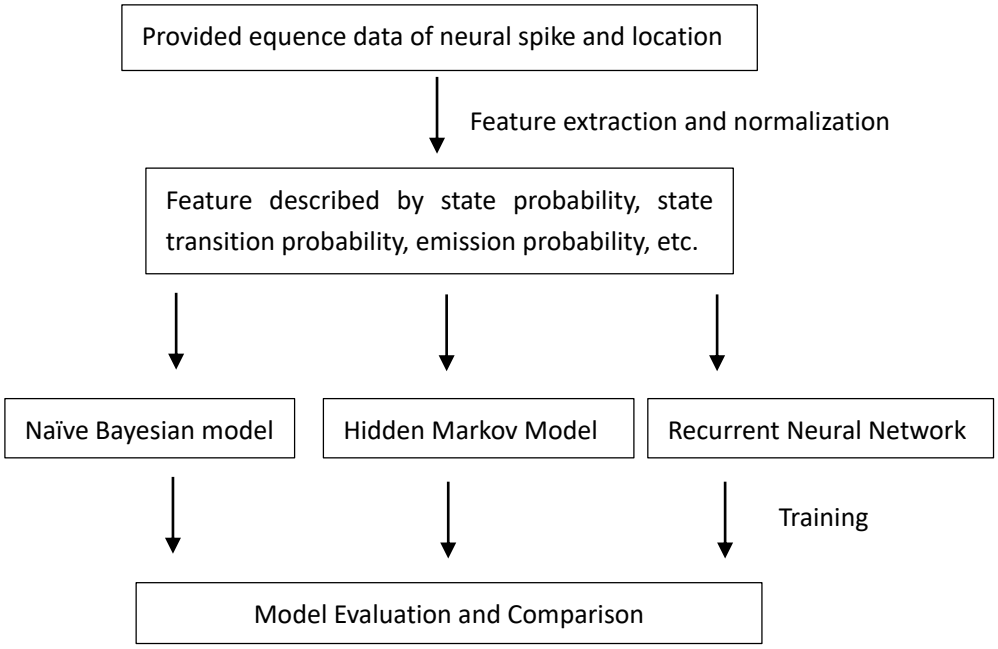
Bayesian Decoding of Neural Spike

Li Yuhui 2013012470

Introduction

Neural spike train analysis is an important task in computational neuroscience which aims to understand neural mechanisms and gain insights into neural circuits. Here in this course project, three basic statistical models are used in the field of neural decoding, predicting the position based on neural spike data. The methods, performance and comparison are discussed carefully as follows. The flowchart of the whole experience is shown in Scheme 1.



Scheme 1. The Flowchart of the Experience Done in Neural Decoding Project

Methods

Experimental Setup

All the codes are written in python 2.7 and can be run in windows 10 and Ubantu 14.04. Useful packages are included such as numpy, scipy, pybrain and hmmlearn.

Data Source

Dataset is provided by TA, which includes 7 neural spike dataset through time and the one-dimensional position of rats for each.

Data Preprocessing

Here, each data for unit time (smallest interval) is considered as an example. Neural spike data constitutes the observation, and the position of rats constitutes the hidden state. For neural spike data, an m*p matrix is constructed where m is the example number and p is the recorded neuron

number. For position data, a one-dimensional sliding window is used to divide the whole continuous coordinate without overlap into small discrete bins. The number of state can be thus decided by equation (1).

$$\text{States} = \frac{\max(corrdinate) - \min(corrdinate)}{window\ size} \tag{1}$$

The transition matrix A can be estimated by continuous probability function as shown in equation (2). However, here the transition matrix is simply calculated as discrete probability function estimated by frequency.

$$a_{ij} = ke^{-(j-i)} \tag{2}$$

Where k is the normalization factor.

The posterior probability of position when neural spike is provided is estimated by frequency.

Bayesian Decoding

The Naïve Bayesian model is based on Bayes' theorem with independence assumptions between predictors. A Naïve Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naïve Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods. Here, a Naïve Bayesian model is used to decode the state of rats which is described in equation (3).

$$\Pr(\mathrm{x}_t|\mathrm{y}_t) = \frac{\Pr(\mathrm{x}_t)\Pr(\mathrm{y}_t|\mathrm{x}_t)}{\Pr(\mathrm{y}_t)} \tag{3}$$

Where, x_t is the position at time t, y_t is the spikes at time t.

Hidden Markov Model Decoding

A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. In this problem here, one important information is the relation between the last position and the next position of rats. Therefore, the HMM model is built to describe the problem more exclusively, the general architecture is shown in Fig. 1.

