Language Modelling with LSTM [LM2]

Luca Danilo Melis (221698)
University of Trento
Via Sommarive, 9, 38123 Povo,Trento TN
lucadanilo.melis@studenti.unitn.it

Abstract

This work focuses on the well-known task of language modelling. This represents a relatively important challenge when doing Natural Language Processing applications. This report details the realization of a Recurrent LM module based on long short-term memory (LSTM) using the Penn TreeBank dataset. It also shows the implementation, the performance obtained and the results.

1 Introduction

Language modeling (LM) is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence. Language models analyze bodies of text data to provide a basis for their word predictions. They are used in natural language processing (NLP) applications, particularly ones that generate text as an output. Some of these applications include machine translation and question answering.

Language models determine word probability by analyzing text data. They interpret this data by feeding it through an algorithm that establishes rules for context in natural language. Then, the model applies these rules in language tasks to accurately predict or produce new sentences. The model essentially learns the features and characteristics of basic language and uses those features to understand new phrases. There are several different probabilistic approaches to modeling language, which vary depending on the purpose of the language model. From a technical perspective, the various types differ by the amount of text data they analyze and the math they use to analyze it.

Some common statistical language modelling types are:

N-gram. N-grams are a relatively simple approach to language models. They create a probability distribution for a sequence of *n* The *n* can be any number, and defines the size of the "gram", or sequence of words being assigned a probability.

Unigram. The unigram is the simplest type of language model. It doesn't look at any conditioning context in its calculations. It evaluates each word or term independently. Unigram models commonly handle language processing tasks such as information retrieval. The unigram is the foundation of a more specific model variant called the query likelihood model, which uses information retrieval to examine a pool of documents and match the most relevant one to a specific query.

Bidirectional. Unlike n-gram models, which analyze text in one direction (backwards), bidirectional models analyze text in both directions, backwards and forwards. These models can predict any word in a sentence or body of text by using every other word in the text. Examining text bidirectionally increases result accuracy. This type is often utilized in machine learning and speech generation applications. For example, Google uses a bidirectional model to process search queries.

Exponential. Also known as maximum entropy models, this type is more complex than n-grams. Simply put, the model evaluates text using an equation that combines feature functions and n-grams. Basically, this type specifies features and parameters of the desired results, and unlike n-grams, leaves analysis parameters more ambiguous -- it doesn't specify individual gram sizes, for example. The model is based on the principle of entropy, which states that the probability distribution with the most entropy is the best choice. In other words, the model with the most chaos, and least room for assumptions, is the most accurate.

Exponential models are designed maximize cross entropy, which minimizes the amount statistical assumptions that can be made. This enables users to better trust the results they get from these models.

Continuous space. This type of model represents words as a non-linear combination of weights in a neural network. The process of assigning a weight to a word is also known as word embedding. This type becomes especially useful as data sets get increasingly large, because larger datasets often include more unique words. The presence of a lot of unique or rarely used words can cause problems for linear model like an n-gram. This is because the amount of possible word sequences increases, and the patterns that inform results become weaker. By weighting words in a non-linear, distributed way, this model can "learn" to approximate words and therefore not be misled by any unknown values. Its "understanding" of a given word is not as tightly tethered to the immediate surrounding words as it is in n-gram models.

The scope of this project is to provide a simple LSTM network capable of been used as Language Model. Ultimately we discuss the obtained result and strength and limitation.

2 Long short-term memory (LSTM)

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural network, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video).

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell. Three gates input, output and forget are computed:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f\right),
\end{aligned}$$

Where σ is a sigmoid function x tis the input at the t-th timestep. The memory cell is computed:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh\left(\mathbf{W}_c \mathbf{x} + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c\right)$$

Where o is an element-wise multiplication. This adaptive leaky integration of the memory cell allows the LSTM to easily capture long-term dependencies in the input sequence, and this has recently been widely adopted many works involving language models.The output, or the activation of this LSTM layer, is then computed as

