MT using RNNs enriched with Universal Dependencies

Anonymous Author(s)
Affiliation
Address
email

Abstract

Sutskever et al. (2014) popularized a simple NN model for machine translation that encodes a sentence of the target language into a vector space with an LSTM, then decodes the vector into a sentence of the target language with another one. This paper explores the effect of replacing the LSTMs with GRUs, and whether augmenting the input word vectors with dependency labels from improves performance. It finds that GRUs optimized for speed substantially outperform LSTMs with the same number of parameters and identical optimization. The dependency information seems to help the GRUs, but less so with the LSTMs, possibly owing to the reduced hidden state size of the LSTMs.

1 Introduction

The machine translation system used by Sutskever et al. (2014) was able to achieve impressive performance on the task of English-French machine translation by encoding a reversed source sentence into a vector space using an LSTM network (Hochreiter and Schmidhuber, 1997) and then decoding it into the target language using another LSTM. However, it was not able to outperform the state-of-the-art phrase-based system (Durrani et al., 2013), and it required a total of 1.9 billion parameters (five LSTMs with 384M parameters each). This raises two questions: *can we build a similar model that works better?* and *can we build a smaller model that works as well?*.

The simple RNN [1] they use had no access to syntactic information that it couldn't induce from the sentence string. Thus one strategy for improving performance would be to enrich the model with syntactic knowledge. A basic step would be to pretrain the word vectors using the `word2vec` (Mikolov et al., 2013) or GloVe (Pennington et al., 2014) models, as these models generally encode some information about a word's part of speech. However, part of speech information is extremely shallow; for machine translation, it would be more useful to know the functional role that a word or phrase plays in the source sentence. This is exactly the kind of information encoded in the dependency labels in the Universal Dependencies (UD) project (Agić et al., 2015); so indicating to the model that a verb is the root of the sentence (`root`) as opposed to an adverbial clause (`advcl`) or some kind of aside (`parataxis`) might help it determine the correct translation of the sentence. Because the goal of UD is to develop a cross-linguistically applicable tagset, rather than tying the annotation scheme to a particular language, there are currently 18 languages with UD annotations that could be used to train a dependency parser, making it an appealing source of dependency information for an MT task.

[1] Throughout this paper, the abbreviation 'RNN' will be used to mean *recurrent NN* as opposed to *recursive NN*

The cononical RNN[2] calculates the value of its hidden state(s) from two sources–the input from a previous layer (possibly an input layer) and the activation from a previous timestep. Thus to compute the value of a 100-dimensional hidden state from a 50-dimensional input, an RNN requires $100^2 + 100 \times 50 = 15000$ parameters. The canonical LSTM, by contrast, uses three "gates" to condition the value of the layer's output–an *input gate*, a *forget gate*, and an *output gate*–where the value for each vector of gates is computed from three sources–the input, the previous output, and the value of the unit (i.e. the value before the output gate activates). Thus to compute the output of a 100-dimensional LSTM layer from a 50-dimensional input, an LSTM requires $3(100^2 + 100^2 + 100 \times 50) + (100^2 + 100 \times 50) = 90000$ parameters. Finally, gated recurrent unit (GRU) networks (Cho et al., 2014) are comparable to LSTMs, but use only two gates (an *update gate* and a *reset gate*) without a cell state used to determine the values of the gates; consequently, a GRU for the same task would require $2(100^2 + 100 \times 50) + (100^2 + 100 \times 50) = 45000$ parameters. While all three models use $O(nm + m^2)$, in some difficult tasks (such as MT) the bound on the number of parameters set by available memory could actually limit the model's performance (by preventing larger models from being built and by blocking more memory-intensive optimization algorithms). It is therefore important to make sure every parameter in the model is actually making a valuable predictive contribution.

2 The Models

Due to limitations on time and computational resources[3], only four small models were trained. All use word vectors pretrained with GloVe; two models use 200-dimensional vectors with no dependency augmentation, and two models use 150-dimensional word vectors and 50-dimensional, randomly initialized dependency vectors. At each timestep for the latter, the input word's vector and the input dependency label's vector are concatenated together and fed into the network, giving the model slightly more syntactic information about the word's role in the sentence but reducing the amount of semantic information it has.

