Time Series Classification with Recurrent Neuro Network

EE660 Final Project

Huo Chen

Data Set Description

The data set I used for this project can be found on UCI machine learning archive:

https://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring

Goal

The author had an electronic nose in his home along with another set of temperature and humidity sensors. They were exposed to background home activities. Two different stimuli, banana and wine, were placed close to the sensor arrays at random times. And the duration of the stimuli varies form 7 minutes to 2 hours.

This sensor array produced time series data under three different conditions: banana, wine and background. The goal is to correct classify a time-series with respect to those three labels.

An example response of the sensor array to two different stimuli is given in [1]:

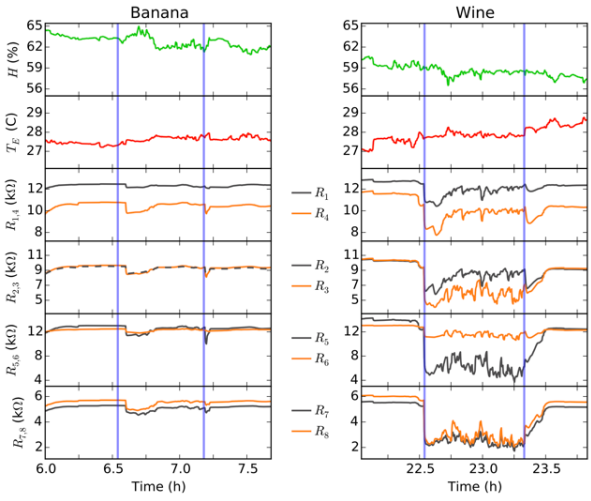


Figure 1 Example of sensor response

Data Pre-Processing

The entire data set was collected from 7/04/2015 to 9/17/2015 at an approximate interval of 1 second. For each day, the sensor array was at most exposed to only one type of stimuli for a certain length. And the 'background' data were taken between 11:00 am to 3:00 pm each day to avoid extra noise. Each time series data with the presence of banana, wine or the specific background activity is called one presentation of the experiment.

There are totally 928991 sample points in the data file, from which 99 presentations can be extracted. To compare our results with those reported in [1], we pre-process the data the same way as described below:

1. For each presentation, we divide the entire time series into 10 minutes' windows. Each consecutive time window is separated by 1 minutes.
2. Each window is truncated to a length of 540 data points. If there are not enough points within one window, then it is discarded.

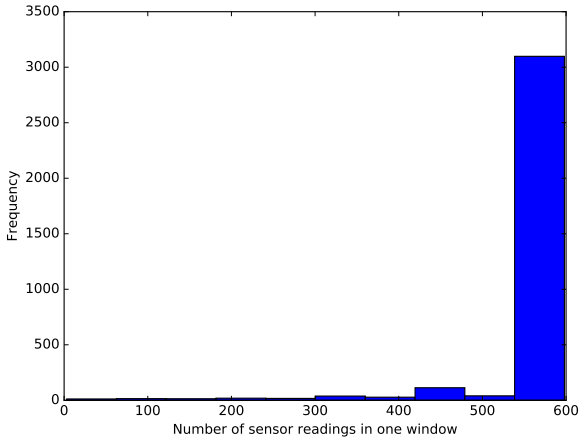


Figure 2 Distribution of data points in all windows

After pre-processing, there are a total of 3098 windows (samples), each of which is a 10 by 540 matrix. Our classification task is done on each sample. Thus, each sample points of our data lies in a 5400-dimension space.