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t).$$

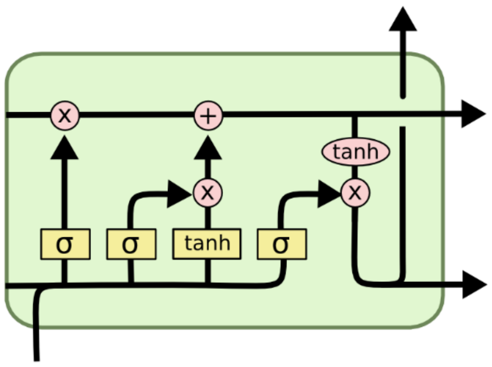


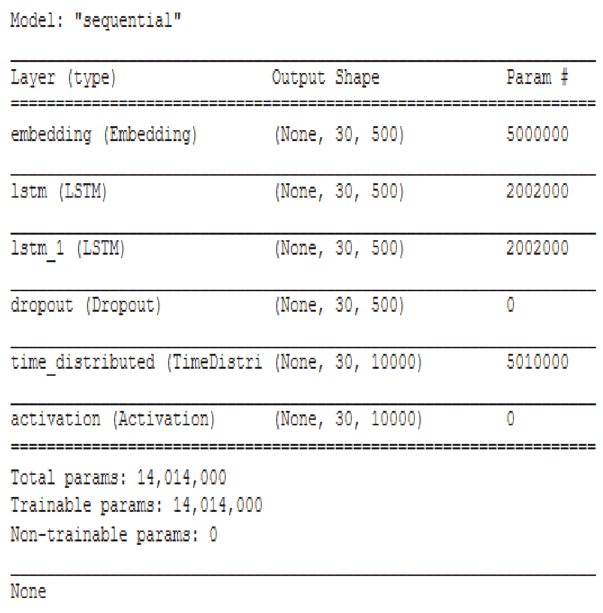
Figure 1: LSTM cell can process data sequentially and keep its hidden state through time.

3 Data Analysis

The dataset used for training and testing the model is the Penn Treebank. PTB in its eight years of operation (1989-1996), produced approximately 7 million words of part-of-speech tagged text, 3 million words of skeletally parsed text, over 2 million words of text parsed for predicate argument structure, and 1.6 million words of transcribed spoken text annotated for speech disfluencies. The material annotated includes such wide ranging genres as IBM computer manuals, nursing notes, Wall Street Journal articles, and transcribed telephone conversations, among others

Word-level PTB does not contain capital letters, numbers, and punctuations, and the vocabulary is capped at 10k unique words, which is relatively small in comparison to most modern datasets which can result in a larger number of out of vocabulary tokens. Every sentence is divided by a line and inside of it there are annotated <unk> tag.

