Efficiency Evaluation of Character-level RNN Training Schedules

Cedric De Boom, Sam Leroux, Steven Bohez, Pieter Simoens,
Thomas Demeester, Bart Dhoedt FIRSTNAME.LASTNAME@UGENT.BE
Ghent University – iMinds, INTEC, Technologiepark 15, 9052 Zwijnaarde, Belgium

Abstract

We present four training and prediction schedules for character-level recurrent neural networks. The efficiency of these schedules is tested in terms of model effectiveness as a function of training time and training data seen. We show that the choice of training and prediction schedule potentially has a considerable impact on the prediction effectiveness for a given training budget.

1. Introduction

Recurrent neural networks (RNNs) take data sequences of arbitrary length as input, map them to a hidden state, and then use this hidden state to make a prediction. This prediction can be a single value, but it can also be an entire sequence. In this paper we will study 'character-level RNNs', i.e. language models that learn to predict the next character of a text given the history of characters seen. At every time step a new character is fed into the RNN, which then transforms its hidden state to be able to predict the next character. These types of RNNs are also used in applications such as speech to text translation (Graves & Jaitly, 2014) and video frame tagging (Pigou et al., 2015). In the next section we will illustrate how such RNNs can be trained using different training and prediction schedules, and how this affects model efficiency.

2. Experimental setup

In all experiments from this section we will use the following neural network architecture: Input (65 dimensions) – LSTM layer (50 dimensions, input/output/forget gate non-linearity: sigmoid, cell non-linearity: tanh) – Dense layer (65 dimensions, softmax non-linearity). The number of input and output dimensions is 65, since there are 65 different characters in our text corpus, and we encode every character in a one-hot representation. We choose to use a Long Short-Term Memory (LSTM) layer instead of a simple recurrent connection as this is currently state of the art in many text mining tasks (Hochreiter & Schmidhuber, 1997; Greff et al., 2015).

To train this neural network we use Truncated Backprogratation Through Time (TBPTT). Every k_1 time steps the gradients are backpropagated for k_2 time steps in the past (we use the same parameter notation as in (Sutskever, 2013)). To allow for temporal dependencies in the input sequence between every gradient update, we can additionally ensure that $k_1 \leq k_2$. The choice of these parameters can be a trade-off between computation time and model effectiveness. To calculate the gradients in the TBPTT, we use the categorical cross entropy as loss function.

Apart from choosing these parameters, there is a choice of how to implement the training and prediction procedures, which can influence both model effectiveness and training speed. Below we list four different schedules for the character-level RNN. A visualization of each of the schedules is shown in Figure 1.

Schedule 1 – In this schedule we take sequences of fixed length k_2 as input for training, and for every input character in such a sequence we predict the next character. This leads to k_2 losses that we backpropagate through the network. The hidden and cell states in the LSTM layer are reset to their initial values for every input sequence. These initial states are learned through backpropagation. At test time, we take an entire sequence of length k_2 as input, and we ask to predict the single next character.

Schedule 2 – This schedule is similar to the first one, but instead of making a prediction for every input character at train time, we now only predict the final next character. We therefore only have a single loss that is backpropagated. At test time, the procedure is the same as in schedule 1.

Appearing in *Proceedings of Benelearn 2016*. Copyright 2016 by the author(s)/owner(s).

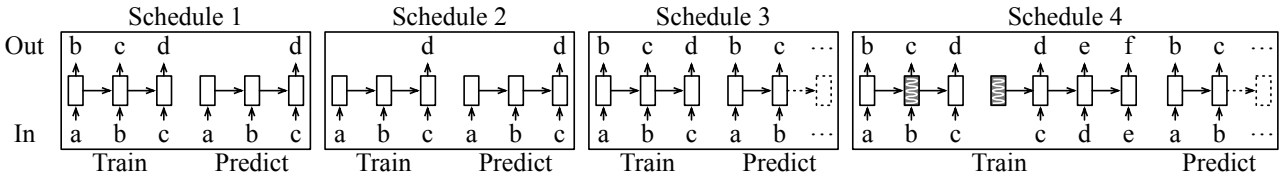


Figure 1. Four different schedules of training and predicting from a character-level RNN.