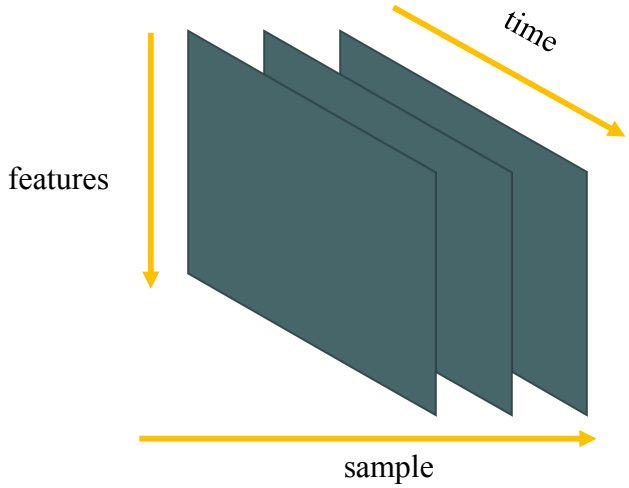
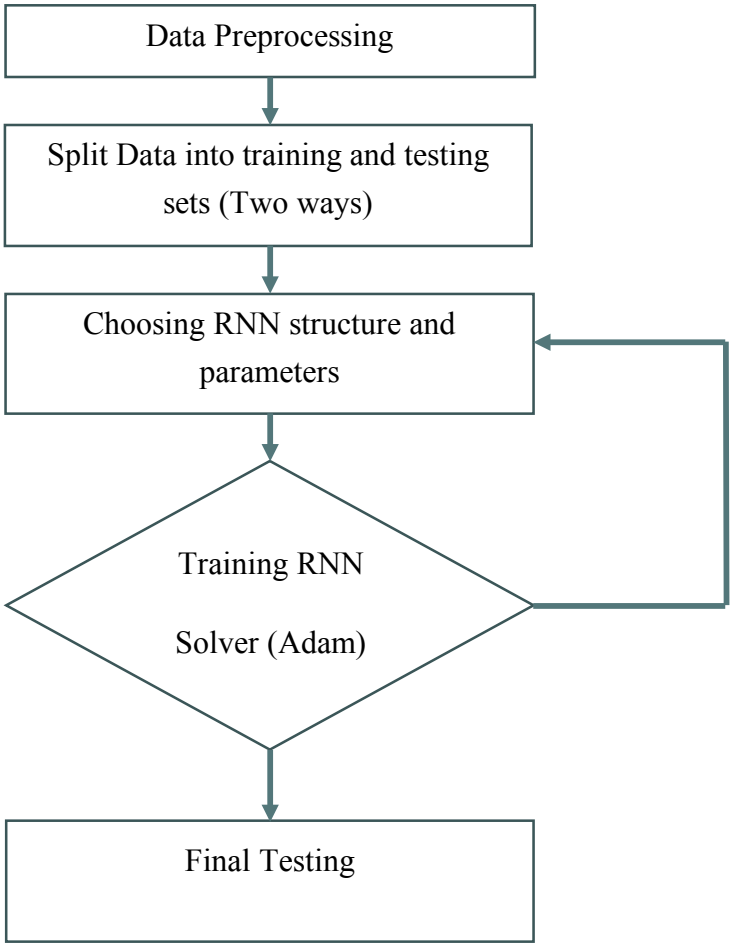


Figure 3 Data dimension

Methodology

Below is a flowchart of my overall procedure:



Recurrent Neural Network

I used recurrent neural network to do the classification. My main idea is based on [2]. The basic architecture of RNN is show below:

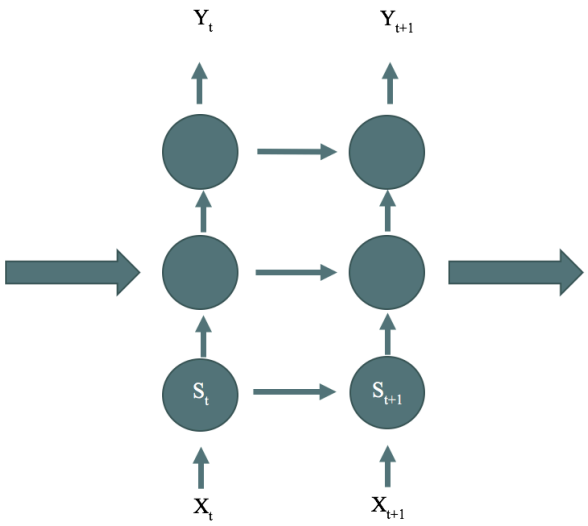


Figure 4 Architecture of RNN

The basic idea is that, the internal state for each neuron is fed forward as input for the next time step. The original neural network suffers greatly from 'vanishing gradient' problem, which makes the training process quite slow and inaccuracy. In our experiment, we use the long-short term memory (LSTM) architecture RNN, which is an improved version of the simple RNN scheme. This makes the training much faster.

