Sentence Correction using Recurrent Neural Networks

Gene Lewis
Department of Computer Science
Stanford University
Stanford, CA 94305
glewis17@stanford.edu

Abstract

In this work, we propose that a pre-processing method for changing text data to conform closer to the distribution of standard English will help increase the performance of many state-of-the-art NLP models and algorithms when confronted with data taken "from the wild". Our system receives as input a text word, sentence or paragraph which we assume contains (possibly none) random corruptions; formally, we say that the input comes from a corrupted language domain that is a superset of our target language domain. Our system then processes this input and outputs a "translation" or "projection" to our target language domain, with the goal of the output being to preserve the latent properties of the input text (sentiment, named entities, etc.) but mutated in a way that embeds these properties in a representation familiar to other NLP systems.

1 Introduction/Related Work

In our literature search, we've found that there is a multiplicity of representations for languages, ranging from rule-based models that encode "hard" grammatical knowledge [1] to stochastic models that learn a suitable representation from data; these stochastic representations range from simple n-gram models [2] to highly complex probabilistic network representations [2, 3]. Hidden Markov Models have been shown to exhibit a strong ability to capture many of the high-level dynamics of natural English language [2]; however, such models make the rather strong assumption of conditional independence of the current word from all previous words given the immediately previous word, which prevents HMM's from modeling crucial long-range dependencies. Recurrent Neural Networks, on the other-hand, have been shown to be more adept at capturing these long-range dynamics [4]. In our work, therefore, we turn to the recent success of neural networks in the field of Natural Language Processing for model inspiration.

Our review of Luong et al. (2015) [5] demonstrated a very similar system performing the task of attention-based neural translation. Their aim was to use a neural network that models the conditional probability $p(y|x)$ of translating a source sentence, $x_1, x_2, ..., x_n$, to a target sentence, $y_1, y_2, ..., y_n$. Their system consists of two components: (a) an encoder that computes a representation s for a source sentence and (b) a decoder that generates one target word at a time and decomposes the conditional probability as

$$log(p(y|x) = \sum_{j=1}^{m} logp(y_j; y_{<j}, s)$$

This is the system we chose to utilize for our task. We found that almost all recent related work such as (Kalchbrenner and Blunsom, 2013[6]; Sutskever et al., 2014[7]; Cho et al., 2014[8] Bahdanau et al., 2015 [9]; Luong et al., 2015[5]; Jean et al., 2015[10]) have the same NMT system. The

difference between the approaches are in which RNN architectures were used for the decoder and how the encoder computed the representation of the source sentence s. Kalchbrenner and Blunsom (2013) used an RNN with the standard hidden unit for the decoder and a convolutional neural network for encodingthe source sentence representation. On the other hand, both Sutskever et al. (2014) and Luong et al. (2015) stacked multiple layers of an RNN with a Long Short-Term Memory (LSTM) hidden unit for both the encoder and the decoder. Cho et al. (2014), Bahdanau et al. (2015), and Jean et al. (2015) all adopted a different version of the RNN with an LSTM-inspired hidden unit, the gated recurrent unit (GRU), for both components.

While our approach is very similar to that of Luong et. al 2015, our aim was to apply those principles to a different task. Their task was machine translation, while our task was correcting social media at the character level. Our training differed from Luong et al. 2015. Their stacking LSTM models have 4 layers, each with 1000 cells, and 1000-dimensional embeddings. Futhermore, their parameters were uniformly initialized in [-0.1, 0.1], they used 10 epochs, they started with a learning rate of 1 where after every 5 epochs they halved the learning rate, their mini-batch size was 128, and their normalized gradient was rescaled whenever the norm reached 5. In contrast, due to the significant time taken to train, we didn't tune our parameters. For example, we only had at most 2 layers.

