Multivariate time series analysis with a CNN - Part 2

Dealing with multivariate time series for CNN

Initially, the data look like this:

```
 (instance 1) f1 f2 f3 ... fn label
(instance 2) f1 f2 f3 ... fn label
(instance 3) f1 f2 f3 ... fn label
(instance 4) f1 f2 f3 ... fn label
(instance 5) f1 f2 f3 ... fn label
...
(instance m) f1 f2 f3 ... fn label
```

Each instance is an *observation*, a measurement of the different sensors in the system. Data are collected every second: instance 2 occurred 1s after instance 1 and so on.

Using a CNN architecture will allow us to learn pattern occurring over a **timeframe**.

If we had a univariate time series, such as `0.66 , 0.08, 0.76, 0.37, 0.16, 0.57 , 0.54, 0.22, 0.63, 0.23` we could be interested on predicting a value of the series given the previous *n* values (a **sliding window**).

For example:
we could predict `0.37` from the window `0.66, 0.08, 0.76` and then
we could predict `0.16` from the window `0.08, 0.76, 0.37` and then
we could predict `0.57` from the window `0.76, 0.37, 0.16` etc.

Here, the window has size 3 and the stride of the window is 1.

This can be done with a regular, fully connected neural network, but CNN are better suited for high dimensional data and their ability to learn *local* patterns in data.

Going back to the data we are interested in, we will rely on the same idea to process them:

```
 (sample 1)
 f1 f2 f3 ... fn
 f1 f2 f3 ... fn
 f1 f2 f3 ... fn label
(sample 2)
 f1 f2 f3 ... fn
 f1 f2 f3 ... fn
 f1 f2 f3 ... fn label
...
(sample k)
 f1 f2 f3 ... fn
 f1 f2 f3 ... fn
 f1 f2 f3 ... fn label
```

We have to group observations using a time frame (for example, here we have a timeframe of 3 seconds: 3 observations to predict the label at time `t+3`)

Here is how my data look once I applied this transformation to them:

```
[[[f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]

 [[f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]]

 [[f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]
  [f1 f2  f3  f4  f5]]]
```

Those are the first 3 instances of the transformed data. The timeframe used in this example is 4 (4 observations are aggregated into a single instance, each observation has 5 attributes).

The shape of the data is `(number of instances, timeframe, number of features)` .

Here are the labels for the corresponding instances:

```
[label1 label2 label3]
```

The shape of the labels is `(number of intances)`

The data

Source : https://www.sciencedirect.com/science/article/abs/pii/S0920410520309372 (Although the paper is interesting to understand the global context of the data, it is pointless to go and grab some code presented there: the methods we will use is totally different and much more simple. If I suspect the tiniest attempt of using the code developed by the author of this paper, you will (1) have 0 to your project and (2) be reported as cheater).

The data you will be working on are sensors measurement from oil wells. This is a classification problem, in which you will have to predict if the system is functioning normally (`0`), if the system is in a transient state (`108`) or if the system is malfunctioning (`8`).

Features

- P-PDG Pressure at permanent downhole gauge (PDG) in Pa
- P-TPT Pressure at temperature/pressure transducer (TPT) in Pa
- T-TPT Temperature at temperature/pressure transducer (TPT) in Celcius
- P-MON-CKP Pressure upstream of production choke (CKP) in Pa
- T-JUS-CKP Temperature downstream of production choke (CKP) in Celcius
- P-JUS-CKGL Pressure downstream of gas lift choke (CKGL) in Pa
- T-JUS-CKGL Temperature downstream of gas lift choke (CKGL) in Celcius
- QGL Gas lift flow rate in m^3/s

=> `P-JUS-CKGL`, `T-JUS-CKGL` and `QGL` are all NaN in the 2 simulated data used for this PW. The entire column must be dropped from the data.

=> The timestamp must be converted to an actual timestamp (initially read as an `object`)

=> There are 3 classes: `0` is normal; `108` is a transient state (from normal to abnormal) and `8` is abnormal (hydrate in production line). Those classes are imbalanced. This calls for a **stratified split** for the training/validation data (test data will be loaded from an other file, dedicated for testing). That same imbalance is observed in the testing data as well.

```
 Training data:
8      15999
108     9200
0       1800

Testing data:
108    17300
8       7899
0       1776
```

=> Attributes have different scales: when the data are transformed for processing by Conv1D, they must also be **normalized**.

Useful functions to help transforming the data

`concat` (pandas) https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html
`shift` (pandas) https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.shift.html?highlight=shift#pandas.DataFrame.shift
`vstack` and `hstack` (numpy) https://numpy.org/doc/stable/reference/generated/numpy.hstack.html and https://numpy.org/doc/stable/reference/generated/numpy.vstack.html

You do not have to use those functions, I merely found them handy so I thought I'd share.

The data you must use for training (resp. testing) are in the file `training.csv` (resp. `testing.csv`).

I also provide the transformed data in csv files: * you can use them to have a better understanding of what is expected from the transformation and/or * you can use them for the modelling part if you did not manage to transform the data adequately yourselves (although in this case, you will not have points for the data transformation in your final grade, but at least you do not end up stuck and can keep working).

If you need to load the transformed data I provide

Files are pretty heavy! (~90MB)

The file for training contains 26975 instances, each of them has 125 points (it's actually 25 --the timeframe I used-- $\times$ 5 features). The labels `y`) do not need reshaping.

To load these data:

```
 x_train = np.loadtxt('x_train_transformed.csv')
x_train = x_train.reshape(x_train.shape[0], 25, 5)

y_train = np.loadtxt('y_train_transformed.csv')
```

The same goes for the testing data.

Predicting normal/transient/abnormal state

1. Propose a CNN architecture for processing the data. You must use 1D convolutions. How many convolutional blocks do you intend to use? (justify your answer)
2. State which optimizer you will use for training. Set a callback for early stopping (with patience=3) (for this you will need to specify the validation data later on, in the call of `fit()`. For this split, do you need a stratified split or not? (explain your answer)
3. Train your model and plot the training and validation losses recorded in the `history` object returned by the `fit()` function
4. Evaluate your model using the test data. What can you say about the accuracy of the model with regard to the training accuracy observed during training?
5. Randomly select 1000 instances from your testing data (you will need to extract the labels as well) and compute the predicted label (function `predict()`)
6. Compute the confusion matrix for these predictions (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html). You can plot it using Seaborn:

```
 import seaborn as sns
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=['Normal','Transient','Abnormal (event 8)'],
            yticklabels=['Normal','Transient','Abnormal (event 8)'],
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

7. Using your confusion matrix, explain which are the specific errors your model did.