Recurrent neural network as a language model - technical report

Wojciech Zaremba WOJ.ZAREMBA@GMAIL.COM
New York University

Abstract

Recurrent neural networks (RNN) offer a powerful framework to learn any arbitrary dependency. They are expressive as a finite memory Turning machine. However, their training is difficult and computationally expensive.

This technical note focuses on training RNNs for modeling language at the character level. We provide set of pragmatic recommendations about how to train a simple one layer RNN for this task. Moreover, we provide CPU and GPU Theano (Bergstra et al., 2010) code which reproduces results close to state-of-the-art on the Penn Treebank Corpus.

1. Introduction

Neural networks (NN) are stacked linear transformations alternated with non-linearities. Currently, state-of-the-art results in many computer vision tasks are achieved with feed-forward neural networks of this type. In feed-forward networks, computation flows in one direction from the input layer to the output layer(s). Recurrent neural networks (RNN) contain connections between instances of feed forward networks shifted in time. Such connections allow them to maintain *memory*, and perform prediction dependent on a history. Based on current advances in computer vision thanks to feed-forward networks, we are optimistic that models heavily utilizing RNNs can superior results to the current state-of-the-art on NLP tasks. Moreover, we believe that they might be crucial for further advances in computer vision (attention based models, and video prediction).

A common setting for RNNs is the prediction of the next element in a sequence. The input is a single element of a sequence, and a previous state. The network attempts at every stage to predict next element of sequence. We examine these models for language modelling and for simplicity, we constrain ourselves to a character level language model.

The typical training procedure for RNNs is stochastic gradient descent (SGD). However, it is difficult to obtain effective RNN models by applying unconstrained SGD. Recurrency brings much higher expressive power compared to feed-forward networks, but also makes them more difficult to train. There are several well-known issues: (1) vanishing gradient; (2) exploding gradient and (3) short memory. We address exploding gradient issue by clipping gradients, but don't tackle the remaining problems.

We proceed by presenting related work (Section 2). Next, we describe our framework (Section 3), and finally we present experimental results (Section 4). Code reproducing experiments (and train any arbitrary RNN on CPU or GPU) is available online[1].

2. Related work

There has been extensive interest in different flavours of neural networks with recurrent connections (Hopfield, 1982; Hinton et al., 2006). These approaches consider recurrency not to account for time dependency, but for internal data dependency (e.g. correlation between pixel values).

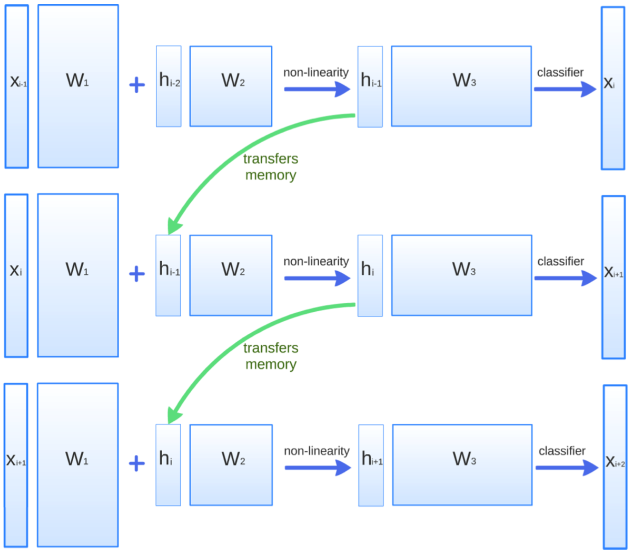
In this note, we are mainly interested in RNNs which aim to predict a temporal sequence. (Mikolov, 2012) (Sutskever, 2013) consider training of such networks at the character and word level. Moreover, they analyze how best optimize such models (e.g. with Hessian-free method, or by clipping gradients).

(Graves, 2013) shows how the memory of the model may be extended by using Long-Short-Term-Memory units (LSTSs). Some evidence is shown that LSTMs prevent gradients from vanishing.