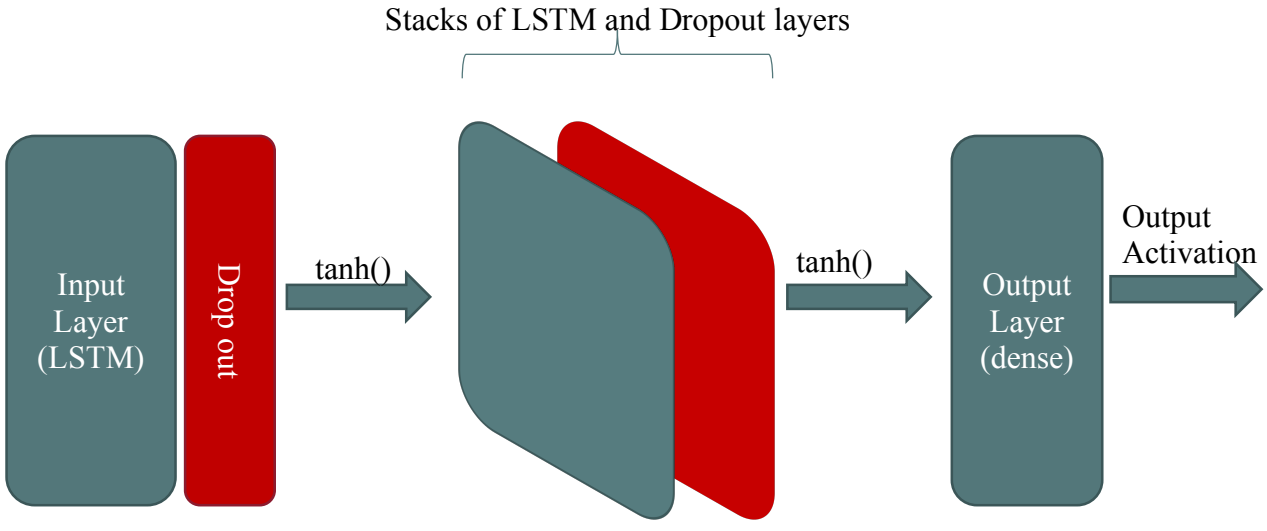


Figure 5 RNN structure

The output activation function is different based on different ways of encoding the classification problem.

Complexity Analysis

For neural network, it is hard to find an analytic expression for the computational complexity. A single backpropagation iteration is timed to one data sample, which is a 10 by 540 matrix. The back-propagation algorithm is 'Adam', which is one of the standard solver for RNN.

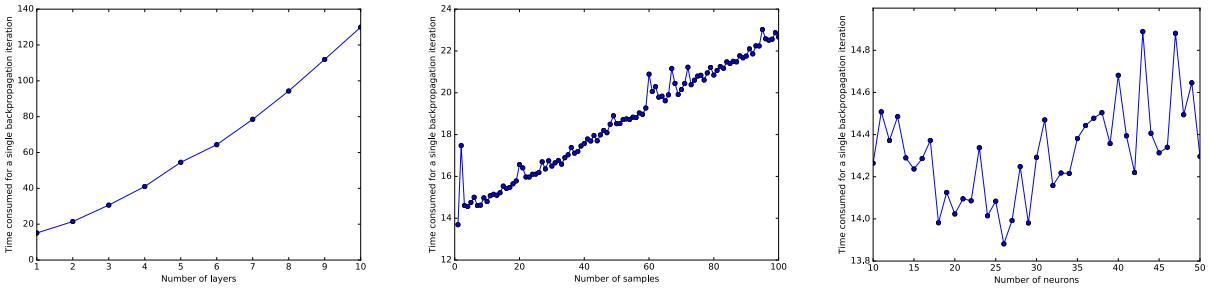


Figure 6 Timing for RNN

From the figures, we can see that the algorithm is linear in the number of sample points. However, it grows as polynomials (or even exponentials) as the number of layers increases. This means that we should keep the number of layers as small as possible, although this might result in losing some accuracy.

