train_steps)

val_steps = (x_scaled.size-12_000-window_size-horizon-batch_size)//window
print(val_steps)
```

```
33
14
```

7.4.2 Basic architecture

The basic architecture uses the whole input sequence at once to make the prediction.

In [69]:
```
from keras.models import Sequential
from keras import layers
import tensorflow
from tensorflow.keras.optimizers import RMSprop
```

```
import tensorflow as tf

seed=42
np.random.seed(seed)
tf.random.set_seed(seed)


model = Sequential()
model.add(layers.Dense(units=64,input_shape=(window_size,)))
model.add(layers.Dense(units=32, activation='relu'))
model.add(layers.Dense(1))

# model .compile(optimizer=RMSprop(learning_rate=0.0001), loss='mse', met
model .compile(optimizer='adam', loss='mse', metrics=['mse','mae'])
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_29 (Dense)            (None, 64)                21568

 dense_30 (Dense)            (None, 32)                2080

 dense_31 (Dense)            (None, 1)                 33

=================================================================
Total params: 23,681
Trainable params: 23,681
Non-trainable params: 0
_________________________________________________________________
```

In [70]:

```
hist = model.fit(train_gen,steps_per_epoch=train_steps,epochs=20,batch_si
                                                validation_data=val_gen,
                                                validation_steps=val_step
```

```
Epoch 1/20
33/33 [==============================] - 1s 14ms/step - loss: 0.7914 - m
se: 0.7914 - mae: 0.6913 - val_loss: 2.4575 - val_mse: 2.4575 - val_mae:
1.2735
Epoch 2/20
33/33 [==============================] - 0s 1ms/step - loss: 0.8463 - ms
e: 0.8463 - mae: 0.7089 - val_loss: 8.9027 - val_mse: 8.9027 - val_mae:
2.5366
Epoch 3/20
33/33 [==============================] - 0s 1ms/step - loss: 0.9909 - ms
e: 0.9909 - mae: 0.7843 - val_loss: 2.1246 - val_mse: 2.1246 - val_mae:
1.1339
Epoch 4/20
33/33 [==============================] - 0s 1ms/step - loss: 0.5113 - ms
e: 0.5113 - mae: 0.5427 - val_loss: 5.8752 - val_mse: 5.8752 - val_mae:
1.9840
Epoch 5/20
33/33 [==============================] - 0s 1ms/step - loss: 0.6231 - ms
e: 0.6231 - mae: 0.6078 - val_loss: 3.4087 - val_mse: 3.4087 - val_mae:
1.4633
Epoch 6/20
33/33 [==============================] - 0s 1ms/step - loss: 0.6550 - ms
e: 0.6550 - mae: 0.6073 - val_loss: 4.0740 - val_mse: 4.0740 - val_mae:
1.7074
Epoch 7/20
33/33 [==============================] - 0s 1ms/step - loss: 0.5344 - ms
e: 0.5344 - mae: 0.5540 - val_loss: 2.8495 - val_mse: 2.8495 - val_mae:
1.3932
Epoch 8/20
33/33 [==============================] - 0s 1ms/step - loss: 0.5551 - ms
e: 0.5551 - mae: 0.5373 - val_loss: 0.8861 - val_mse: 0.8861 - val_mae:
0.7238
Epoch 9/20
33/33 [==============================] - 0s 1ms/step - loss: 0.3685 - ms
e: 0.3685 - mae: 0.4675 - val_loss: 2.8953 - val_mse: 2.8953 - val_mae:
1.5320
Epoch 10/20
33/33 [==============================] - 0s 1ms/step - loss: 0.4720 - ms
e: 0.4720 - mae: 0.5027 - val_loss: 3.4896 - val_mse: 3.4896 - val_mae:
1.4977
Epoch 11/20
33/33 [==============================] - 0s 1ms/step - loss: 0.5321 - ms
e: 0.5321 - mae: 0.5407 - val_loss: 3.9955 - val_mse: 3.9955 - val_mae:
1.7239
Epoch 12/20
33/33 [==============================] - 0s 1ms/step - loss: 0.3172 - ms
e: 0.3172 - mae: 0.4263 - val_loss: 6.0127 - val_mse: 6.0127 - val_mae:
2.2152
Epoch 13/20
33/33 [==============================] - 0s 1ms/step - loss: 0.5501 - ms
e: 0.5501 - mae: 0.5442 - val_loss: 4.0822 - val_mse: 4.0822 - val_mae:
1.7028
Epoch 14/20
33/33 [==============================] - 0s 1ms/step - loss: 0.6682 - ms
e: 0.6682 - mae: 0.6300 - val_loss: 4.5407 - val_mse: 4.5407 - val_mae:
1.8822
Epoch 15/20
33/33 [==============================] - 0s 1ms/step - loss: 0.4874 - ms
e: 0.4874 - mae: 0.5150 - val_loss: 4.8723 - val_mse: 4.8723 - val_mae:
1.7895
```

