CERI - 8104 – HW #5

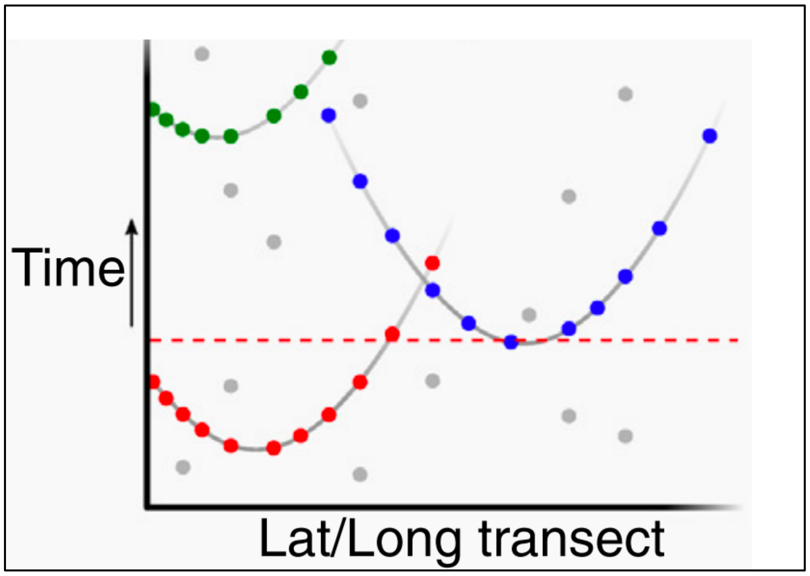
ML Seminar – Recurrent Neural Networks

Recurrent Neural Networks and Long-Short-Term-Memory Units for sequence prediction tasks

One common problem in observational seismology is the discrimination of seismic phases from noise recorded on seismograms. This task is especially challenging in low signal-to-noise (SNR) environments and in situations with many overlapping event arrivals. Commonly, seismic phases are difficult to identify based on single station records. Rather, analysts have to integrate information over many stations and time windows.

The task for this assignment is to solve a binary classification problem for which you will distinguish earthquakes (label=1) and noise (label=0) sources based on the arrival time patterns observed over an array of stations.

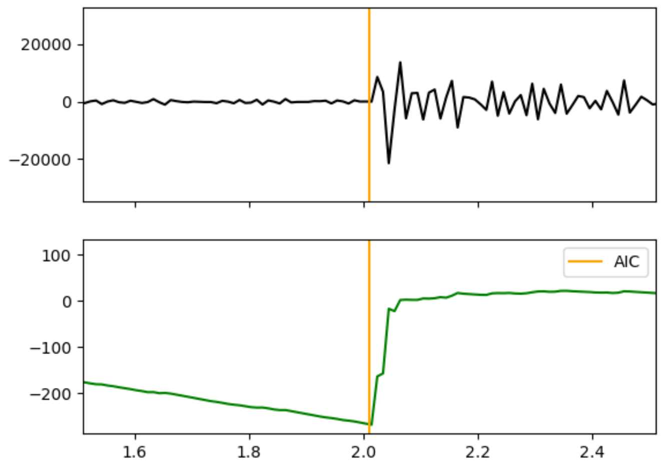
An example of arrival time patterns is shown below:



Here three earthquakes (red, blue, green) can be identified by their parabolic move-out which is easily distinguishable from incorrect phase picks highlighted in gray.

from: Woollam, J., Rietbrock, A., Leitloff, J. & Hinz, S. HEX: Hyperbolic Event eXtractor, a Seismic Phase Associator for Highly Active Seismic Regions. *Seismol. Res. Lett.* **91**, 2769–2778 (2020).

The following figure shows an example of a single station seismogram, with corresponding characteristic function and P-pick. Such picks are the basis for determining move-outs across an array as shown in the previous picture.



