Recurrent Neural Networks & Google's Neural Machine Translation System

Jacob Chuslo

Since the inception of "Big Data" and cloud computing within the last 10 years, major multinational companies and corporations have begun to use machine learning algorithms, especially those related to *deep learning*, to enrich user experiences or solve business problems. One of the most popular services powered by machine learning technology is language translation services. This paper will focus on the underlying machine learning technology for Google's language translation service, Google Translate, and the deep learning framework underlying the service's state-of-the-art technology: the Google Neural Machine Translation system. The system is based on a variant of Recurrent Neural Networks (RNNs), which are a form of feed-forward neural network that specialize in analyzing sequential data. The algorithm has been used in numerous applications including artificial music composition, textual analysis and time-series analysis (Malhotra et al., 2015). To date, one of the most effective application of RNNs has been machine translation.

In a standard feed-forward neural network, the network is designed to have 1 input layer, a set number of hidden layers, and an output layer. The hidden layers and output layer have a specific number of "neurons" where a linear combination of all inputs from the previous layer and weights linking each neuron of the previous layer to each neuron of the next deepest layer are linearly combined and passed through a normalization function at each neuron in the deeper layer. After the output for all functions within the neurons of the output layer (deepest possible layer) are computed, these values are used to calculated the error of the prediction based on the actual output. The loss function is then used in the back-propagation process whereby weights are updated based on their current value and this error value. This process of sending weights forward, calculating loss, and updating (calculating new) weights continue until weights converge based on some predetermined criteria (Allen, 2020).

Figure 1 shows the structure of an RNN. RNNs are constructed based on the same principles as a standard feed-forward network, but with a modification to account for sequential data. Standard feed-forward networks send data from the input layer to the output layer in a single motion without interruption, while an RNN re-computes its hidden layer *and* computes a unique output for each sequential input (Goodfellow et al., 2016). On the left side of the figure, the prototypical RNN's structure is presented in a frame similar to a standard feed-forward neural network, with the main difference being that hidden layers are used to

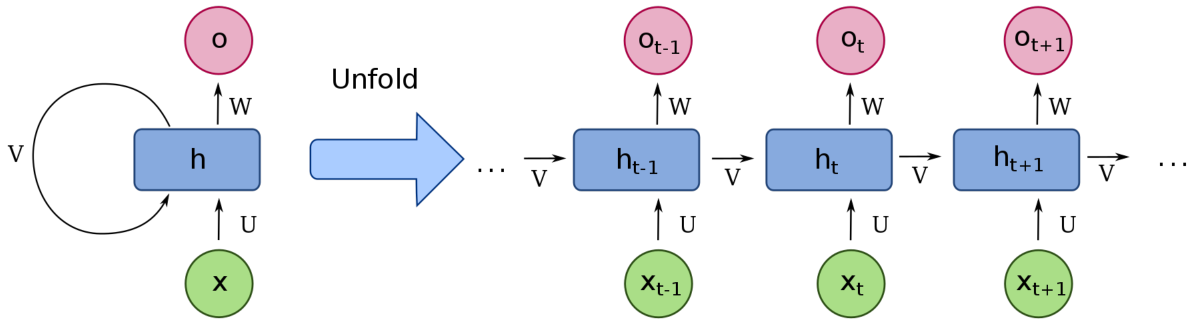

Figure 1: Structure of Recurrent Neural Network
Image by fdeloche, (Wikipedia), Creative Commons Attribution-Share Alike 4.0 International licensing

recompute themselves (at different sequential indexes). On the right side of the figure, we see this structure "unfolded" into its constituent sequential indexes, and it is here we see explicitly how each hidden layer is computed as a function of the hidden layer from the previous sequential index and the input of the current index. So, for hidden layer h at time t, $h_t = f(h_{t-1}, x_t)$. Each output is a function of the hidden layer and the inputs in the "local network" for its respective sequential index t, implying $o_t = f(h_t, x_t)$.