Because Sutskever et al. noted that deeper networks noticeably improved performance, the models here all use two hidden layers. Two models use the GRU architecture and two use the LSTM one—however, all have the same number of parameters per layer. The LSTMs and GRUs were optimized for speed by modifying their structure to allow the computation to be done with only one dot product. For the LSTM this paper follows the approach to LSTMs on the Theano (Bergstra et al., 2010) webpage (as they were coded using Theano) in removing the dependence on the current/previous hidden state (which also decreased the number of parameters needed per output hidden node); for the GRU this only means changing the *reset gate* into an LSTM-style *forget gate*. The modifications to the structure are shown in Figure 1. In order to keep the number of parameters consistent across models, the GRUs had hidden sizes of 200 nodes whereas the LSTMs had hidden layers of 150, giving both of them the same number of parameters as a basic RNN with hidden size 600 (i.e. $600^2 + 200 \times 600 = 480000$ for the first layer and $600^2 + 600^2 = 720000$ for the second).

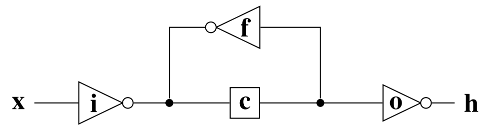
Following the insights of Le et al. (2015), all matrices (here including standard, non-recurrent weight matrices) are initialized with nonzero values along the diagonal only. Because the network was fairly small, it could be optimized using AdaDelta (Zeiler, 2012), which in practice converges much faster than stochastic gradient descent and is less sensitive to hyperparameters than Nesterov's accelerated gradient (Nesterov, 1983) Finally, the output layer was non-recurrent, and used a softmax classifier to predict the next word in the sequence (this layer is only relevant for the decoder RNN, of course).

3 Data

Since the scope of this project is very limited, only English to French translation has been attempted so far. But because UD has data in a lot of European languages (none of it parallel, unfortunately), it seemed very forward-thinking to use a corpus with a lot of parallel languages; consequently, the data come from Europarl (Koehn, 2005). However, the Europarl corpus is very "raw", and demanded

[2] Throughout this paper, the abbreviation 'RNN' will be used to mean *recurrent NN* as opposed to *recursive NN*

[3] I didn't want to hog the NLP cluster during finals week



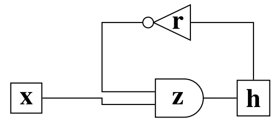
Original LSTM

$$\mathbf{i}_t = \mathrm{sig}\left(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + V_i\mathbf{c}_{t-1}\right)$$
$$\mathbf{f}_t = \mathrm{sig}\left(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + V_f\mathbf{c}_{t-1}\right)$$
$$\mathbf{o}_t = \mathrm{sig}\left(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + V_o\mathbf{c}_t\right)$$

Optimized LSTM

$$\mathbf{i}_t = \mathrm{sig}\left(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1}\right)$$
$$\mathbf{f}_t = \mathrm{sig}\left(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1}\right)$$
$$\mathbf{o}_t = \mathrm{sig}\left(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1}\right)$$

(a) Modifications to the LSTMs used in this paper



Original GRU

$$\tilde{\mathbf{h}}_t = \tanh\left(W\mathbf{x}_t + U\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right)\right)$$
$$\mathbf{h}_t = \left(1 - \mathbf{z}_t\right) \odot \mathbf{h}_{t-1} + \mathbf{z} \odot \tilde{\mathbf{h}}_t$$

Optimized GRU

$$\tilde{\mathbf{h}}_t = \tanh\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right)$$
$$\mathbf{h}_t = \left(1 - \mathbf{z}_t\right) \odot \mathbf{r}_t \odot \mathbf{h}_{t-1} + \mathbf{z}_t\tilde{\mathbf{h}}_t$$