Another observation is that there is no clear relationship between the running timing and number of neurons per layer as far as our application is considered. However, since we only have 10 features and 3 class labels, 50 neurons are more than enough.

Encoding the Classification Problem

We have two ways to encode the classification problem.

One for all encoding

This idea is mainly described in [2]. Basically, we normalize all the feature vectors into the [-1,1]. Then we encode the class labels as a 3-dimension vector

$$c_1 = \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}, \qquad c_2 = \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}, \qquad c_3 = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}$$

After this, we append this class vector to our feature vector. And the problem becomes a standard regression problem. The loss function in this case is

$$L(\vec{w}) = \frac{1}{N}\frac{1}{T}\sum_{N}\sum_{T}(\hat{y}_c\text{-}y_c)^2 + (\hat{y}_p\text{-}y_p)^2$$

where y_c is the output dimension for classification task and y_p is the output dimension for prediction task.

The smallest decoding unit for our case is the 10 minutes' window. After the prediction is finished for a window, we sum all the values of y_{ct} which belong to this window. We choose the neuron with maximum activity as the class label. According to [2], this should be faster and more accurate than the standard classification procedure.

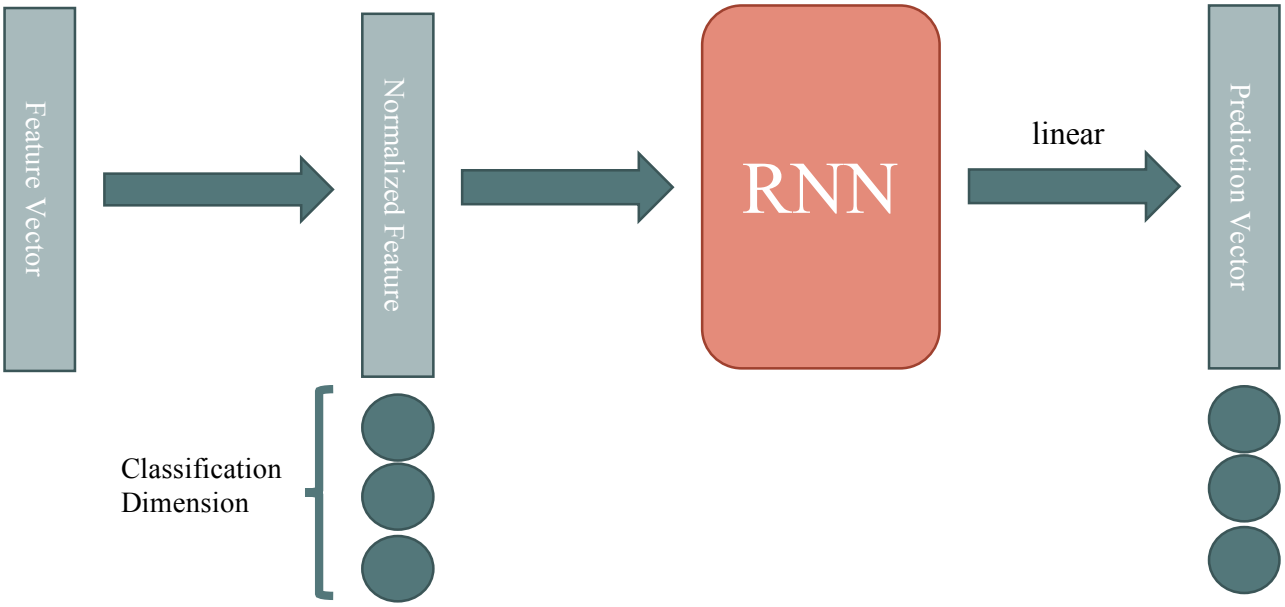


Figure 7 One for all encoding

Standard Classification

The standard classification procedure is, instead of appending class labels to feature vectors, we change the activation function of the output layer to softmax function. Then the output of the neural network is only the class vectors. In this case, we can also modify the loss function as

$$L(\vec{w}) = \frac{1}{N}\frac{1}{T}\sum_{N}\sum_{T} H(\hat{y}_c, y_c)$$

where $H(\cdot)$ is the cross-entropy between the predicted class vector and the target class vector.