In the application of RNNs to real world problems, data which span over a relatively long set of sequential indexes (i.e. in the sequence $t = 1 \dots T$, T is very large) cause problems in the back-propagation procedure of the algorithm. Neural networks update weights between a "shallower" (closer to the initial inputs) and a "deeper" (closer to the final outputs) layer during back-propagation by using the output error (from the output layer), the output from the neuron in the shallower layer, and a delta value computed using the derivative of the output of the neuron in the shallower layer and the output error from the output layer. One important complication of the back-propagation procedure is the problem of vanishing gradients. Consider a hypothetical network with hidden layers $h_1 \dots h_n$ where h_n is the final output layer. For $h_1 \dots h_{n-1}$, the updates for weights connecting h_{i-1} and h_i are a function of weights connecting h_i and h_{i+1}. Because weights are typically < 1, the delta values for weight update values shrink substantially, and thus the weights themselves change less and less for layers further and further from the output layer h_n (closer and closer to the input layer h_1). Because RNNs essentially have a hidden layer for each step in the sequence $t = 1 \dots T$, there are T hidden layers, and so weights connected shallow layers do not significantly change during the RNN's back-propagation process (which is specifically called "Back-Propagation Through Time").

To solve this problem of vanishing gradients, "gated" RNNs were constructed to allow networks to retain

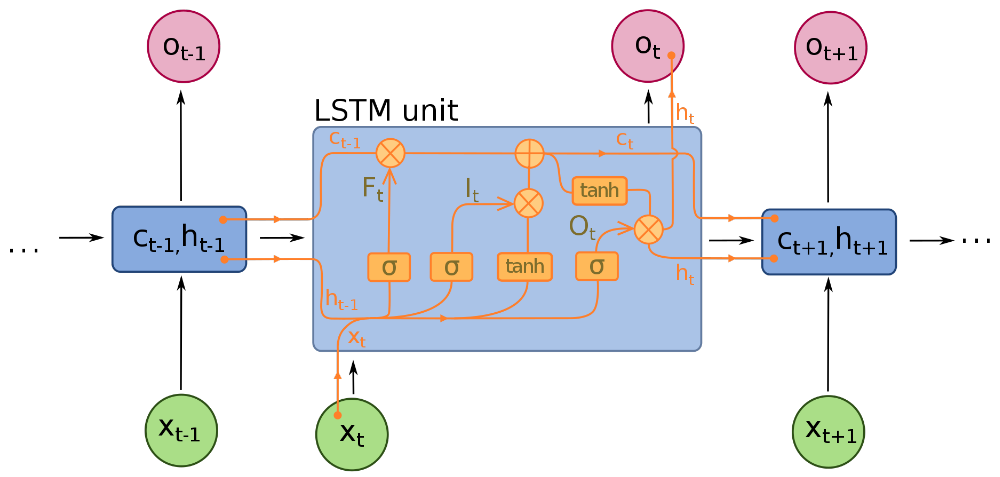

Figure 2: Structure of Long Short Term Memory (LSTM) gates
Image by fdeloche, (Wikipedia), Creative Commons Attribution-Share Alike 4.0 International licensing

information from earlier steps in the sequence. One of the newest effective variants of gated RNNs is the Long Short Term Memory (LSTM) model which is the basis of the RNN used in the Google Neural Machine Translation System. LSTM models use gates to determine which information from past hidden layers is unimportant, which information from new inputs may potentially be important, and mechanisms for implementing information that is finally deemed important (Hochreiter and Schmidhuber, 1997).

Figure 2 displays how the gates of an LSTM model operate.

The network is comprised of an *input* gate, a *forget* gate, and an *output* gate. Information is inputted into each of the 3 gates, and at each time step a new "state" C is created to influence an output o which will in turn influence a new hidden layer h, which will in turn influence a new "state" C_{t+1} and so forth. Initially at time t, the system receives a new input x_t and the previous hidden layer h_{t-1} (which contains information on the previous state C_{t-1}). The *forget* gate F_t decides which information from x_t and h_{t-1} (and therefore C_{t-1}) is irrelevant and generates a new set of weights to reflect this. The *input* gate I_t decides which information from x_t and h_{t-1} (and therefore C_{t-1}) is relevant and reflects that in its new respective weight vector. The new state C_t is then computed as a function of (1) the previous state weighted by the forget gate and (2) the sigmoidal output of the linear combination of x_t and h_{t-1} weighted by the input gate: $C_t = F_t C_{t-1} + I_t \phi(x_t, h_{t-1})$ where $\phi()$ represents the sigmoid function. Finally, the system output generated by the output gate is multiplied by the regularized function of the state $tanh(C_t)$ in order to compute the new hidden layer h_t which will influence the next state C_{t+1} (Goodfellow et al., 2016).

