CS281 Project Proposal

Bernd Huber (bhb@seas), Sebastian Gehrmann (gehrmann@seas),
Leonhard Spiegelberg (spiegelberg@g)

Abstract

All languages from programming languages or XML to natural languages are deeply nested. Finite state systems like neural networks cannot handle the potentially infinite nestings of these languages, although they are commonly used for language modelling. Thus, researchers have been trying for language models to account for deep nestings by giving them a memory.
The most popular model for language modelling has, in recent times, been the Long Short-Term Memory Neural Network (LSTM), an improved form of the Recurrent Neural Network (RNN). We aim to contribute towards the understanding of when and how LSTM fail to recognize nestings. Additionally, we want to improve LSTM performance to keep track of nestings (e.g. through intelligent parameter choice).

Methods

Our project will focus on when and what type of nesting errors occur when using LSTMs. We aim to control the training size and sample size and classify the types of nesting errors that occur (ie. which information an LSTM tends to forget first).
1) To do this, we need to generate a simple to evaluate, but deeply nested language. As a preliminary approach, we are thinking of using a Dyck language (only uses the vocabulary '(',')','{','}','[',']' and parenthesis nesting rules). We will generate this language and test/training data using python. To simulate a real nested language, we will also try a language with input on each nesting level (e.g. a random number of characters or a word in addition to more parentheses).
2) We want to use LUA with the torch framework to build a character-based LSTM based on Kaparthy's Char-RNN (https://github.com/karpathy/char-rnn). We will probably have to train the model using the Odyssey Cluster.
No one in our group has experience with either LUA/Torch or Odyssey, so we can't give too many details about our implementation plan in that regard.
3) We also want to compare different language models and parameterization for these. Thus, we will implement or use existing implementations for LSTMs and GRUs and compare the results.

Experiments

A first experiment we intend to run is to try and train LSTMs with an increasing amount of cells and then look at the following:

a. What does each individual cell learn?
b. How is the overall performance of the LSTM based on cell number?

Repeat this experiment with same/different training-corpus and look at:

a. Is there a difference in what a cell learns (for LSTMs with same cell number)?
b. Look at outliers in performance (if there are some) and evaluate, why?

After looking at the result, we will formulate further experiments as needed.

Related Work

Our work is based on the analysis of Kaparthy et al. in the paper “Visualizing and Understanding Recurrent Networks” in which the authors analyze cells and typical mistakes LSTMs make. They also show that the memory of a LSTM is way longer than its context, which is relevant for our work. Kaparthy further expands the analysis in his blog post “The Unreasonable Effectiveness of Recurrent Neural Networks” (May 2015) in which he trains a character-based LSTM to generate various different sources of text. We intend to, as opposed to Kaparthy et al., analyze one common mistake more in-depth. We can still use their methods for our work.

On a more technical side, there is a recent paper by Jozefowicz et al., named “An Empirical Exploration of Recurrent Network Architectures” which compares over 100 different architectures for neural network for language models. We can use their results to have a baseline to compare with.

Recently, LSTMs got a lot of attention in blog articles, including “Understanding LSTM Networks” (August ‘15), “Recurrent Neural Networks Tutorial” (September ‘15), and “Implementing a Neural Network from Scratch in Python” (March ‘15) which offer insights about how they work on a technical basis and how to implement them.

In order to understand, how to optimize LSTMs, we also consider “Distributed Representations of Words and Phrases and their Compositionality” by Mikolov et al. and “word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method” by Goldberg et al.) to optimize the input vector.

Finally, to get an overview over all different models that we will compare in our analysis, we consider the very detailed document “A Primer on Neural Network Models for Natural Language Processing” by Goldberg et al. which explains everything about RNNs on more than 70 pages

Collaboration

Bernd Huber	Test language definition and generation, Ground Truth, Definition of Error Classes (and development of test scripts for these)
Sebastian Gehrmann	Initial Implementation using LUA/Torch, training on Odyssey Cluster
Leonhard Spiegelberg	Error classification and model testing,

After the implementation of the first model, we will divide further work so that everyone contributes equal amount of time.

Double Dipping

While not technically double dipping, one team member (Sebastian Gehrmann) works with Stuart Shieber on learning about language models for a 299r this fall. We intend to talk about this research project, its progress and its results with Stuart. Since the project involves language modelling, we also intend to consult Alexander Rush about using the Odyssey-Cluster and LUA/Torch. During the creation of this proposal we talked to both of them and both agreed to help framing this project.