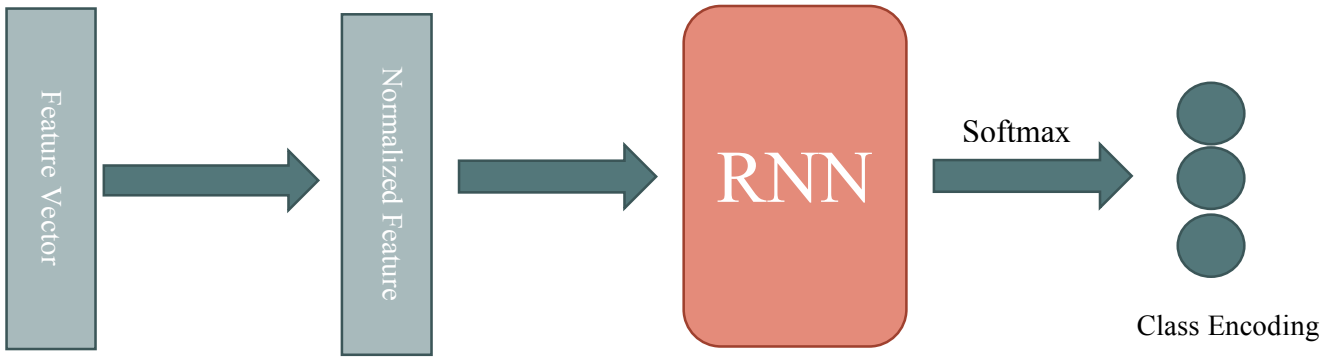


Figure 8 Illustration of standard classification procedure

Data Splitting

There are also two ways to split the data into training and testing set.

With Respect to Presentation

The data is split into 79 training presentations and 20 testing presentations, which means a split ratio of 0.8. In training process, all the data within one presentation is fed into the neural network in time order.

With Respect to Windows

The data is split into 2478 training windows and 620 testing windows. The split ratio in this case is also 0.8. In training process, all the windows are randomly shuffled before feeding into RNN.

Results

Table one shows the result for the case with one for all encoding scheme and group splitting method. The result is average over 10 different random split of training and testing sets:

Table 1 Result for the case of one for all encoding and group splitting

	Number of Hidden Layers	Neurons per Layer	Mean Misclassification Rate	Standard Deviation
1	1	10	0	0
2	2	14	0.002	0.006
3	1	10	0	0
4	2	14	0	0

The misclassification is calculated with respect to the number of windows. The result is impressive considering the number of data we have.

Table two show the result for the case of standard classification and group splitting.

Table 2 Result for the case of standard classification and group splitting

	Number of Hidden Layers	Neurons per Layer	Mean Misclassification Rate	Standard Deviation
1	1	10	0.0029	0.0077
2	1	15	0.0012	0.0033
3	1	20	0.0054	0.0021
4	2	10	0.018	0.056
5	2	11	0.0023	0.0074

	Number of Hidden Layers	Neurons per Layer	Mean Misclassification Rate	Standard Deviation
6	2	12	0.023	0.066

We can see from the table that one for all encoding does help to improve the accuracy for the classification task. A very counter intuitive observation is the accuracy seems irrelevant to the number of layers, which is usually not the case for neural network. It might indicate that one layer is enough for this task.

Table three show the case of one for all encoding and window splitting. In this case, the results are averaging over 20 random splitting of the original data set.

Table 3 Result for the case of one for all encoding and windows splitting

	Number of Hidden Layers	Neurons per Layer	Mean Misclassification Rate	Standard Deviation
1	1	10	0.27	0.033
2	1	13	0.22	0.04
3	1	14	0.19	0.032
4	2	10	0.23	0.05
5	2	13	0.18	0.026
6	2	14	0.185	0.05

We see that the accuracy drops by a large margin in this case. This might indicate that, by feeding a longer time series into RNN, or at least feeding the entire presentation in time order together, the

algorithm converges closer to the global optimal point. This agrees with our intuition about recurrent neural networks.