```
Epoch 16/20
33/33 [==============================] - 0s 1ms/step - loss: 0.5393 - ms
e: 0.5393 - mae: 0.5442 - val_loss: 0.8829 - val_mse: 0.8829 - val_mae:
0.7207
Epoch 17/20
33/33 [==============================] - 0s 1ms/step - loss: 0.4801 - ms
e: 0.4801 - mae: 0.5208 - val_loss: 4.7497 - val_mse: 4.7497 - val_mae:
2.0368
Epoch 18/20
33/33 [==============================] - 0s 2ms/step - loss: 0.4745 - ms
e: 0.4745 - mae: 0.5079 - val_loss: 3.8968 - val_mse: 3.8968 - val_mae:
1.5782
Epoch 19/20
33/33 [==============================] - 0s 2ms/step - loss: 0.6814 - ms
e: 0.6814 - mae: 0.6054 - val_loss: 2.4319 - val_mse: 2.4319 - val_mae:
1.2763
Epoch 20/20
33/33 [==============================] - 0s 2ms/step - loss: 0.3163 - ms
e: 0.3163 - mae: 0.4251 - val_loss: 3.1546 - val_mse: 3.1546 - val_mae:
1.4452
```

To test on the remaining data, we will configure a test generator, which produces batches containing one single element. The sizes of predicted values `y_pred` and test values `y_test` should match.

In [71]:
```
test_gen = generator(x_scaled,
                     window_size=window_size,
                     horizon=horizon,
                     min_index=12_000,
                     max_index=None,
                     shuffle=False,
                     step=1,
                     batch_size=1)

steps = len(x_scaled)-12_000

y_pred = model.predict(test_gen,steps=steps)
print(y_pred.shape)
y_pred = y_pred.reshape(-1)
print(y_pred.shape)
```

```
5379/5379 [==============================] - 2s 366us/step
(5379, 1)
(5379,)
```

In [72]:
```
y_test = x_scaled[12_000:]
print(y_test.shape)
# print(y_test)
```

```
(5379,)
```

We will compute some basic regression scores. Analyzing `r2` (determination coefficient) we state that predictive capabilities of our model are not that great.

In [73]:
```
import sklearn.metrics

def compute_scores(y_test,y_pred):
  scores={'r2':sklearn.metrics.r2_score,
        'mse':sklearn.metrics.mean_squared_error,
```

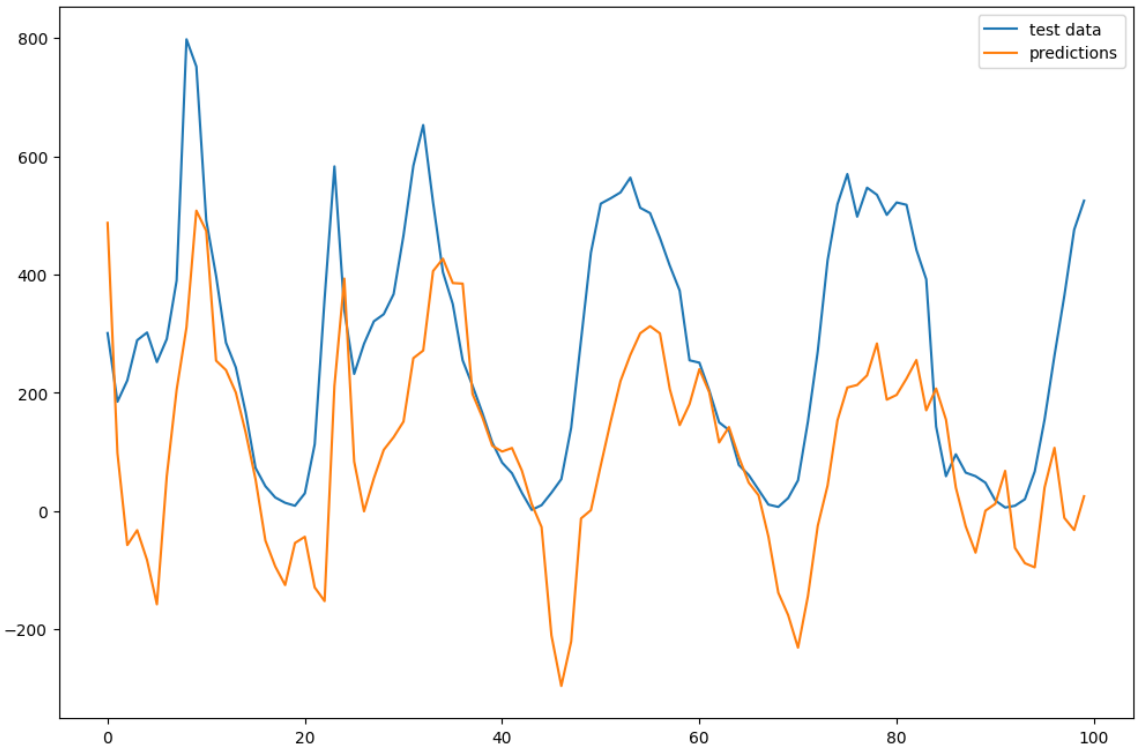
```
        'rmse':lambda y_true,y_pred : np.sqrt(sklearn.metrics.mean_square
        'maxe':sklearn.metrics.max_error,
        'med':sklearn.metrics.median_absolute_error,
        'mae':sklearn.metrics.mean_absolute_error,
        'mape':sklearn.metrics.mean_absolute_percentage_error,
        }
  results={}
  for k in scores:
    results[k] = scores[k](y_test,y_pred)
  return results