We will build a Neutral Net for earthquake phase detections, starting with simple One-to-One and Many-to-One sequence prediction problems. The first two sections of the homework will help build intuition for RNN architecture in tensorflow and keras. The third section will focus on solving the phase classification problem for specific training and testing datasets.

Part 1 – One-to-One sequence prediction

You will start be solving a simple sequence prediction problem for which the target label is a constant multiple of the feature vector.

1. Start by writing the header information (what does the program do, what are key variables, what is the final output, etc.) and import the usual python modules (numpy, matplotlib) needed for array processing and plotting. In addition, include the following imports:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

2. **Create the synthetic data:** `createSynData()`:
    a. Let's write a separate python function that returns a synthetic data set in the correct input shape for a sequential layer within the Keras module.

       To do so:
        i. create a numpy array of integers between [1 and 21). This will be your feature vector, ***X***.
        ii. The target labels are simply: $y = X*const$, here *const* = 15.

        iii. Now shift the targets back by 1 step 2 → 15; 3 → 30; 4 → 45; to make things more interesting.

e.g.: X = [2,3,4,5,6, …]; y = [15, 30, 45, 60, 75, ….]

The first Sequential layer in keras requires an input tensor of:
`inpute_shape = ([‘nInstances’], [‘nFeatures’], [‘nChannels’]).`
You will need to replace the strings in brackets with the correct values.

You can use:
`numpy.reshape()` to bring the tensor into the right shape. What do variables should your *`createSynData()`* function return?

b. Use the above function to create the training data set.

c. Testing data: Get the last feature element from the training data and add a small integer. You can use this number as starting point for your testing data so that training and testing data are not within the same range of numbers.

3. **Build the RNN.** The syntax for building a RNN is reasonably simple since you are just adding one layer at a time to your “Sequential” object:

```
model = Sequential()
model.add(LSTM(50, return_sequences=True,
               activation='relu', input_shape=(1, 1)))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
print( model.summary())
```

How many different layers does this RNN have? What are the layer types?

4. **Train and Predict**.
Now we can train the model and predict the target labels for the test data. To access the training performance records, assign the output of model.fit().history to a python dictionary:

```
dResults = model.fit(X_train, y_train, epochs=n_epochs,
             validation_split=0.2,
     batch_size=batch_size).history

print( 'simple test: ', model.predict(
np.array([30]).reshape((1,1,1))),'should be: ', 15*15)
y_predict = model.predict( X_test)
print( ‘y_hat: ’, y_predict[0:10])
print( ‘y_true: ’, y_test[0:10])
```

Next, we will use dResults ['loss'] and dResults['val_loss'] to check performance during training.

5. **Performance evaluation.**
    a. Plot the cost function for training and validation data.
    b. Report the coefficient of determination and mean-squared-error. You can also plot the residuals (y_test – y_predict) as a histogram and check whether they are Gaussian.
    c. Plot the target labels vs. feature vector for the test dataset.
    d. What do you observe? Is there an obvious correlation between *X_test* and *y_test*? Compare your ML-results with predictions using linear regression analysis.

Part 2 – Many-to-One sequence prediction (e.g. stock price)

We will now build on the previous simple example to predict a number based on a preceding series of numbers:

X_train	y_train
3.0, 3.1, 3.2, 3.3, 3.4, 3.5,…,3.9	4.0
6.0, 6.2, 6.4, 6.6, 6.8, 7.0, …,7.8	8.0
…	

For now, we will simply predict the next consecutive number in the sequence, but the problem could easily be modified to predict the penultimate number or the second number from the left etc.

This is referred to as a Many-to-One sequence prediction problem.

6. **Create the synthetic dataset**: The data set will consist of `nSequences (= 25)` with length, `iLength=10`. Consequently, your feature matrix should be brought into the following shape: (nSeq, iLength, 1). Chose random integer numbers between 1 and 10 as first numbers in the sequence, and randomly add a number between 0.1 and 0.5 to each consecutive element.

    Test data: Use the same step size between sequence elements but shift the starting point outside of the training range to give the RNN a bit of a challenge.