(b) Modifications to the GRUs used in this paper

Figure 1: This paper modifies the gated architectures so that they depend only on the input and the previous hidden output, allowing a single dot product to calculate all the vectors needed for the final output

a considerable amount of pre-processing. First it had to be properly aligned, with sections not common to both languages removed; then it had to be tokenized, which was done with hand-crafted rules (to make sure the treatment of French contractions was handled consistently); then the data was POS tagged using nltk's HMM POS tagger (Bird et al., 2009), which was trained on the UD POS tags; then the POS-tagged data was parsed using MaltParser (Nivre et al., 2007; Ballesteros and Nivre, 2012), also trained on the UD data. The resulting corpus consists of 328,118 sequences, with 27,508,979 words in the English corpus and 31,267,702 in the French one. This means that the average English sequence has 84 words, and the average French sequence has 95—these are relatively long sequences for the model to learn. Additionally, the POS tagger and dependency parser were far from perfect, introducing considerable noise into the dependency labeling. To save memory, and hopefully reduce some noise from capitalization, the words of both languages were converted to lower case, resulting in vocabularies of size 32,058 types for English and 42,635 for French.

4 Results

While iterating through the entire dataset was impossible due to the time limitations, all models were able to see about 8600 training pairs (in 215 minibatches of 40) over the course of about 36 hours, at a rate of about 12 words/sec. This is extremely slow; part of the reason is undoubtedly because it was not GPU accelerated, part because the machine used to do the computations for the GRUs was accidentally overloaded, but part of it may have to do with an inefficient implementation.[4] The training cost is shown in Figure 2.

The most apparent trend between the models in the graph is that the optimized LSTMs described in Figure 1 perform much worse than the optimized GRUs, in spite of having the same number of parameters. This indicates that the LSTMs aren't using their parameters as efficiently as the GRUs; since the biggest difference between the GRUs and the LSTMs are the presence of an output gate in the LSTMs (as both have a gate for their input and their hidden layer), it seems likely that this may be the source of the inefficiency, and that substituting the output gate with a larger hidden layer may provide better gains. The second most apparent trend in the graph is that after about 100 minibatches, the GRU augmented with dependency labels starts consistently outperforming the

[4] I'm not sure, but I may have had the program update the entire library at each computation rather than just the words that were used in the computation…

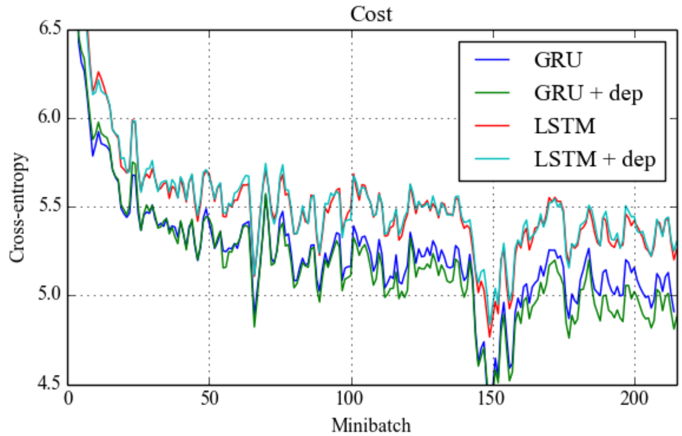


Figure 2: Training cost of the model according to minibatch (slightly smoothed for readability)

GRU without them, suggesting that even this small syntactic enrichment of the model may lead to appreciable gains, especially with a larger network and more training.

At this point, I would like to examine translations that the models make, in particular comparing the GRU without dependency augmentation to the GRU with dependency augmentation, to see whether the slight increase in training performance is matched by a slight improvement in the quality of the produced sentences. I would also like to show heatmaps of the weight matrices connecting the input layer to the first hidden layer, in order to see whether the part of the weight matrix corresponding to the dependency label is "hot", and if so, which labels are generally the hottest (indicating more valuable contribution to the resulting translations). Unfortunately, I can't yet—something seems to have gone wrong at the last minute. The first pickled save states from each network (generated after the first four hours of training) are exactly identical to the last ones (created after 32 hours), suggesting that either AdaDelta "broke" after the first few minibatches and started rendering all updates zero or the pickler only ever saved the initial state of the models, somehow ignoring subsequent updates—the latter seems most likely, because the weight matrices have all retained the initial diagonal shape, the biases are all set to zero, and performance appears to be continuing to improve beyond the first handful of iterations (the GRU with dependency labels especially so). Consequently, I can only show the improvement in the model cost–which *was* pickled correctly–at this time and speculate from that.☹