We contrast our character-level approach with the commonly used word-level approach. We examine the approach of Bahdanau et al. 2015, which is a word-level approach at machine translation. Their paper aimed to address the issue of representing the source sentence as a fixed-length vector. Their strategy was a proposed model that (soft-)searches for a set of positions in a source sentence where the most relevant information to the generated word is concentrated. The model predicts a target word based on the context vectors associated with these source positions and all the previously generated target words. Instead of attempting to encode the source sentence into a fixed-length vector, they encoded the input sentence into a sequence of vectors and chooses a subset of that sequence while decoding the translation, allowing them to better handle long sentences.

2 Approach

2.1 Sentence Representation

Before we can construct a full model for English language, we must first choose a sufficient input representation. Because our ultimate goal is correction of text, we anticipate the need to handle an exponentially large input space of text tokens that don't correspond to standard vocabulary dictionaries by choosing to model character-level dynamics [4] over the standard word-level representation.

We model characters as a one-hot 94-dimensional vector that selects which of the 94 printable ASCII characters we wish to represent. We forgo using a pre-trained character-embedding matrix; though research has been fairly vigorous in this field, no readily useful implementations could be found. Thus, we choose to learn our own character-embedding as part of the model training.

2.2 Language Dynamics Representation

Given that our raw representation will be characters, we can now turn to the question of modeling English language itself. We choose to utilize Recurrent Neural Networks [11] as our basic model, as recent literature has shown it's effectiveness in capturing many of the dynamics of fluent English [4]. RNN's are similar to Hidden Markov Models in that they ingest and process the input in a step-wise manner, utilizing the way the previous input interacted with the model to inform how the current input should interact with model. However, unlike Hidden Markov Models, RNN's learn to model input dynamics by embedding them into a shared hidden representation space, allowing them to implicitly capture sophisticated interactions that the explicit probabilistic representation of an HMM can't [4].

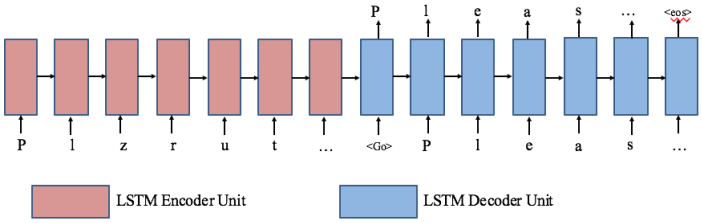

Figure 1: Example of RNN normalization model. The unnormalized text is fed in the bottom of the encoder, and the decoder generates the translation.

2.2.1 Number of Layers

Motivated by results in the literature suggesting that a larger number of layers in a neural model are capable of learning progressively "higher-level" and more abstract concepts from data [12], we postulate the hypothesis that a deeper neural model will be able to learn natural language concepts beyond character dynamics, such as proper word-level tokenization, recognition of named entities, and simple grammatical rules. To this end, we trained a one layer network, and a two layer deep network.

2.2.2 Hidden Units

We choose to utilize Long-Short Term Memory (LSTM) Units [13] for the hidden units in our neural network, as LSTM units have special "gating" machinery that allows them to propagate error signals over many time steps; we reasoned that this choice is logical given our decision to model characters instead of words, leading to very long decompositions of sentences as character chains and finer-grained, more-difficult-to-capture character level language dynamics.

2.2.3 Probabilistic Emission Model

For our probabilistic emission model, we choose to utilize the standard maximum element of the result of the Softmax operation [12] on an output 94-dimensional vector, given by:

$$\sigma_t(f_t(x))_j = \frac{e^{f_t(x)_j}}{\sum_{i=1}^{n} e^{f_t(x)_i}}$$

where $n = 94$ and $f_t(x) \in \mathbb{R}^n$ is the pre-softmax output of our neural translation model at timestep t.

This choice has dual motivation: Softmax is an easily differentiable operation, which comes in handy for our training algorithm (see section 2.3); Softmax is also a simple, easily-understood transformation that allows as much of the probabilistic machinery as possible to be learned and modeled by the underlying Recurrent Neural Network, preventing difficult-to-calculate inference and difficult-to-interpret equations from entering into emission calculations.