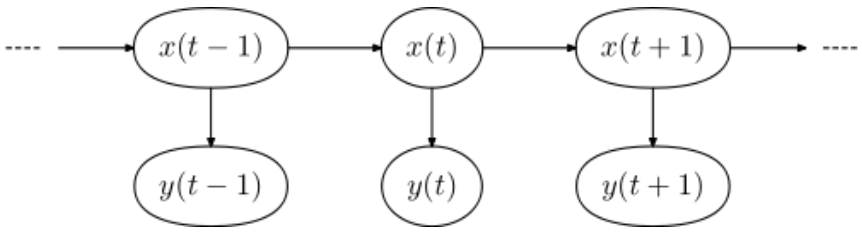


Figure 1. The Architecture of Hidden Markov Model. Here, $\mathrm{x}_t = \mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3 \ldots$ is the position of rats (hidden state), $\mathrm{y}_t = \mathrm{y}_1, \mathrm{y}_2, \mathrm{y}_3 \ldots$ is the neural spike of rats (observations).

The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states – called the Viterbi path – that results in a sequence of observed events. The Algorithm is described in equations following.

$$\mathrm{V}_{t,k}(x) = max_{x_1,x_2 \ldots x_{t-1}} \Pr(x_t, x_{t-1} \ldots x_1, y_t, y_{t-1} \ldots y_1 | k) \tag{4}$$

$$\mathrm{V}_{t+1,k}(x) = max_{x_1,x_2 \ldots x_{t-1}} \Pr(x_{t+1}, x_t \ldots x_1, y_{t+1}, y_t \ldots y_1 | k) = \max\{\mathrm{V}_t(x)\mathrm{a}(x) \Pr(\mathrm{y}_{t+1} | k)\} \tag{5}$$

Where $\mathrm{V}_t(x)$ is the probability of the most probable state sequence responsible for the first t observations that have k at its final state. The Viterbi path can be retrieved by saving back pointers that remember which state x was used in equation (5). Let $Ptr(x_t, t)$ be the function that returns the value of x used to compute $\mathrm{V}_{t,k}$ if $\mathrm{t} > 1$, or k if $\mathrm{t} = 1$, Then

$$x_T = argmax(\mathrm{V}_T(x)) \tag{6}$$

$$x_{t-1} = Ptr(x_t, t) \tag{7}$$

Recurrent neural network Decoding

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs, which is typically as the equation (8) and the model is shown in Fig. 2.

$$\mathrm{S}_k = f(\mathrm{S}_{k-1} * W_{rec} + X_k * W_x) \tag{8}$$

Where, S_k is the state at time k, X_k is an exogenous input at time k. W_{rec} and X_k are parameters like the weights parameters in feedforward nets.

The training procedure is basically derived from backward propagation, which is shown in equation (9).

$$\frac{\partial \varepsilon}{\partial S_{k-1}} = \frac{\partial \varepsilon}{\partial S_k} \frac{\partial S_k}{\partial S_{k-1}} = \frac{\partial \varepsilon}{\partial S_k} w_{rec} \tag{9}$$

And starts at:

$$\frac{\partial \varepsilon}{\partial y} = \frac{\partial \varepsilon}{\partial S_n} \tag{10}$$

Where, ε is the cost, n is the number of timesteps the network is unfolded. Note that only the recursive parameter that connects states w_{rec} plays a role in propagating the error down the network. Then, the gradients of the cost function with respect to the parameters can then be found by accumulating the parameter gradients while propagating the error. The update rules for the weights are shown in equation (11) and (12).

$$\frac{\partial \varepsilon}{\partial w_x} = \sum_{k=0}^{n} \frac{\partial \varepsilon}{\partial S_k} x_k \tag{11}$$

$$\frac{\partial \varepsilon}{\partial w_{rec}} = \sum_{k=0}^{n} \frac{\partial \varepsilon}{\partial S_k} S_{k-1} \tag{12}$$

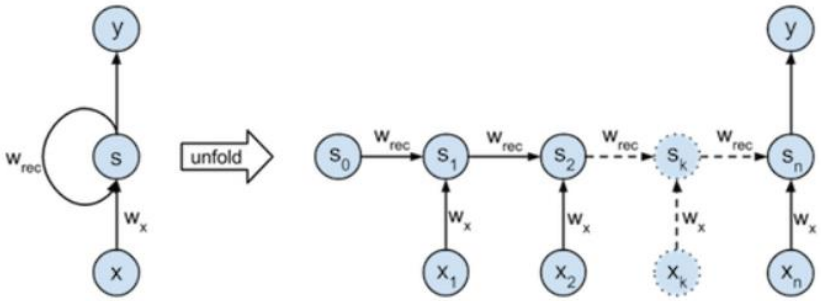


Figure 2. The Typical Linear Model of RNN. The left part is a graphical illustration of the recurrence relation. The right part illustrates how the network is unfolded through time over a sequence of length n.