In summary, information from new inputs and previous hidden layers (and therefore previous states) arrive at the LSTM system, a forget gate determines which information from the previous state is irrelevant, an input gate determines which information from the new input is important, a new state is computed based on information from the forget gate, the previous state, the input gate and the new input, and finally a new hidden layer is computed based on the important information from the system output (generated from the output gate) and information from the newly computed state regularized by the *tanh* function. Armed with this knowledge of RNNs and the LSTM framework, the structure of the GNMT can now be explored.

Originally, machine translation was performed deterministically by simply applying rules of known syntax for any 2 languages, translating each word of a sentence from one language to another individually or in phrases. The first improvement on this model was the introduction of statistical machine translation (SMT) which solves the problem of finding the sentence in the native language **n** which has the greatest probability of being translated to the target sentence **f** in the target language by Bayes' rule: $p(\mathbf{n}|\mathbf{f}) = p(\mathbf{f}|\mathbf{n})p(\mathbf{n})$ (Koehn et al., 2007). The IBM "Alignment Models" created in the 1980s and 1990s is a modern example of an SMT-based language modeling system (Brown et al., 1993). These models used n-gram models as their basis, which is a set of models that predict a word t in a sentence or phrase based on the $t-(n-1)$th word, implying $p(w_t|w_{t-1}\dots w_1) = p(w_t|w_{t-(n-1)})$. The next improvement was the use of neural networks. n-gram models do not generalize well from training to testing data because every n-gram is assigned a unique probability distribution, so when the model observes an n-gram phrase not observed during the training phrase, the model makes a best guess based on text observed during training. Neural networks solve this problem by learning general features for each word/phrase which allow the n-gram model to consider features of words that appear before word $t-1$ (Sundermeyer et al., 2015). To generalize the neural network-based n-gram into a model which uses the full history of a sentence $w_{t-1}\dots w_1$ instead of $w_t \dots w_t - n - 1$, the next improvement was to use a recurrent neural network in order to retain information from arbitrarily long sentences/textual examples, and because RNNs trained on arbitrarily long sentences are subject to vanishing gradients during the Back-Propagation Through Time process, LSTM models were implemented (Mikolov et al., 2010; Sundermeyer et al., 2015).

The GNMT is built on an LSTM-based RNN language model called a Neural Machine Translation (NMT) model (Wu et al., 2016). The NMT is a model that uses all of language modeling features above implemented in a Sequence-to-Sequence LSTM-based RNN model (Sutskever et al., 2014). Figure 3 displays the structure of a Sequence-to-Sequence LSTM-based RNN language model (the encoder-decoder mechanism is beyond the scope of this paper). The first half of the Sequence-to-Sequence model (before the "encoding vector" output) uses an LSTM-based RNN with no outputs, and the second half (after the "encoding vector" output) uses

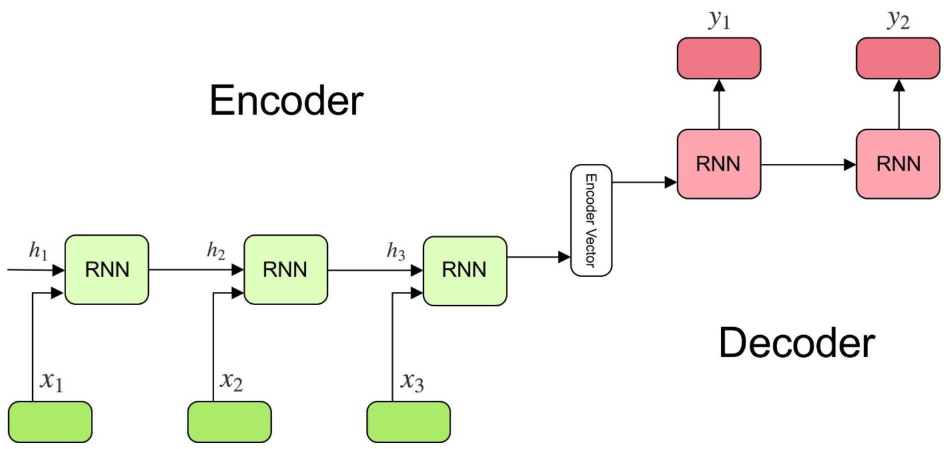

Figure 3: Structure of Long Short Term Memory (LSTM) gates
Source: Kostadinov (2019)