r = compute_scores(y_test,y_pred)
print(r)
```

```
{'r2': -0.42897834424032966, 'mse': 3.0113139999171836, 'rmse': 1.735313
8044507062, 'maxe': 6.197833985519388, 'med': 1.027468854636469, 'mae':
1.3318156623588446, 'mape': 3.848662361940473}
```

To prepare plots in original units, we must apply inverse transformation to that introduced by the scaler.

In [74]:
```
y_test_orig = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_orig = scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test_orig = y_test_orig.reshape(-1)
y_pred_orig = y_pred_orig.reshape(-1)
# print(y_test_orig.shape)
```

In [75]:
```
plt.plot(y_test_orig[100:200],label='test data')
plt.plot(y_pred_orig[100:200],label='predictions')
plt.legend()
plt.show()
```



7.4.3 GRU

We will apply a basic GRU configuration.

TODO 7.4.1 Make the model more complex, eg. increas a number of units and introduce a smal dense layer.

In [116…

```
from keras.models import Sequential
from keras import layers
#from keras.optimizers import RMSprop
import tensorflow
from tensorflow.keras.optimizers import RMSprop

model  = Sequential()

model.add(layers.GRU(units = 64, input_shape=(None,window_size)))
model.add(layers.Dense(8))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mse', metrics=['mse','mae'])
model.summary()
```

```
Model: "sequential_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_7 (GRU)                 (None, 64)                77184

 dense_79 (Dense)            (None, 8)                 520

 dense_80 (Dense)            (None, 1)                 9

=================================================================
Total params: 77,713
Trainable params: 77,713
Non-trainable params: 0
_________________________________________________________________
```

In this case we applied `expand_dims` option - to convert 2D matrix `batch_size x window_size` into `batch_size x window_size x features`. The number of features is 1.

In [100…

```
train_gen = generator(x_scaled,
                      window_size=window_size,
                      horizon=horizon,
                      min_index=0,
                      max_index=12_000,
                      shuffle=False,
                      step=1,
                      batch_size=batch_size,
                      expand_dims=True)
val_gen = generator(x_scaled,
                    window_size=window_size,
                    horizon=horizon,
                    min_index=12_000,
                    max_index=None,
                    shuffle=False,
                    step=1,
                    batch_size=batch_size,
                    expand_dims=True)
```

```
train_steps = (12_000-window_size-horizon-batch_size)//window_size
print(train_steps)

val_steps = (x_scaled.size-12_000-window_size-horizon-batch_size)//window
print(val_steps)
```

```
33
14
```

In [101…

```
X,y = next(train_gen)
print(X.shape)
```

```
(128, 336, 1)
```

In [102…

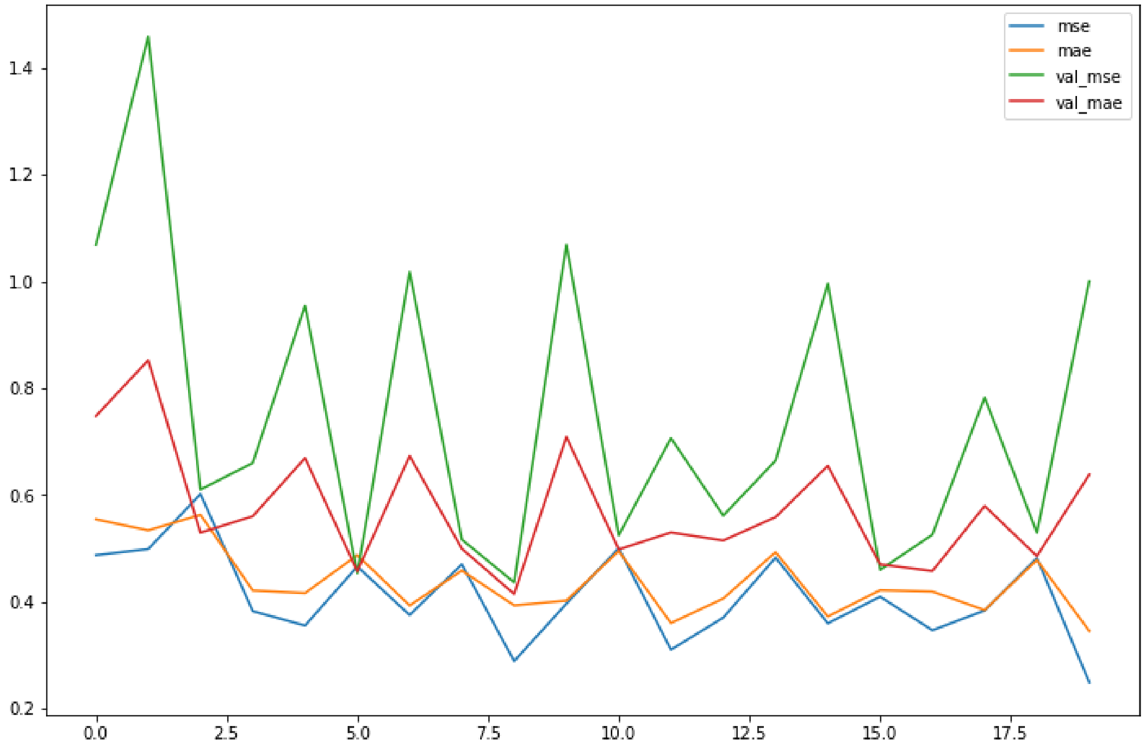
```
hist = model.fit(train_gen,steps_per_epoch=train_steps,
                                            epochs=20,
                                            batch_size=batch_size,
                                            validation_data=val_gen,
                                            validation_steps=val_steps)
```