5 Future Research

There are a two primary ways this research can be extended (beyond fixing the pickle bug). The first would be to continue examining models made more complex with syntactic information, and the second would be to continue examing models made more simple with different kinds of RNNs. The simplest next step for adding syntactic information would be to make the model predict dependency labels as well as words (which it does not currently do); in this way, during training the model would receive feedback regarding the correct syntactic structure of the target sentence, rather than only learning how to predict the right strings. Another possibility would be to turn the input layer into a tensor layer; rather than concatenating the word vector and the dependency vector, this version would take their outer product and dot this times a weight *tensor* rather than a weight *matrix*. This would allow the model to capture different kinds of dependencies between the input word and its role in the sentence. An even *more* complex model would encode the sentence using the whole dependency tree, using some form of the DT-RNN model proposed by Socher et al. (2014), and the *most* complex model would attempt to *decode* the sentence into a dependency tree. This poses some significant difficulties that would need to be overcome, since dependency trees are in general not binary, and in some cases non-projective (i.e. there may be non-dependent words separating dependents from heads).

In the other direction, it would be worth examining further how the recurrent architecture affects performance. The first thing to do would be to compare the optimized models shown here to the originally proposed models, to see whether the smaller and faster versions (especially of the LSTM) are inherently inferior to the original versions for this task. It should also be shown that this effect scales with the size of the hidden layer—that is, it may be that the simpler architecture only makes a difference for small hidden sizes, and when more information can be stored at each layer the more complex structure may become more efficient. Next, it would be worth examining how the models compare to diagonally-initialized basic RNNs with the same number of parameters, exploring whether basic, well-initialized RNNs outperform these more complex models. Does the ability to modulate input and easily forget previous hidden states significantly improve performance? If smaller models can work as well as larger models, it may open the gates to better optimization algorithms, potentially resulting in better final solutions.

6 Conclusion

This paper has found some evidence suggesting that providing an encoder-decoder RNN model with richer syntactic information may improve performance without even needing to increase the size of the model. It also found some evidence suggesting that the simpler GRU architecture may be more efficient for capturing linguistic dependencies than the LSTM architecture, potentially allowing for better performance either from more representational power or faster optimization algorithms. While there are a few caveats (the dependency augmentation didn't improve the LSTMs and it is possible that the gain in efficiency from smaller architectures won't scale to larger hidden sizes), the results of this study point to these lines of research as directions to explore in our attempt to make deep learning the highest-performing approach for MT.

References

Agić, Ž., Aranzabe, M. J., Atutxa, A., Bosco, C., Choi, J., de Marneffe, M.-C., Dozat, T., Farkas, R., Foster, J., Ginter, F., Goenaga, I., Gojenola, K., Goldberg, Y., Hajič, J., Johannsen, A. T., Kanerva, J., Kuokkala, J., Laippala, V., Lenci, A., Lindén, K., Ljubešić, N., Lynn, T., Manning, C., Martínez, H. A., McDonald, R., Missilä, A., Montemagni, S., Nivre, J., Nurmi, H., Osenova, P., Petrov, S., Piitulainen, J., Plank, B., Prokopidis, P., Pyysalo, S., Seeker, W., Seraji, M., Silveira, N., Simi, M., Simov, K., Smith, A., Tsarfaty, R., Vincze, V., and Zeman, D. (2015). Universal dependencies 1.1. Github repository.

Ballesteros, M. and Nivre, J. (2012). Maltoptimizer: an optimization tool for maltparser. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 58–62. Association for Computational Linguistics.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python*. " O'Reilly Media, Inc.".

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Durrani, N., Haddow, B., Heafield, K., and Koehn, P. (2013). Edinburghs machine translation systems for European language pairs. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 114–121.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12.

Socher, R., Karpathy, A., Le, Q. V., Manning, C. D., and Ng, A. Y. (2014). Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.