Procedure

Basically, to train our model, we first preprocess and clean the training data so that the dataset only contains regular 5-character poems written in Tang dynasty. Then a multi-layer LSTM is trained to generate Tang style Chinese poems. To generate the whole poem, we use different models on the generating of first sentence and the rest sentences of the poem.

1. Data preprocessing and cleaning
2. First sentence generator
3. Following sentences generator

Thanks to LSTM’s ability of modelling long-term dependencies[1], we use a rule-based multi-layer LSTM to generate the following sentences of the poem. The structure of this generator is shown in Fig.1. This neural network is basically a sequence-to-character model, which models probability of a character given the preceding *n-*character sequence:

We use one-hot encoding to represent each Chinese character so that the model can be a generative model which outputs the probabilities of all the candidates. This makes it easier to apply tone pattern constrains on the poem, which will be discussed later.

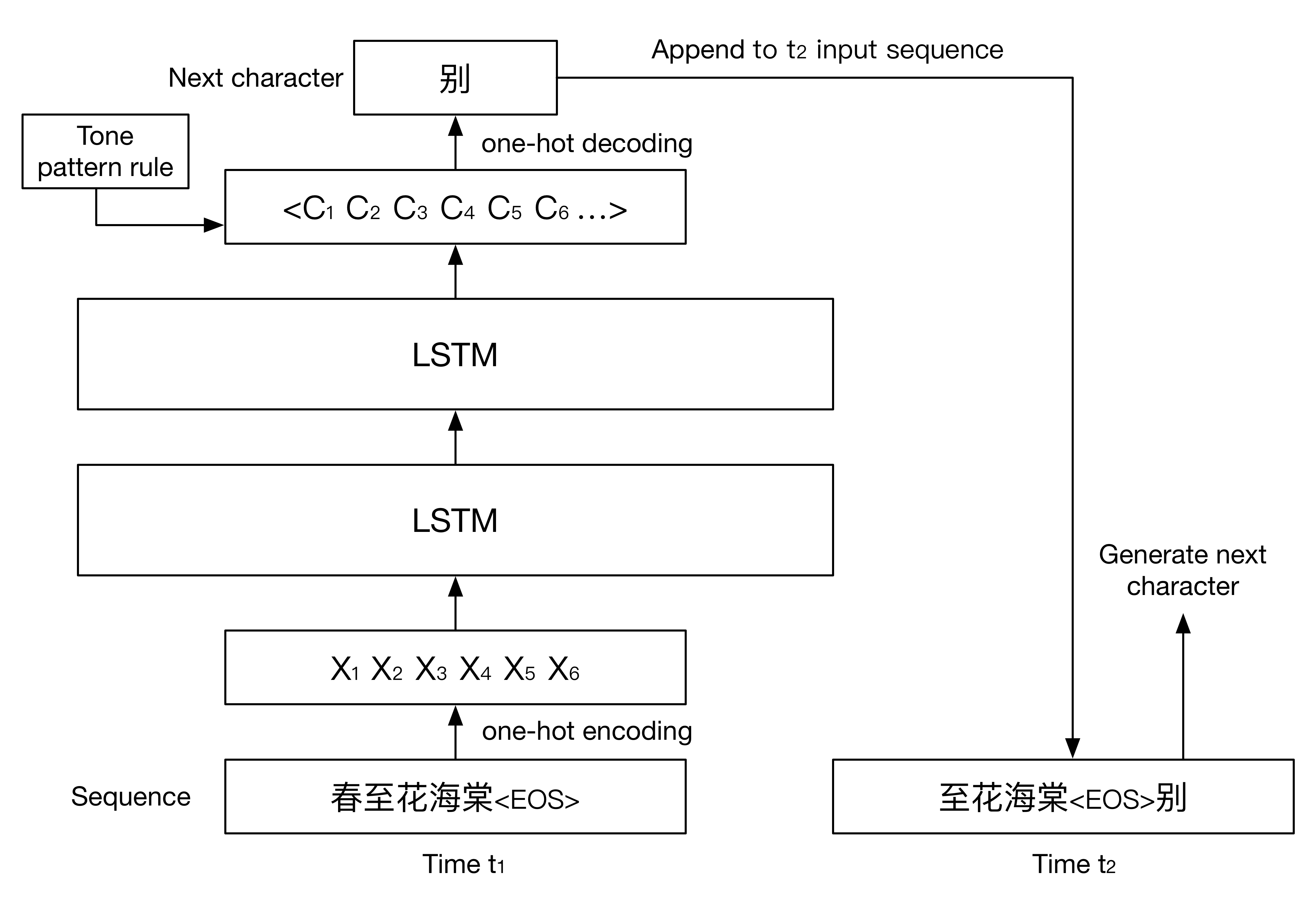


Fig. 1 Structure of following sentences generator with two layers LSTM

(Input sequence means “The spring is full of forest fragrance”. Output character means “Not”)

* 1. Training

All processed data containing about 16,000 5-character ancient Chinese poems written in Tang dynasty is used on training the model. We tried different number of layers and memory units to get the best result. Other configurations and hyper parameters are fixed: we use *softmax* as the activation function; loss is calculated through cross entropy; *Adam*[2] is used to optimize the learning rate; dropout is 0.2; sequence length is 6 (for better estimating each 5-character sentence since we also encode *<EOS>* as a character); batch size is 128.

* 1. Generating

To generate the following sentences given the first sentence which is also a 6-character sequence ended with *<EOS>*, we simply input the first sentence and use the most likely character as the next character and recursively generate the whole poem:

To make the poem more human alike, there are some rules we need to consider. First, we want the model learn to generate poem in rhyme, so we encode *<EOS>* to represent the end of sentence. However, *<EOS>* should never be output in the middle of each sentence, and must be output at the end. The other thing is we want the poems be in correct tone patterns. There are four different tone pattern templates for 5-character ancient Chinese poem. Under this constraint, the generator will choose the most likely character that also follow the tone pattern rule.

[1] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

[2] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13.