```
Epoch 1/20
33/33 [==============================] - 1s 16ms/step - loss: 0.3556 - m
se: 0.2156 - mae: 0.3556 - val_loss: 0.4574 - val_mse: 0.3682 - val_mae:
0.4574
Epoch 2/20
33/33 [==============================] - 0s 15ms/step - loss: 0.3869 - m
se: 0.3830 - mae: 0.3869 - val_loss: 1.1010 - val_mse: 1.8897 - val_mae:
1.1010
Epoch 3/20
33/33 [==============================] - 1s 16ms/step - loss: 0.4512 - m
se: 0.4171 - mae: 0.4512 - val_loss: 0.6982 - val_mse: 1.0747 - val_mae:
0.6982
Epoch 4/20
33/33 [==============================] - 1s 16ms/step - loss: 0.3813 - m
se: 0.3596 - mae: 0.3813 - val_loss: 0.5284 - val_mse: 0.4692 - val_mae:
0.5284
Epoch 5/20
33/33 [==============================] - 0s 15ms/step - loss: 0.3569 - m
se: 0.2997 - mae: 0.3569 - val_loss: 0.7631 - val_mse: 1.1928 - val_mae:
0.7631
Epoch 6/20
33/33 [==============================] - 1s 16ms/step - loss: 0.3990 - m
se: 0.3410 - mae: 0.3990 - val_loss: 0.5984 - val_mse: 0.7061 - val_mae:
0.5984
Epoch 7/20
33/33 [==============================] - 1s 15ms/step - loss: 0.3418 - m
se: 0.3168 - mae: 0.3418 - val_loss: 0.7062 - val_mse: 0.9907 - val_mae:
0.7062
Epoch 8/20
33/33 [==============================] - 1s 15ms/step - loss: 0.4178 - m
se: 0.3958 - mae: 0.4178 - val_loss: 0.5408 - val_mse: 0.5767 - val_mae:
0.5408
Epoch 9/20
33/33 [==============================] - 1s 16ms/step - loss: 0.3101 - m
se: 0.2057 - mae: 0.3101 - val_loss: 0.5016 - val_mse: 0.4679 - val_mae:
0.5016
Epoch 10/20
33/33 [==============================] - 1s 15ms/step - loss: 0.3616 - m
se: 0.3395 - mae: 0.3616 - val_loss: 0.6711 - val_mse: 1.0438 - val_mae:
0.6711
Epoch 11/20
33/33 [==============================] - 0s 15ms/step - loss: 0.4159 - m
se: 0.3870 - mae: 0.4159 - val_loss: 0.5680 - val_mse: 0.6867 - val_mae:
0.5680
Epoch 12/20
33/33 [==============================] - 1s 16ms/step - loss: 0.3136 - m
se: 0.2468 - mae: 0.3136 - val_loss: 0.5952 - val_mse: 0.6548 - val_mae:
0.5952
Epoch 13/20
33/33 [==============================] - 1s 15ms/step - loss: 0.3769 - m
se: 0.3324 - mae: 0.3769 - val_loss: 0.5254 - val_mse: 0.6036 - val_mae:
0.5254
Epoch 14/20
33/33 [==============================] - 1s 15ms/step - loss: 0.4065 - m
se: 0.3417 - mae: 0.4065 - val_loss: 0.4589 - val_mse: 0.4388 - val_mae:
0.4589
Epoch 15/20
33/33 [==============================] - 0s 15ms/step - loss: 0.3294 - m
se: 0.2943 - mae: 0.3294 - val_loss: 0.6634 - val_mse: 1.0137 - val_mae:
0.6634
```