RNN has been proved to be successful in the field of natural language processing and sequence analysis. Therefore, it should have more or less good performance in this decoding process (different from HMM, RNN is a discriminant model). Here, a naïve RNN is used to do the project.

Model Evaluation

For model evaluation, the error based on location is considered as equation (13).

$$\text{Error} = \frac{||f(x) - y||}{\max(y) - \min(y)} \times 100\% \tag{13}$$

Where, f(x) is the output of the model, y is the true recorded position of rats.

Parameter Setting

For Naïve Bayesian Model, the only parameter is the sliding window size. Here the window size is set as {10, 20, 30} as the rat speed and the feature number is considered (as p neruons can only code 2^p states).

For Hidden Markov Model, the transition matrix and emission matrix is calculated as discrete function. The emission probability is considered as multinomial and the start probability is considered as equal for each window. Also, the window size is set as {10, 20, 30} for better evaluation.

For Recurrent Neural Nework, considering the small amount of data, the structure is built as one hidden layer, one input layer and one output layer. All hidden layers are fully connected in one direction. Therefore, the structure of RNN and HMM is quite similar except that the HMM is a generative model and RNN is a discriminative model. As the training procedure is quite time-consuming (GPU is not used in this project), only 100 epochs training for one experiment. Also, the window size is set as {10, 20, 30}.

Result and Discussion

The overall result (average error except data7, for dataset 7 may include certain unexpected

error that the performance is too good to be true) is shown in Table 1.

Table 1. The Performance of Naïve Bayesian Model, Hidden Markov Model (HMM), Recurrent Neural Network (RNN) of 10, 20, 30 Window Size. Note that the error is calculated as equation (13) and take average, and dataset 7 is excluded (due to unconvincing result).

Window Size (cm)	10	20	30
Naïve Bayesian	48.21%	47.49%	48.72%
HMM	37.34%	35.07%	44.13%
RNN	28.51%	28.37%	28.34%

The visual presentation of the performance of three kinds of algorithm in dataset 1 (prediction versus true value) is shown in Fig. 3.