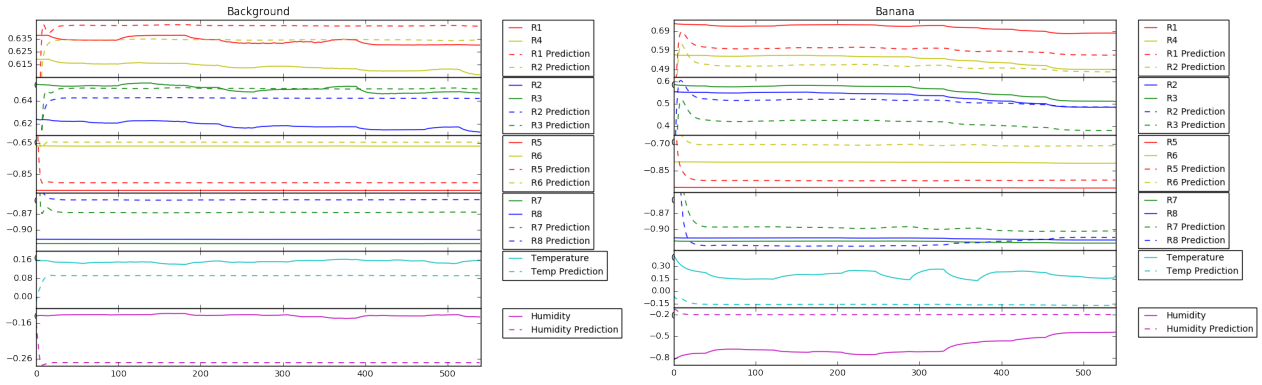
The result reported in [1] is:

Feature set	Cross-validated accuracy	Accuracy in test	Std	*p*-Value
RS	78.5%	76.5%	6.8%	0.02*
RS,T,H	73.3%	71.1%	6.8%	$1 \cdot 10^{-12}$**
FS	72.4%	71.2%	**4.8%**	$2 \cdot 10^{-12}$**
RS,FS	82.6%	**80.9%**	6.3%	1

In their best case, they have a test accuracy of 80.9%. This mean RNN is really a great and powerful tool to analyze time series data.

Final Thought

I tried to find out the magic behind the performance of RNN as the final part of this project. Below I plot three random case of original and predicted data from one of our best runs:



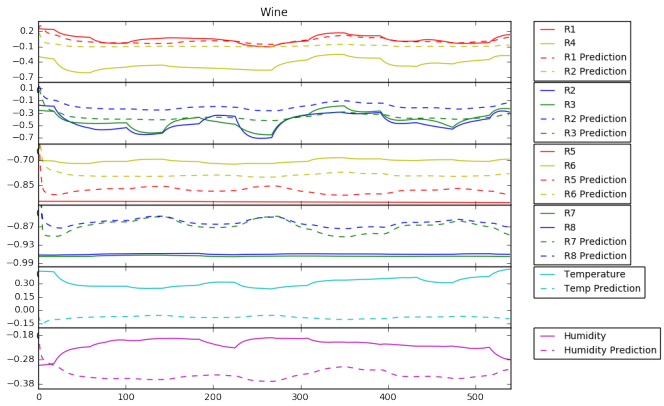


Figure 9 Time series predicting performance

From the figures above we can see that the performance of predicting task of RNN is not so good. To me, the performance for classification may be related to some indirect features of the sensor readings. For example, there is a lot of wiggles when wine is present. And when banana is present, the predicting curve for the sensor pair R2, R3 and R7, R8 are inverted. However, much more work could be done to explore the magic and power behind RNN.

Bibliography

[1] R. Huerta, T. Mosqueiro, J. Fonollosa, N. Rulkov and I. Rodrigues-Lujia, "Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring.," *Chemometrics and Intelligent Laboratory Systems,* no. 157, pp. 169-176, 2016.

[2] M. Husken and P. Stagge, "Recurrent neural networks for time series classification.," *Neurocomputing,* no. 50, pp. 223-235, 2003.