```
Epoch 16/20
33/33 [==============================] - 1s 16ms/step - loss: 0.3953 - m
se: 0.3553 - mae: 0.3953 - val_loss: 0.4733 - val_mse: 0.4675 - val_mae:
0.4733
Epoch 17/20
33/33 [==============================] - 1s 16ms/step - loss: 0.3465 - m
se: 0.2314 - mae: 0.3465 - val_loss: 0.4671 - val_mse: 0.4098 - val_mae:
0.4671
Epoch 18/20
33/33 [==============================] - 0s 15ms/step - loss: 0.3390 - m
se: 0.3125 - mae: 0.3390 - val_loss: 0.6018 - val_mse: 0.9035 - val_mae:
0.6018
Epoch 19/20
33/33 [==============================] - 1s 18ms/step - loss: 0.3935 - m
se: 0.3696 - mae: 0.3935 - val_loss: 0.4443 - val_mse: 0.4183 - val_mae:
0.4443
Epoch 20/20
33/33 [==============================] - 1s 19ms/step - loss: 0.2700 - m
se: 0.1580 - mae: 0.2700 - val_loss: 0.7127 - val_mse: 1.0537 - val_mae:
0.7127
```

TODO 7.4.2 Display the history content

In [ ]:
```python
import matplotlib.pyplot as plt
```



TODO 7.4.3 Convert predictions to original range of values and plot an interesting part. The example plots show predictions from the range [100:200]

In [81]:
```python
y_test_orig = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_orig = scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test_orig = y_test_orig.reshape(-1)
y_pred_orig = y_pred_orig.reshape(-1)
```

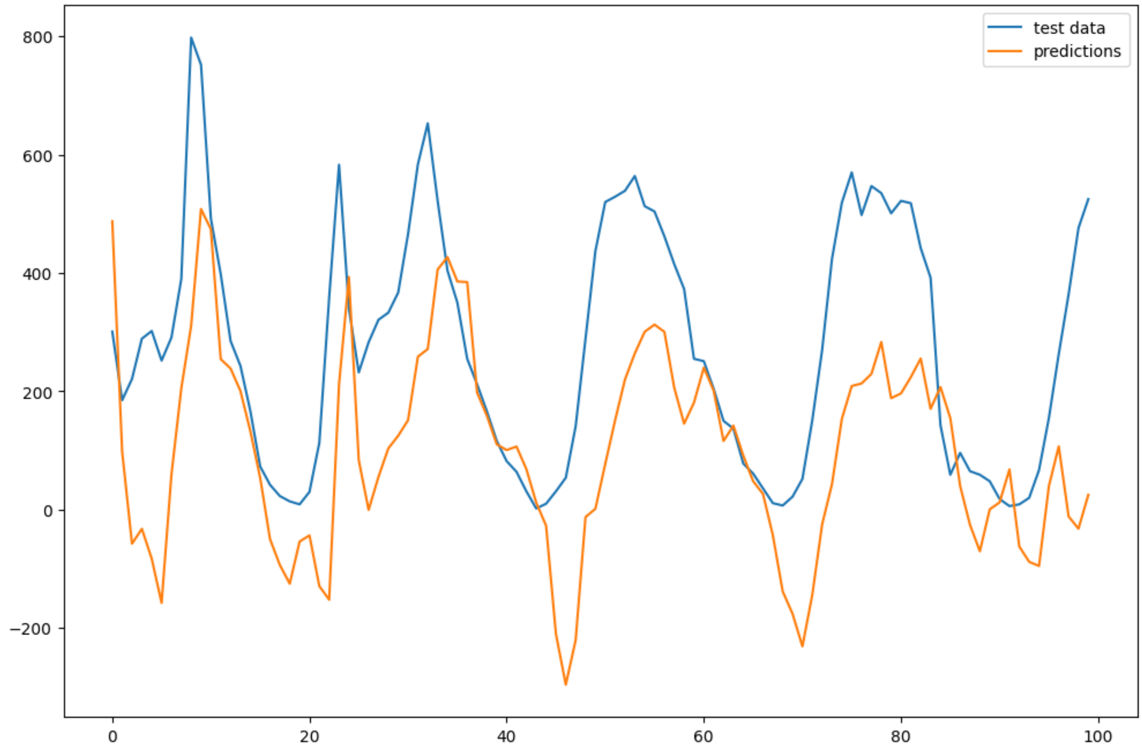
In [82]:
```python
r = compute_scores(y_test,y_pred)
print(r)
```

```
{'r2': -0.42897834424032966, 'mse': 3.0113139999171836, 'rmse': 1.735313
8044507062, 'maxe': 6.197833985519388, 'med': 1.027468854636469, 'mae':
1.3318156623588446, 'mape': 3.848662361940473}
```

In [83]:
```
r = compute_scores(y_test_orig,y_pred_orig)
print(r)
```

```
{'r2': -0.42897835119704064, 'mse': 69012.39148445425, 'rmse': 262.70209
64599526, 'maxe': 938.2648628354073, 'med': 155.54432678222656, 'mae': 2
01.61815425139153, 'mape': 2.434462716420465}
```

In [84]:
```
plt.plot(y_test_orig[100:200],label='test data')
plt.plot(y_pred_orig[100:200],label='predictions')
plt.legend()
plt.show()
```



7.4.4 Stacked GRU

This is rateher a time-demanding configuration, therefore we will train it during a few epochs.

In [85]:
```
from keras.models import Sequential
from keras import layers
#from keras.optimizers import RMSprop
import tensorflow
from tensorflow.keras.optimizers import RMSprop
from keras.layers import BatchNormalization
from keras import regularizers

model  = Sequential()
model.add(layers.GRU(32,  dropout=0.05,
                          recurrent_dropout=0.05,
                          return_sequences=True,
                          input_shape=(window_size,1),
                          kernel_regularizer=regularizers.l2(0.001)))
```