2.3 Training Algorithm

For our training algorithm, we utilize standard mini-batch stochastic gradient descent optimization via the backpropagation algorithm [12] with sequence cross-entropy as our loss function, given by:

$$L(x, y) = \frac{1}{t} \sum_{i=1}^{t} y_t \log(\sigma_t(f_t(x_t)))$$

where, for each timestep, we sum over the cross-entropy loss in predicting the character for the current time-step. This loss attempts to syncronize two probability distributions together; since the output of the Softmax of our network represents a probability distribution over characters, and the

target probability distribution is a one-hot vector with a one in the slot corresponding to the target character and zeros everywhere else, this loss is interpretable as encouraging our model to not only predict the correct character but predict the correct character with as strong a confidence as possible (captured by trying to push all the probability mass to the slot with the desired character).

3 Experiments

3.1 Data

A detailed review of related literature led to an English corpus of 2000 texts from the National University of Singapore [14]. From our review, it seems that this is the only publicly available normalized corpus for texts.

When preprocessing our data, we first examined the histogram of lengths of source sentences and target sentences; we noticed that though the longest source sentence and target sentence were approximately of length 200 and 220 respectively, the number of sentences with length larger than 170 and 200 were very sparse. Because the number of steps to unroll the neural translation model is dictated by the largest number of steps we expect from an input and output sentence, we filtered these long sentences out to cut down on model size and model compile time. We also filtered out text that didn't contain strictly printable ascii characters; all in all, these two processing steps only eliminated 11 source-target message pairs out of 2000, leaving us with plenty of data to work with.

Because the data was received in a randomized format (subsequent text messages aren't correlated), we didn't bother to perform randomization pre-processing. Because we didn't have an large parallel corpus to work with, we were interested in preserving as much data for training as possible; this motivated our decision to split 99% of the data into a training set and 1% into a testing set. Our final data counts were 1970 for the training set and 19 for the testing set.

Example data from the testing set:
Input: "Ic...Haiz,nv ask me along?Hee,im so sian at hm."
Output: "I see. Sigh, why do you never ask me along? I'm so bored at home."

3.2 Baseline Models

For our baseline model, we implemented a character-level unigram model where each line in our source set is mapped character by character to its corresponding line in our target set; the result of this training is an unnormalized conditional probability distribution, represented by a dictionary mapping each character in the source set to a list of counts of characters from the target set. To generate translations from a given source file, we examine each source character and consider the related list of target characters, outputting the character with the highest count (this is equivalent to probabilistic maximization over the target characters).

A word-level unigram model was also implemented. Punctuations and capitalization were preserved in the training set. During text generation, we again output the most common word that is mapped to by each corrupted word in question.

We computed the perplexity of both models. The average perplexity in our validation set for the character-level model and word-level models are 5.366 and 2.009, respectively. We found that using a character-level model introduced much more variation in the translation process, where characters often mapped to spaces. Because the source and target words were often not aligned, the mapping for characters led to inaccurate results. Therefore, the word-level model for our baseline achieved much better, meaningful results in translation.

3.3 Neural Translation Models

Both neural-translation models were trained for 8000 steps with a batch size of 64, for a total of approximately 260 epochs over our training set. We used an initial learning rate of 0.5, a learning

rate decay factor of 0.99, and a maximum gradient norm of 5 to prevent "gradient blow-up" [15]. Validation testing and serialization was carried out every 100 steps; at the 8000 step, the 1-layer model achieves a validation perplexity of 1.060, while the 2-layer model achieves a validation perplexity of 1.102.

We used a hidden state size of 100 for both models. This choice serves dual purpose: for computational reasons, this size helped the models fit nicely in memory, which helped speed up the training process. It's also approximately a one-to-one correspondence with the total number of characters, which we hypothesize forces the model to learn a character-by-character dynamical system; though we acknowledge that it's likely that a larger hidden state will allow our model to achieve a better perplexity in a shorter amount of training time, we're interested in the ability of the models to learn natural language characteristics directly on a character level.