3. Framework

Our code is build in Theano. This Python framework permits the definition of symbolic expressions, and their automatic differentiation. Moreover, it compiles code to fast

[1] https://github.com/wojzaremba/rnn



(a) This figure presents the information flow in a recurrent network based on the simplest possible neural network. Experiments in this paper have been perfomed on such networks, where non-linearity is a sigmoid, and classifier is a softmax. We consider models with hidden layer of the size $200, 600, 1000$, and input is restricted to ASCII characters. The input can be represented as an indicator vector having 256 dimensions, however in practise is better to store as an uint8.

CPU or GPU executable versions. We now present the setup for our best model.

We train a simple RNN as shown in Figure 3. Our best model has 600 hidden units. We initialize all weights randomly from a zero-mean Gausssian with variance 0.001, and all biases are set to 0. When switching between different instances of text, the hidden units should be reset to all ones (this works better than all zeros). We train the model with SGD on minibatches of size 10, and we unroll RNN for 10 steps. We don't use any momentum (for such a small model it makes little difference). We clip gradients above an ℓ_2 norm of 15, to have a norm of 15. The initial learning rate is 0.1. We decrease it by factor of 2 every time when perplexity on a validation set increases (we measure perplexity in every epoch). It takes around $5 - 8$ epochs to decrease the learning rate. A single epoch takes around 20 minutes to compute. If perplexity increases for 2 epochs then we finish training (early stopping criteria).

The final layer is a softmax. This classifier gives probabilities of ocurrence for next character. During sequence generation , we sample from this probability.

4. Experiments

First, we use synthetic data to show experiments that verify if our model has any memory. Then we trained it Penn Treebank Corpus[2].

Generated sequence from input sequence "a"
abbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccdd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
abbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccdd
adaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbc

Table 1. Toy example showing the memory of the RNN. See text for details.

Generated sequence from input sequence "aabb"
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd

Table 2. Given an initial sequence "aabb", the subsequent sequence is unambiguous.

4.1. Synthetic data

Our first experiment demonstrates the memory of the model. A network is trained on sequences "aabbccddaabbccdd...", starting from any arbitrary position. In order to give a proper prediction of the next character, the network has to remember what is the current position, i.e if it is a first "a", or a second "a". Starting from any random position (with clean hidden state), for first two predictions the network is uncertain about next character. However, after two predictions, all subsequent predictions are deterministic and the network learns to predict them correctly. Very short networks can obtain a perplexity close to one on this task. Moreover, we can condition on initial part of sentence, and generate rest of it. Tables 1, 2 shows exemples of generated sequences.

4.2. Penn Treebank Corpus

After 24 epochs of training (8h) we have achieved a perplexity on the test data of 2.94. This compares favorably with (Mikolov, 2012) which reports a perplexity of 2.6 from their best model. Table 3 presents generated sentences for our model.

5. Discussion

Although the RNN language model generates impressively realistic text, it is still far from human generated text. It is crucial to understand what components are the missing to close the gap to human performance. It might be an optimization problem, or maybe a missing computational unit problem. We would like to understand these challenges on the large scale datasets both in the context of NLP, as well as computer vision tasks.

[2] http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz

Generated sequence from input sequence “My name is”
My name is relatively ms. <unk> hill which i limit it social it can
My name is r. <unk> eurocom <unk> to <unk> more <unk>
My name is <unk> on smaller sales are clearance by long-term alterna
My name is reinvestment on senate is a voting oil co. his claim of s
My name issues culming was having a market acknowledged that it is in

Table 3. Most of generated words are correct English words. Moreover, they are combined in an approximately grammatical way.

6. Acknowledgment

Thanks are due to Ilya Sutskever, Tomas Mikolov and Caglar Gulcehre for insightful discussions on how to train RNNs.

References

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Graves, Alex. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Hinton, Geoffrey E, Osindero, Simon, and Teh, Yee-Whye. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Hopfield, John J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8): 2554–2558, 1982.

Mikolov, Tomáš. *Statistical language models based on neural networks*. PhD thesis, Ph. D. thesis, Brno University of Technology, 2012.

Sutskever, Ilya. *Training Recurrent Neural Networks*. PhD thesis, University of Toronto, 2013.