```
model .add(layers.GRU(64, activation='relu',
                          dropout=0.05,
                          recurrent_dropout=0.05,
                          kernel_regularizer=regularizers.l2(0.001)))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae',metrics=['mse','mae'])
```

In [103…

```
train_gen = generator(x_scaled,
                      window_size=window_size,
                      horizon=horizon,
                      min_index=0,
                      max_index=12_000,
                      shuffle=False,
                      step=1,
                      batch_size=batch_size,
                      expand_dims=True)
val_gen = generator(x_scaled,
                     window_size=window_size,
                     horizon=horizon,
                     min_index=12_000,
                     max_index=None,
                     shuffle=False,
                     step=1,
                     batch_size=batch_size,
                     expand_dims=True)
train_steps = (12_000-window_size-horizon-batch_size)//window_size
print(train_steps)

val_steps = (x_scaled.size-12_000-window_size-horizon-batch_size)//window
print(val_steps)
```

```
33
14
```

In [104…

```
hist = model.fit(train_gen,steps_per_epoch=train_steps,
                                            epochs=5,
                                            batch_size=batch_size,
                                            validation_data=val_gen,
                                            validation_steps=val_steps)
```

```
Epoch 1/5
33/33 [==============================] - 1s 17ms/step - loss: 0.2807 - m
se: 0.1635 - mae: 0.2807 - val_loss: 0.4034 - val_mse: 0.3058 - val_mae:
0.4034
Epoch 2/5
33/33 [==============================] - 1s 15ms/step - loss: 0.3538 - m
se: 0.3392 - mae: 0.3538 - val_loss: 0.7619 - val_mse: 1.0944 - val_mae:
0.7619
Epoch 3/5
33/33 [==============================] - 1s 16ms/step - loss: 0.3894 - m
se: 0.3641 - mae: 0.3894 - val_loss: 0.6422 - val_mse: 0.9231 - val_mae:
0.6422
Epoch 4/5
33/33 [==============================] - 1s 15ms/step - loss: 0.3143 - m
se: 0.2698 - mae: 0.3143 - val_loss: 0.4538 - val_mse: 0.3735 - val_mae:
0.4538
Epoch 5/5
33/33 [==============================] - 0s 15ms/step - loss: 0.3301 - m
se: 0.2692 - mae: 0.3301 - val_loss: 0.6402 - val_mse: 0.9436 - val_mae:
0.6402
```

TODO 7.4.3 Testing with `batch_size=1` is rather a long process.

- Configure `test_gen` to produce larger batches.
- Compute a number of steps to be done to cover testing data [12000:]
- Some data at the end will not be tested
- Select appropriate subsequence of `x_scaled`
- Compute scores

In [88]:
```
length = len(x_scaled)-12_000

y_pred = model.predict(test_gen,steps=length)
y_pred = y_pred.reshape(-1)
print(y_pred.shape)
```

```
5379/5379 [==============================] - 96s 18ms/step
(5379,)
```

In [89]:
```
y_test = x_scaled[12_000:]
print(y_test)
```

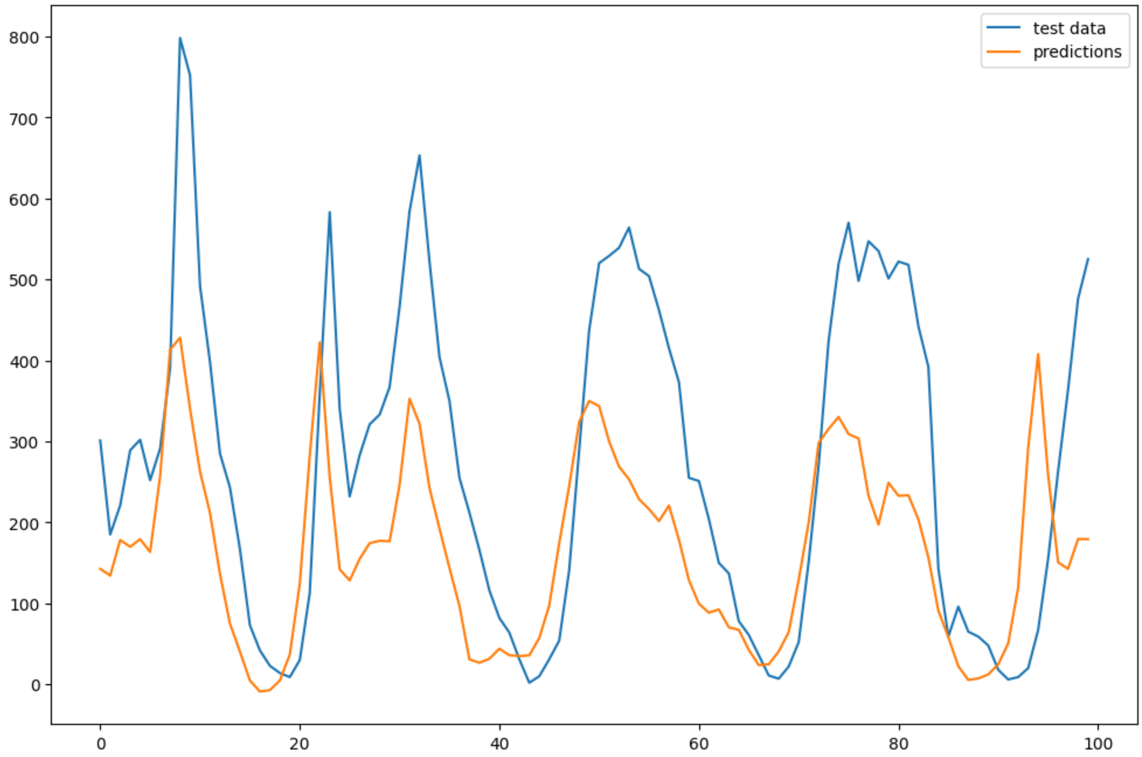
```
[-0.99469668 -0.88240091 -0.45303472 ... -0.45303472 -0.6445981
 -0.7238657 ]
```

TODO 7.4.4 Return to the original ranges and plot the data

In [90]:
```
y_test_orig = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_orig = scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test_orig = y_test_orig.reshape(-1)
y_pred_orig = y_pred_orig.reshape(-1)