an LSTM-based RNN with outputs. The first segment inputs data from the target sentence in the native language, updating states and hidden layers as the model moves through the sentence. The second segment then produces the translation of the target sentence, 1 word at a time, continuously updating the hidden layers and states of the network and using the predicted word at index t as the input to predict the word at index $t + 1$ (not seen in Figure 3. One problem with this approach is that input sentences are recorded by the LSTM-only segment in sequence, and so translations to languages where word order differs from the native language (e.x in English $\leftrightarrow$ Chinese translation). As a result, an *attention mechanism* is added to the GNMT. This is a hidden layer within the Sequence-to-Sequence model that processes information from the hidden layers generated from both the LSTM-only and LSTM-based RNN sections of the model which detects more general features of the input sentence and word pairs within the input sentence (Bahdanau et al., 2015; Graves, 2013).

The GNMT's performance is measured on datasets linking English $\leftrightarrow$ French and English $\leftrightarrow$ German translation in terms of a BLEU score, a metric which computes the accuracy of a potential translation based on provided references (Papineni et al., 2002). At its core, the BLEU score is a geometric mean of performance metrics collected for each n-gram, where each n-gram's metric measures the maximum number of times a certain word from teh n-gram occurs in one of the reference texts divided by the total length of the n-gram. Wu et al. (2016) report the best BLEU score among GNMT model variants for English $\leftrightarrow$ French data is 37.90. The authors also report that a traditional phrase-based translation model scores a 37.00 BLEU score on the same data. For English $\leftrightarrow$ German data, the best BLEU score is 23.12 while a phrase-based model scores a 20.7. Wu et al. note that no RNN-based language model, GMNT or otherwise, has outperformed a human translator, although the GMNT's BLEU score is close to the BLEU score of a typical human translator.

Various problems still exist with NMT models, namely regarding (1) the treatment of rare words and (2) what the model will do when it encounters an input that was not encountered during training. Much of the NMT literature is considering various ways to deal with the former problem, but none have been put into production. Regarding the latter, Google has implemented a feature into the GNMT that at least makes an improvement on translating n-grams or sentences between language pairs that were not encountered during training (Johnson et al., 2017). Typically, the GNMT is trained based on a set number and ordering of language pairs, but has trouble translating a sentence between language pairs that the model did not encounter during training. To remedy this, the GNMT adds a token to the beginning of an input sentence specifying the proposed target language which partially allows the GMNT to perform this action. After training, Johnson et al. (2017) find that the modified GNMT has learned deeper characteristics of the translation of a specific sentence between input and target languages, regardless of the input/native language, and is able to at least partially translate sentences/phrases within a language pair that wasn't encountered during training. To illustrate, if the GNMT is trained based on a sentence that is known in languages A, B, C, and the model is trained on translations between $A \leftrightarrow B$ and $A \leftrightarrow C$, then the model can reasonably translate the sentence for the pairing $B \leftrightarrow C$.

References

Allen, M. (2020). Neural networks. Lecture Notes on Machine Learning.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *arXiv e-prints*, page arXiv:1308.0850v5.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Koehn, P., Och, F., and Marcu, D. (2007). Statistical phrase-based translation. *STAR*, 45(5).

Kostadinov, S. (2019). Understanding encoder-decoder sequence to sequence model. `https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346`.

Malhotra, P., Vig, L., Shroff, G., and Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning — ESANN*.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, USA. Association for Computational Linguistics.

Sundermeyer, M., Ney, H., and Schluter, R. (2015). LSTM neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):517–529.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv e-prints*, page arXiv:1609.08144v2.