**Introduction**

The goal of statistical language modelling (SLM) is to define a probability distribution over a sequence of words (Zhai & Massung, 2016, p. 50) or phonemes/characters. Statistical language models (SLMs) are of central importance to any natural language application involving prediction, for example automatic speech recognition, machine translation, and text generation. Perhaps the best-known model is the *N*-gram, but the largest progress in SLM in recent years can be attributed to artificial neural networks (ANNs).

Specifically, recurrent neural networks (RNNs) are at the heart of this progress. Because their output is not only fed forward, as it is in traditional ANNs, but can be looped back onto itself, RNNs have a kind of ‘memory’ that traditional ANNs lack. This is crucial to SLM, because the next word in a sentence usually depends on the words that came before it. The superiority of RNNs over *N*-gram models was demonstrated by Mikolov, Karafiát, Burget, Černocký and Khudanpur (2010), whose best mixture of three RNN models achieved a reduction of 50% in perplexity as compared to a state-of-the-art 5-gram back-off model.

However, conventional RNNs are difficult to train, and their memory capacity only suffices for short-term dependencies (Bengio, Simard & Frasconi, 1994). For example, consider this sentence: “We visited Berlin, the capital of Germany”. Here, to predict “Germany”, it is essential to still remember “Berlin”. Should the distance between “Berlin” and “Germany” increase, however, then it becomes more difficult for conventional RNNs to make the right prediction. A crucial innovation in this respect was the invention of long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997): a specific type of RNN in which each neuron consists of four interacting layers. Within a neuron, three gates regulate which information is let through, and which is forgotten. In 2012, Sundermeyer, Schlüter and Ney showed that LSTM networks yielded a perplexity about 8% lower than conventional RNNs.

Despite being state-of-the-art, text generated by LSTM RNNs can currently still fairly easily be distinguished from human-generated text, usually because these texts are not very coherent. For example, Karpathy (2015) trained a 3-layer RNN with 512 nodes per layer on a corpus of all of Shakespeare’s work, with characters as input. This is a fragment of the text that was generated:

*PANDARUS:*

*Alas, I think he shall be come approached and the day*

*When little srain would be attain'd into being never fed,*

*And who is but a chain and subjects of his death,*

*I should not sleep.*

Thus, the quest to find the optimal architecture is still ongoing. People have explored both variations in architecture and in hyperparameter settings (examples: Sak et al., 2014; Greff et al., 2015).

there are still many open questions regarding what the optimal architecture of such RNNs would be. What counts as ‘optimal’ of course is dependent on factors such as the available computational resources, …, and … .

In this report, we will investigate how four RNN hyperparameters affect the quality of the resulting language model. This ‘quality’ is operationalised in two ways: the lowest categorical cross-entropy that the model achieved during training (objective), and the quality of a lyric generated by this model (subjective).

The four parameters under investigation are:

1. Vocabulary size
2. Number of layers (1, 2, 3)
3. Number of nodes (256, 512)
4. Drop-out rate (0.2, 0.5)

In the context of RNNs, the vocabulary size concerns the number of input and output nodes. Seemingly trivial, a decision to both include upper- and lower-case letters implies doubling the number of nodes, making correct classification more difficult.

The number of layers…

The number of nodes…

The drop-out rate…

Points for discussion

We didn’t explore learning rate, although Greff et al. (2015) found that to be the hyperparameter that explains most variance (67% in speech recognition and 89% in handwriting recognition). What were our learning rate settings?

Literature overview

Chelba et al. (2014) compared various types of SLMs on a billion word dataset, and demonstrated that RNNs achieved the lowest perplexity, meaning that they were best at predicting upcoming words.[[1]](#footnote-1)

Hermundstad et al. (2011): compared neural network architectures

Sak et al. (2014): compared architectures of conventional LSTMs and LSTMs with a recurrent projection layer. The latter achieved (slightly) better results while using less parameters. Further aspects of architecture that were compared were the number of layers (1, 2, 3, 5, 7), and the number of parameters, both in terms of word error rate, and in terms of training time. Also an introduction to how LSTMs work.

Sutskever et al. (2014): a relatively unoptimized neural network architecture which has much room for improvement outperforms a mature phrase-based SMT system 🡪 because training takes so long, unoptimized ANNs are still often published

Greff et al. (2015): compared the vanilla LSTM architecture to eight possible modifications. Vanilla performs reasonably well and none of the modifications significantly improves this. In terms of hyperparameters, by far the biggest effect was found for learning rate, followed by the hidden layer size.

Mikolov et al. (2010) showed that RNNs are better than *N*-grams.

Sundermeyer et al. (2012) showed that LSTM RNNs are better than conventional RNNs by an 8% drop in perplexity. They also compared different model architectures: units with a sigmoid activation function vs. LSTM units; hidden layer size (50-350); 1 vs. 2 hidden layers; sequence length (1-64 sentences); number of clusters (1-1000)

References

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157–166.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 1997), 1735-1780. DOI=http://dx.doi.org/10.1162/neco.1997.9.8.1

<http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-3/>

1. While Chelba et al. (2014) do not provide any details on their RNN architecture, we assume they used LSTMs, as their success in long-term dependencies has made them the default choice in SLM. [↑](#footnote-ref-1)