Recurrent neural network as a language model - technical report

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

Rob Fergus FERGUS@CS.NYU.EDU
New York University
Facebook AI Group

Abstract

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

This work focuses on training RNNs for character level language modelling task. We provide set of pragmatic recommendations about how to train a simple, one layer RNNs for such task. Moreover, we support our statements with Theano (Bergstra et al., 2010) code which reproduces close to state-of-the-art results on Penn Treebank Corpus (code executable both on CPU and GPU).

1. Introduction

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

Common setting for RNN is a prediction of next element in a sequence. Input is a single element of a sequence, and a previous state. Consecutive elements of sequence are fetched by RNN. Network attempts at every stage to predict next element of sequence. We examine such setting on task of language modelling. For purpose of simplicity, we constrain ourselves to character level language modelling.

Typical training procedure for RNNs is stochastic gradient descent (SGD). However, it is difficult to obtain a valuable models of RNNs by applying unconstrained SGD. Recurrency brings much higher expressive power in compare feed-forward networks, and in the same time it makes them more difficult to train. There are several common issues (1) vanishing gradient, (2) exploding gradient, (3) short memory. We address exploding gradient issue by clipping gradients, and we don't tackle 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 allowing to reproduce experiments (and train any arbitrary RNN on CPU or GPU) is available online[1].

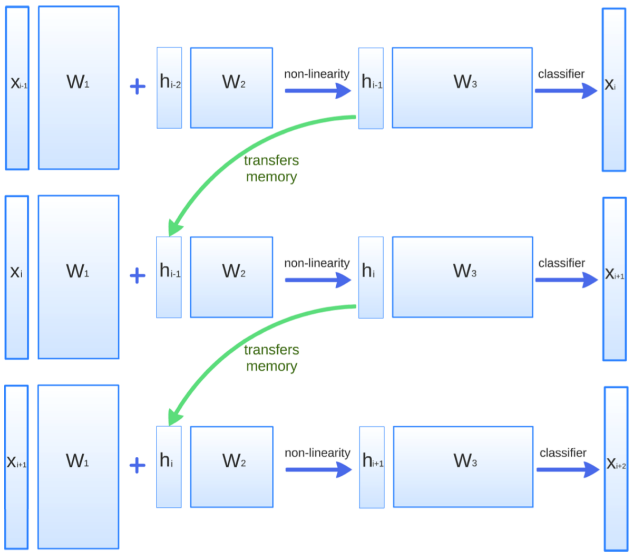
2. Related work

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

We are mainly interest in RNNs, which aim to predict time-dependent sequence. (Mikolov, 2012) (Sutskever, 2013) consider extensively training of such networks for characters, and words. Moreover, they analyze how best optimize such models (e.g. with Hessian-free method, or by clipping gradients).

(Graves, 2013) focuses on how to extend memory of model. He addresses it by using Long-Short-Term-Memory units

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



(a) Figure presents information flow in 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 layers of the size $200, 600, 1000$, and input is always a ASCII character. Input can be represented as an indicator vector having 256 dimensions, however in practise is better to story it as int8.

(LSTSs). There are weak evidences that LSTMs prevent gradient from vanishing.

3. Framework

Our code is build in Theano. This python framework allows to define python symbolic expressions, and differentiate with respect to them. Moreover, it compiles code to fast CPU or GPU executable versions. We present here setup for our best model.

We have trained a simple RNN as depicted on the Figure 3. Our best model has 600 hidden units. We initialized all weights with Gausssian having 0.001 variance, and all biases with 0. We train model with SGD on minibatches of size 10, and we unroll RNN for 10 steps. We don't use any momentum (for such small models it doesn't make a big difference). We clip gradients above norm 15 (if norm of a gradient is higher than 15 then we project it down). Our 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 learning rate. Single epoch computes for 20 minutes. If perplexity increased for 2 epochs then we finish training (early stopping criteria).

Generated sequence from input sequence "a"
abbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccdd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
abbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccdd
adaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbc

Table 1. Toy example showing that network has memory. As it generates sequences starting from "a" there is ambiguity.

Generated sequence from input sequence "aabb"
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd
aabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccddaabbccd

Table 2. Initial sequence "aabb" determines further sequence unambiguously.

4. Experiments

First, we designed experiments to verify that our model has any memory. Next we trained it Penn Treebank Corpus[2].

4.1. Synthetic data

Our first experiment verifies that our model has any memory. I have trained network on sequences "aabbccddaabbccdd. . ." starting from any arbitrary random position. In order to give a proper prediction of next character network has to remember what is the current position. If it is a first "a", or a second "a". Starting from any random position (with clean hidden state), for first two predictions network is uncertain about next character. However, after two predictions, all next predictions are deterministic. Network learns to predict them correctly. Very shortly networks gets 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 exemplary generated sequences.

4.2. Penn Treebank Corpus

After 19 epochs of training (6h) we have achieved perplexity on the test data 3.00. Tomas Mikolov reports perplexity 2.6 on his best model. Table 3 presents generated sentences for our model.

5. Discussion

Although language model generated by RNNs is impressive, still it is far from human generated text. It is crucial to understand what are the missing components to achieve closer to human performance. It might be optimization problem, or maybe missing computational unit problem. We would like to understand this challenges on the large scale dataset both in the context of NLP, as well as com-

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

Generated sequence from input sequence "My name is"
My name is profitable mr. <unk> northern new protecting earnings
My name is pg according to a roller-based in the world the <unk>
My name is <unk> of primary sales bridge brokers rose N N to N a
My name is playing improve new shareholders fell N N in october
My name issues cud holds of <unk> abortion agreement bill one of

Table 3. Most of generated words are correct English words. Moreover, they are combined in close to a grammatical way.

puter vision tasks.

6. Acknowledgment

Thank you to Ilya Sutskever, Tomas Mikolov, 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.