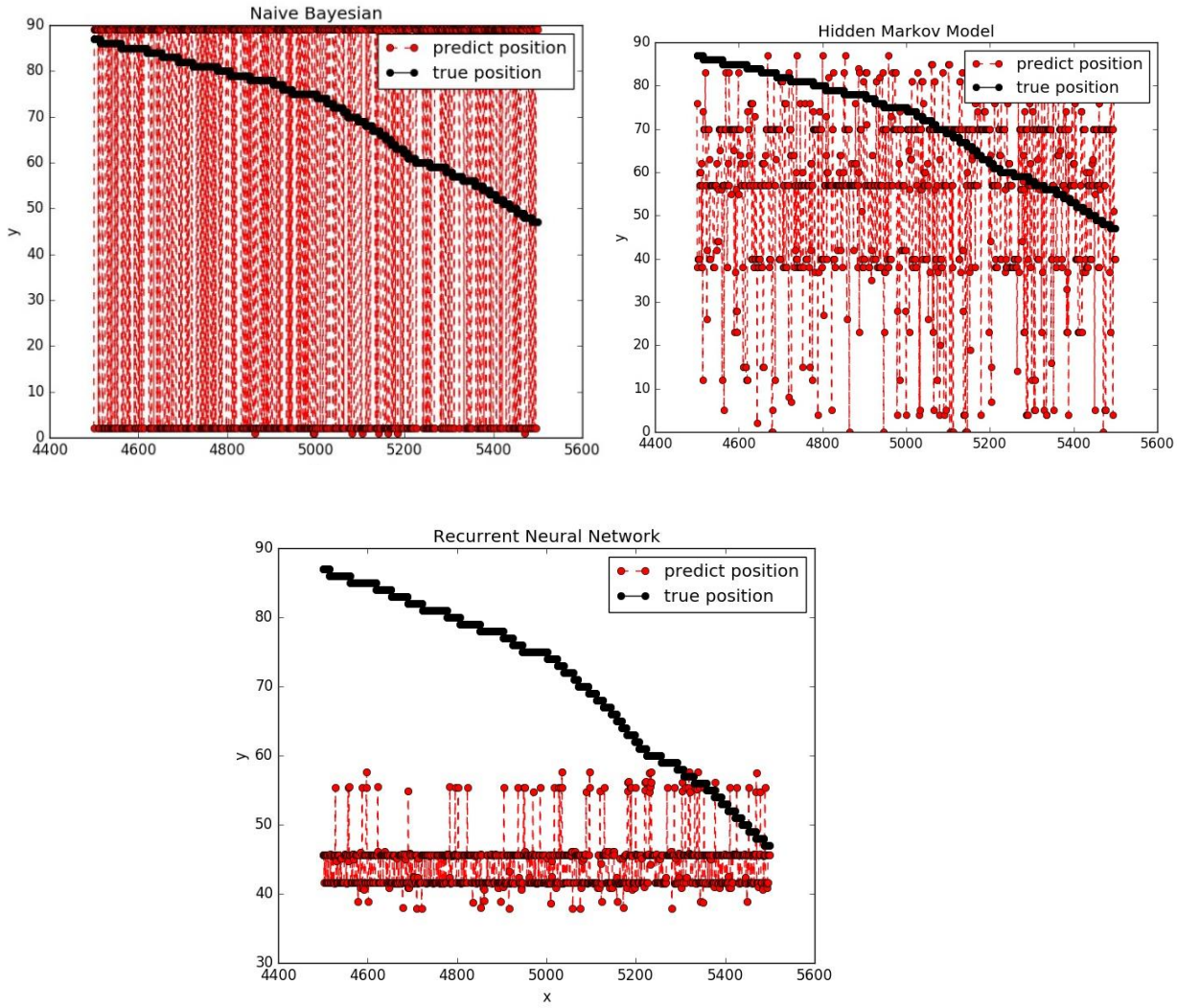


Figure 3. The Typical Example of the Performance of Naïve Bayesian Model, HMM and RNN. Window size is 20 cm, performed in dataset1. X-axis is the records, from 4500 to 5500. Y-axis is the window number. The up left figure is the performance of Naïve Bayesian Model, the up right is HMM and the bottom is RNN.

From Fig. 3, it is clear that there are actually two main problems occurring in Naïve Bayesian Model. One is that the value of the prior probability of the mouse position at two sides are too big so that the predicted position tends to be centered at two sides. One possible solution is to manually cut off two sides at very beginning and add them up in the end. Another solution is to give a penalty (which means another parameter to tune) to both ends of the position to improve the generalization. The other problem is that one important information loss during Naïve Bayesian Model is the correlation between each two steps. While the position at time t is explicitly related to position at time t-1, the equation (3) should be revised to equation (14).

$$\Pr(\mathrm{x}_t|\mathrm{y}_t) = \sum_{\mathrm{x}_{t-1}} \Pr(\mathrm{x}_t|\mathrm{y}_t, \mathrm{x}_{t-1}) \tag{14}$$

This means that this problem can be described as a kind of Markov process. Therefore, the HMM and RNN model are much appropriate for this sequence prediction problem. Nonetheless, one easier way to use Naïve Bayesian Model without dynamic programming in this problem is to train the data 3-step by 3-step, which is based on mini-batch method. This may slightly improve the performance.

Although HMM and RNN can greatly improve the performance in this analysis, the error (see Fig. 3) is still manifest. One possible explanation is that the feature dimension is still too small. As there are only five or six records of neural spike, this feature is considered as highly biased, which cannot divided very convoluted space. As is mentioned before, p neruons can only code 2^p states, so increase the recorded neurons may of vital help in this analysis. Another solution is to simply tune the parameter and experiment more. Increasing the training epochs, for example, may help improve the performance of RNN.

Reference

[1] Chen Z. An Overview of Bayesian Methods for Neural Spike Train Analysis[J]. Computational Intelligence & Neuroscience, 2013, 2013(4):107-111.

[2] http://hmmlearn.readthedocs.io/en/latest/api.html. API reference of hmmlearn.

[3] http://peterroelants.github.io/posts/rnn_implementation_part01/. How to implement a recurrent neural network.

[4] Brun V H, Solstad T, Kjelstrup K B, et al. Progressive increase in grid scale from dorsal to ventral medial entorhinal cortex[J]. Hippocampus, 2008, 18(12):1200–1212.

[5] David Heckerman. 1995. A Tutorial on Learning With Bayesian Networks.