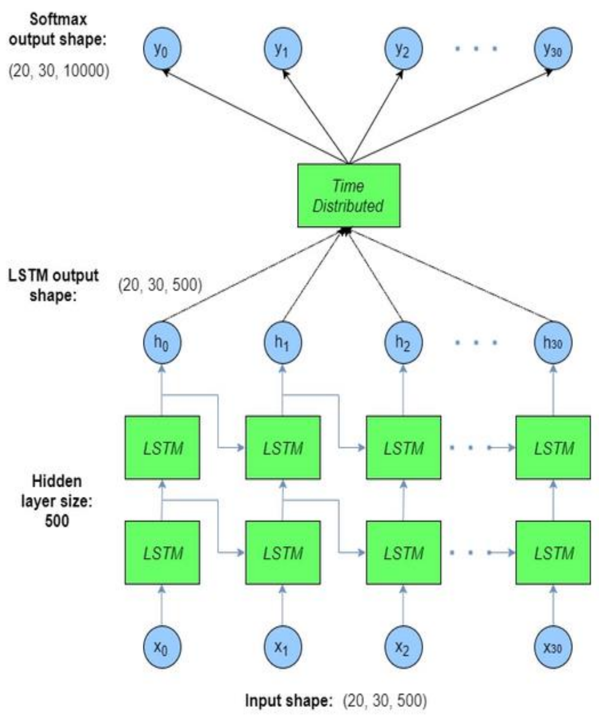
4 Model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 30, 500)           5000000
_________________________________________________________________
lstm (LSTM)                  (None, 30, 500)           2002000
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 500)           2002000
_________________________________________________________________
dropout (Dropout)            (None, 30, 500)           0
_________________________________________________________________
time_distributed (TimeDistri (None, 30, 10000)         5010000
_________________________________________________________________
activation (Activation)      (None, 30, 10000)         0
=================================================================
Total params: 14,014,000
Trainable params: 14,014,000
Non-trainable params: 0
_________________________________________________________________
None
```

The architecture is implemented with the keras libraries of python. The input shape of the text data is ordered: (batch size, number of time steps, hidden size). In other words, for each batch sample and each word in the number of time steps, there is a 500 length embedding word vector to represent the input word. These embedding vectors will be learnt as part of the overall model learning. The input data is then fed into two “stacked” layers of LSTM cells (of 500 length hidden size). The output from these unrolled cells is still (batch size, number of time steps, hidden size). This output data is then passed to a Keras layer called TimeDistributed. This function adds an independent layer for each time step in the recurrent model. The activation for these dense layers is set to be softmax in the final layer of our Keras LSTM model.

Finally, the output layer has a *softmax* activation applied to it. This output is compared to the training *y* data for each batch, and the error and gradient back propagation is performed from there in Keras. As optimizer is chosen the Adam algorithm



5 Experiment and Result

The experiments made involved the use of the text based PTB dataset so with the sentences written not in char form. The data are subject to a pre-processing step. To get the text data into the right shape for input into the LSTM model, each unique word in the corpus must be assigned a unique integer index. Then the text corpus needs to be re-constituted in order, but rather than text words we have the integer identifiers in order.

The three functions which do this in the code are *read_words, build_vocab and file_to_word_ids*. They first split the given text file into separate words and sentence based characters. Then, each unique word is identified and assigned a unique integer. Finally, the original text file is converted into a list of these unique integers, where each word is substituted with its new integer identifier. This allows the text data to be consumed in the neural network. Then load the data for each dataset file txt. Build the complete vocabulary, then convert text data from the train, validation and test dataset to list of integer.

A generative batch class for keras is initialized. This object takes the data as the first argument, the next argument supplied is called num_steps(this is the number of words that we will feed into the time distributed input layer of the network), the batch size and skip steps. Then is defined the method generator called during the training. This function creates the output arrays x and y. After a first check of the index is performed, the input data is consumed into the x array. The data indices consumed are the current index to the current-index-plus-num_steps number of words. Next, a temporary y variable is populated which works in pretty much the same way, the only difference is that the starting point and the endpoint of the data consumption is advanced by 1. After we define the model, it is trained for 40 epochs. The model is evaluated based on perplexity, categorical accuracy, and loss. For calculating the training perplexity the loss is exponentiate.

During training, the loss and the perplexity decrease significantly but remain high. In the last epoch the perplexity is 3596.5088, but as the best result in 1759.6830. The categorical accuracy as its best in the last epoch 0.42. The evaluation of the test set has 0.22 accuracy and 6.85 loss. The model trained is not optimal for the purpose.

References

Jurafsky, D. and Martin, J. H. Speech and Language Processing. Chapter 3: N-gram Language Models *(Draft)* (2019).

Koehn, P. Language Modeling (II): Smoothing and Back-Off (2006). Data Intensive Linguistics *(Lecture slides)*

Vajapeyam, S. Understanding Shannon's Entropy metric for Information (2014).

Iacobelli, F. Perplexity (2015) YouTube

Lascarides, A. Language Models: Evaluation and Smoothing (2020). Foundations of Natural Language Processing *(Lecture slides)*

Mao, L. Entropy, Perplexity and Its Applications (2019). Lei Mao's Log Book

JOUR, Taylor, Ann Marcus, Mitchell, Santorini, Beatrice, 2003/01/31, 978-1-4020-1335-, The Penn Treebank: An overview 10.1007/978-94-010-0201-1_1

https://colah.github.io/posts/2015-08-Understanding-LSTMs/Built by Oinkina with Hakyll using Bootstrap, MathJax, Disqus, MathBox.js, Highlight.js, and Footnotes.js.

Tian Wang, Kyunghyun Cho Larger context language modelling with recurrent neural network, New York University (2016)