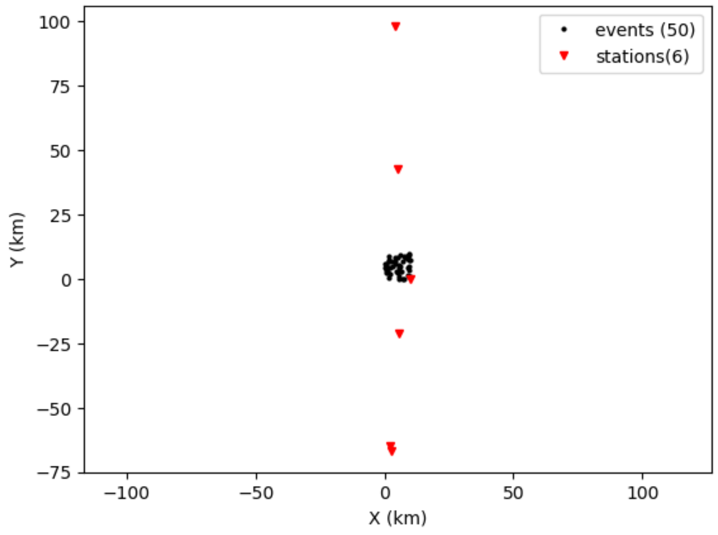
7. **Build, Train and Predict**: This step is the same as before but you will probably have to train over many hundreds of epochs. You can also experiment with a slightly deeper neural net and add `drop-out layers` to avoid getting stuck at a particular training result:

```
model = Sequential()

model.add( LSTM(units= n_hidden, return_sequences=True,
                input_shape=(iLength,1)))
model.add( Dropout( f_dropout))
for s_bool in [True, False]:
    model.add(LSTM(units = n_hidden, return_sequences = s_bool))
    model.add(Dropout(f_dropout))
###5 add fully connected output layer
model.add( Dense(units = 1))

# check size of ANN
model.summary()
# train
model.compile(optimizer = 'adam',loss = 'mean_squared_error')
```

8. **Performance evaluation**: You can copy paste the performance evaluation steps from Section 1.
    a. **What is the highest R2**-score you can achieve?
    b. Try modifying your script to predict the penultimate number. How well does your RNN perform?

Part 3 – Detect earthquake phase arrivals with a bi-directional LSTM

We will now transition to a binary classification problem (earthquake=1,noise=0) using sequences of phase arrivals across a six-station network:



As you can see in the above figure, there is a dense cluster of seismic swarms towards the center of the network, which should generate a nice parabolic signal in arrival times. However, many of the events occurred at similar times and the record is contaminated by many false picks. For this exercise you will decide which sequences of arrival times look promising and should be used for further processing.

1. **Load the data**:
   a. Read the .txt files with training and testing data: `arrival_times_test_ran_1234.txt` and `arrival_times_train_ran_1234`.txt
   b. Separate feature and target vectors to generate X_train, y_train, and X_test, y_test. Make sure to appropriately reshape X_train and X_test
2. **Build the RNN:**
   a. Build an RNN consisting of three bidirectional LSTM layers, three 20% dropout layers and one fully connected layer.
   b. Since we are dealing with a binary classification task, you should choose the 'binary_crossentropy' loss function and 'sigmoid' activation in the last layer.
3. **Train and Predict:**
   a. You can use the same syntax as before to train the model.

```
dRes = model.fit( X_train, y_train, epochs= n_epochs,
                    validation_split=0.2,
                    batch_size=batch_size,
                    shuffle=True).history
```

   b. What are the characteristics of the returned values in the fully-connected layers with sigmoidal activation? How can you convert the continuous output to binary class labels? (Hint: you can choose a threshold of 0.5 for the earthquake detection problem)

4. Performance Evaluation

a. Plot the loss curves for training and validation as well as accuracy for training and validation.
b. Add a binary confusion matrix and discuss your results.