r = compute_scores(y_test_orig,y_pred_orig)
print(r)
```

```
{'r2': -0.09032459280500627, 'mse': 52657.136184571675, 'rmse': 229.4714
2781743366, 'maxe': 863.8881225585938, 'med': 129.95213317871094, 'mae':
171.84590732601342, 'mape': 3.53883617286483}
```

In [91]:
```
plt.plot(y_test_orig[100:200],label='test data')
plt.plot(y_pred_orig[100:200],label='predictions')
plt.legend()
plt.show()
```



7.4.5 Conv1D - applying 1D convolution

Finally, we will perfom experiment with 1D convolutional filters

In [121…
```
from keras.models import Sequential
from keras import layers
model = Sequential()
model.add(layers.Conv1D(filters=32, kernel_size=7, input_shape=(window_si
model.add(layers.MaxPooling1D(pool_size=2))
model.add(layers.Conv1D(filters=64, kernel_size=7, activation='relu',padd
model.add(layers.MaxPooling1D(pool_size=2))
model.add(layers.Flatten())
model.add(layers.Dense(16,activation='relu'))
model.add(layers.Dense(1))

model.compile(optimizer='adam', loss='mae',metrics=['mse','mae'])
model.summary()
```

```
Model: "sequential_41"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d_6 (Conv1D)           (None, 336, 32)           256       
                                                                 
 max_pooling1d_4 (MaxPooling  (None, 168, 32)          0         
 1D)                                                             
                                                                 
 conv1d_7 (Conv1D)           (None, 168, 64)           14400     
                                                                 
 max_pooling1d_5 (MaxPooling  (None, 84, 64)           0         
 1D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 5376)              0         
                                                                 
 dense_81 (Dense)            (None, 16)                86032     
                                                                 
 dense_82 (Dense)            (None, 1)                 17        
                                                                 
=================================================================
Total params: 100,705
Trainable params: 100,705
Non-trainable params: 0
_________________________________________________________________
```

In [119…

```
train_gen = generator(x_scaled,
                      window_size=window_size,
                      horizon=horizon,
                      min_index=0,
                      max_index=12_000,
                      shuffle=False,
                      step=1,
                      batch_size=batch_size,
                      expand_dims=True)
val_gen = generator(x_scaled,
                    window_size=window_size,
                    horizon=horizon,
                    min_index=12_000,
                    max_index=None,
                    shuffle=False,
                    step=1,
                    batch_size=batch_size,
                    expand_dims=True)
train_steps = (12_000-window_size-horizon-batch_size)//window_size
print(train_steps)

val_steps = (x_scaled.size-12_000-window_size-horizon-batch_size)//window
print(val_steps)
```

```
33
14
```

In [94]:

```
hist = model.fit(train_gen,steps_per_epoch=train_steps,
                                            epochs=20,
                                            batch_size=batch_size,
                                            validation_data=val_gen,
                                            validation_steps=val_steps)
```