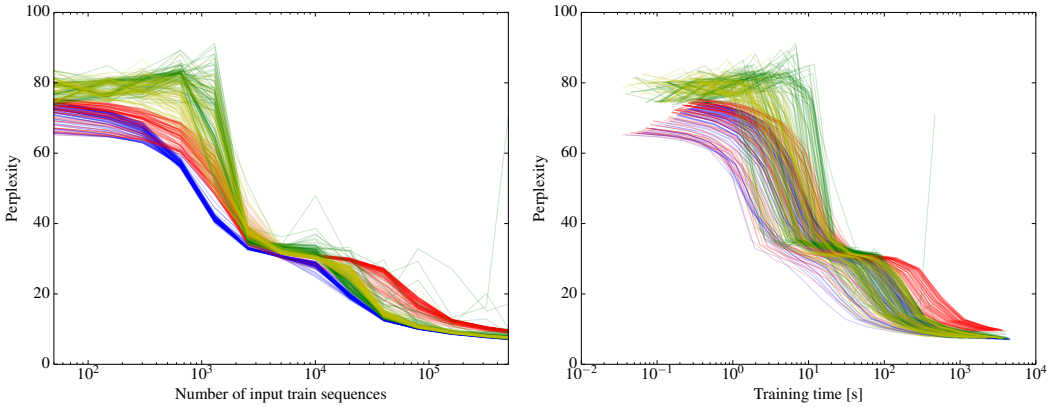


Figure 2. Perplexity as a function of number of input train sequences and elapsed training time. Blue: schedule 1; red: schedule 2; green: schedule 3; yellow: schedule 4.

Schedule 3 – Here, the training procedure is the same as in the first schedule. At test time we start predicting using the learned initial hidden state, and we use the predicted character as input in the next time step while we continue to update the hidden state.

Schedule 4 – At train time, the initial hidden state for a new input sequence is reused from the previous input sequence. More specifically, as initial hidden state for the input character at time t we use the hidden state produced by input character at time $t-1$ from the previous input sequence. The prediction procedure in this final schedule is the same as in schedule 3.

We evaluate all four schedules in terms of perplexity vs. training time and training input sequences: the most efficient implementations should reach a lower perplexity faster than the other ones. Perplexity is defined as: $\exp\left(\sum_{k=1}^{N} P(c_k|c_{1:k-1})/N\right)$, in which c_k is the k'th character, and N the total number of characters. Every implementation is run 100 times with random settings for the k_2 and k_1 parameters: $5 \leq k_1 \leq k_2 \leq 50$. This is all done on the same hardware, for a total of 500,000 train sequences. We employ Adam gradient updates (Kingma & Ba, 2015) with batch size 50 across all experiments, and we use Lasagne as implementation framework (`github.com/Lasagne/Lasagne`). Our dataset consists of excerpts from Shakespearian plays; the train set has around 1,100,000 characters, and the test set around 11,000.

In Figure 2 we see that all settings for schedules 1 and 2 converge smoothly towards an optimum, but schedule 1 is more efficient. Schedules 3 and 4 have noisy behaviour in the beginning of the training phase. After that all settings for schedule 4 seem to converge to the same optimum, but some parameter settings for schedule 3 continue to behave very noisily. Schedule 1, however, seems to be performing best and most consistently. We observe that the settings for which the lowest perplexity is reached the fastest, all have a small $k_2 < 10$. Since $k_1 \leq k_2$, this means that frequent model updates over short input sequences are preferred.

3. Conclusion

We tested the efficiency of multiple training and prediction schedules for character-level recurrent neural networks, in terms of model effectiveness as a function of training time and training data seen. We observed that the choice of a particular schedule can considerably impact the efficiency of the model. It also turns out that training over short input sequences and with frequent model updates is most efficient. Further research is however required to verify if these conclusions hold for more complex models and other datasets.

Acknowledgements

This work has been presented at the Data-Efficient Machine Learning workshop at ICML 2016, New York, USA. We thank Pontus Stenetorp for his useful feedback on drafts of this paper. We also acknowledge Nvidia for its generous hardware support. Cedric De Boom is funded by a PhD grant of the Flanders Research Foundation (FWO). Steven Bohez is funded by a PhD grant of the Agency for Innovation by Science and Technology in Flanders (IWT). Experimental code and data can be found on: `https://github.com/cedricdeboom/CharRNN`.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2015) Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR*.

Graves, A., & Jaitly, N. (2014). Towards End-To-End Speech Recognition with Recurrent Neural Networks. *ICML*.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2015). LSTM: A Search Space Odyssey. *arXiv.org*.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*.

Kingma, D., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *ICLR*.

Pigou, L., Van Den Oord, A., Dieleman, S., Van Herreweghe, M., & Dambre, J. (2015). Beyond Temporal Pooling: Recurrence and Temporal Convolutions for Gesture Recognition in Video. *arXiv.org*.

Sutskever, I. (2013). *Training recurrent neural networks*. Doctoral dissertation.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *NIPS*.