```
Epoch 1/20
33/33 [==============================] - 1s 18ms/step - loss: 0.5885 - m
se: 0.7080 - mae: 0.5885 - val_loss: 1.1053 - val_mse: 2.5588 - val_mae:
1.1053
Epoch 2/20
33/33 [==============================] - 1s 16ms/step - loss: 0.5550 - m
se: 0.6699 - mae: 0.5550 - val_loss: 1.9004 - val_mse: 4.8980 - val_mae:
1.9004
Epoch 3/20
33/33 [==============================] - 0s 14ms/step - loss: 0.7142 - m
se: 1.0982 - mae: 0.7142 - val_loss: 1.0366 - val_mse: 1.9365 - val_mae:
1.0366
Epoch 4/20
33/33 [==============================] - 0s 14ms/step - loss: 0.6023 - m
se: 0.7853 - mae: 0.6023 - val_loss: 1.3446 - val_mse: 3.1480 - val_mae:
1.3446
Epoch 5/20
33/33 [==============================] - 0s 14ms/step - loss: 0.4771 - m
se: 0.5225 - mae: 0.4771 - val_loss: 1.1466 - val_mse: 2.4045 - val_mae:
1.1466
Epoch 6/20
33/33 [==============================] - 0s 14ms/step - loss: 0.7196 - m
se: 1.0789 - mae: 0.7196 - val_loss: 1.1350 - val_mse: 2.3973 - val_mae:
1.1350
Epoch 7/20
33/33 [==============================] - 0s 14ms/step - loss: 0.5847 - m
se: 0.7957 - mae: 0.5847 - val_loss: 1.3251 - val_mse: 2.9392 - val_mae:
1.3251
Epoch 8/20
33/33 [==============================] - 0s 14ms/step - loss: 0.5040 - m
se: 0.5900 - mae: 0.5040 - val_loss: 0.7027 - val_mse: 0.8237 - val_mae:
0.7027
Epoch 9/20
33/33 [==============================] - 0s 14ms/step - loss: 0.4911 - m
se: 0.4556 - mae: 0.4911 - val_loss: 0.6437 - val_mse: 0.8418 - val_mae:
0.6437
Epoch 10/20
33/33 [==============================] - 0s 14ms/step - loss: 0.4653 - m
se: 0.4770 - mae: 0.4653 - val_loss: 1.0951 - val_mse: 2.1039 - val_mae:
1.0951
Epoch 11/20
33/33 [==============================] - 0s 14ms/step - loss: 0.5395 - m
se: 0.5699 - mae: 0.5395 - val_loss: 0.6826 - val_mse: 0.8314 - val_mae:
0.6826
Epoch 12/20
33/33 [==============================] - 0s 14ms/step - loss: 0.3683 - m
se: 0.2772 - mae: 0.3683 - val_loss: 0.7809 - val_mse: 1.0017 - val_mae:
0.7809
Epoch 13/20
33/33 [==============================] - 0s 15ms/step - loss: 0.4490 - m
se: 0.4510 - mae: 0.4490 - val_loss: 0.7990 - val_mse: 1.0544 - val_mae:
0.7990
Epoch 14/20
33/33 [==============================] - 0s 14ms/step - loss: 0.4777 - m
se: 0.4764 - mae: 0.4777 - val_loss: 0.8238 - val_mse: 1.0878 - val_mae:
0.8238
Epoch 15/20
33/33 [==============================] - 0s 14ms/step - loss: 0.4065 - m
se: 0.3846 - mae: 0.4065 - val_loss: 0.8260 - val_mse: 1.2973 - val_mae:
0.8260
```

```
Epoch 16/20
33/33 [==============================] - 0s 14ms/step - loss: 0.4133 - m
se: 0.4003 - mae: 0.4133 - val_loss: 0.6453 - val_mse: 0.6994 - val_mae:
0.6453
Epoch 17/20
33/33 [==============================] - 0s 14ms/step - loss: 0.3890 - m
se: 0.3004 - mae: 0.3890 - val_loss: 0.5561 - val_mse: 0.6058 - val_mae:
0.5561
Epoch 18/20
33/33 [==============================] - 0s 14ms/step - loss: 0.3748 - m
se: 0.3563 - mae: 0.3748 - val_loss: 0.7097 - val_mse: 1.0871 - val_mae:
0.7097
Epoch 19/20
33/33 [==============================] - 0s 14ms/step - loss: 0.4478 - m
se: 0.4423 - mae: 0.4478 - val_loss: 0.5020 - val_mse: 0.5068 - val_mae:
0.5020
Epoch 20/20
33/33 [==============================] - 0s 14ms/step - loss: 0.3291 - m
se: 0.2233 - mae: 0.3291 - val_loss: 0.8171 - val_mse: 1.2689 - val_mae:
0.8171
```

TODO 7.4.5 Make predictions, compute regression scores

In [122…

```
length = len(x_scaled)-12_000

y_pred = model.predict(test_gen,steps=length)
print(y_pred.shape)
y_pred = y_pred.reshape(-1)
print(y_pred.shape)
```

```
5379/5379 [==============================] - 3s 449us/step
(5379, 1)
(5379,)
```

In [96]:

```
y_test = x_scaled[12_000:]
print(y_test.shape)
```

```
(5379,)
```

In [97]:

```
r = compute_scores(y_test,y_pred)
print(r)
```

```
{'r2': -0.6729409455981648, 'mse': 3.5254211589837654, 'rmse': 1.8776104
918176628, 'maxe': 5.678658415124616, 'med': 1.2511941096346928, 'mae':
1.516513884616032, 'mape': 3.7080286270421827}
```

TODO 7.4.6 Plot the data returning to the original data ranges

In [98]:

```
y_test_orig = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_orig = scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test_orig = y_test_orig.reshape(-1)
y_pred_orig = y_pred_orig.reshape(-1)

r = compute_scores(y_test_orig,y_pred_orig)
print(r)
```

```
{'r2': -0.672940926771907 8, 'mse': 80794.54392855865, 'rmse': 284.243810
71284324, 'maxe': 859.6689910888672, 'med': 189.4131851196289, 'mae': 22
9.5788647138534, 'mape': 6.4519597031302025}
```

In [99]:
```
plt.plot(y_test_orig[100:200],label='test data')
plt.plot(y_pred_orig[100:200],label='predictions')
plt.legend()
plt.show()
```