3.4 Error Analysis

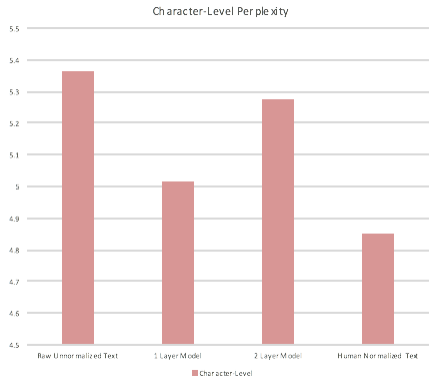

Figure 2: Character-Level Perplexity across models and data

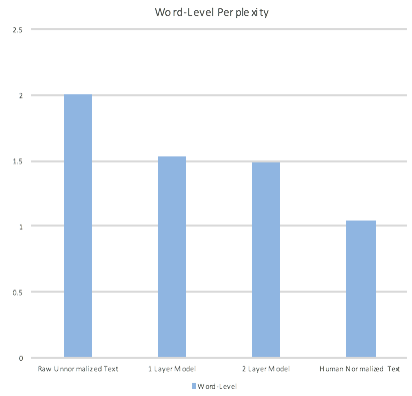

Figure 3: Word-Level Perplexity across models and data

3.4.1 Comparison to Baseline Models

We evaluated the output of our 1-Layer and 2-Layer models by using it as input to calculate perplexity in our baseline models. Our 1-Layer model achieved 5.014 perplexity on the baseline character-level model and 1.533 on the baseline word-level model. Our 2-Layer model achieved 5.278 perplexity on the baseline character-level model and 1.490 on the baseline word-level model.

From these results, we observe that both of our models generated text with a lower perplexity score than the original input text, indicating that the baseline models would more accurately predict the correct translation given our text. Both models achieved relatively close perplexity

values, with the 1-Layer achieving a lower perplexity value for the character-level model while the 2-Layer achieved a lower perplexity value for the word-level model.

3.4.2 Qualitative Results on Test Set

We found that both models are able to correct basic spelling errors ("tink" translated to "think"), perform short and long-range abbreviation expansion ("u" translated to "you"; "ppl" translated to "people"), basic grammatical correction (translated sentences always begin with a capitalized letter and end with some form of punctuation, even when the source sentence did not), and decent preservation of text when no error is present ("I'm outside now" translated to itself).

This final property is especially surprising, given that the training input contained no source-target pairs that were the same to encourage that this behavior should be learned.

Though both models managed to learn the above properties, we note that there are indeed differences between the produced translations of the one-layer and two-layer models.

The two-layer model learned more powerful punctuation rules, such as:

Input: "Lea u there?"
1-Layer: "Lea are you there?"
2-Layer: "Lea, are you there?"

However, the one-layer model has a much easier time expanding complex abbreviations:

Input: "Ic...Haiz,nv ask me along?"
1-Layer: "I see ask me along?"
2-Layer: "Ic5. Sing, neen ask I am"

We believe that this mismatch in predictive ability stems from the sizes of the networks. Because of the larger size of the 2-layer network, more iterations are required before converging on a local optimum that can properly identify properties such as sentence length and abbreviation expansion; because the 1-layer network has significantly fewer parameters, it's probable that the network has converged much closer to a local minimum than the 2-layer network given the same number of training iterations. However, we note that the ability of the 2-layer network to learn proper punctuation rules indicates that it is capable of learning deeper concepts than the 1-layer network. This reasoning has precedence in the literature [12], where deeper neural networks used for image classification require much larger datasets and much longer training times to converge than their shallower counterparts; it's even suggested that shallower networks severely outperform deeper models when data is scarce [16].

4 Conclusions

One of the challenges in our research was the scarcity of normalized datasets widely available for our training. While there are plenty of social media data such as tweets, they are not translated to proper English. We were limited to 2000 normalized text messages from 2003 to 2015. Because of this smaller dataset, our results were less generalizable, performing much better with malformed text written in a similar style to the corrupted input that we trained on. Thus, for future steps, we plan to either utilize Mechanical Turk to utilize human intelligence to translate social media, which according to our literature review didn't work so well for others, or to manually translate more input to use for training. We hypothesize that more data would allow our neural network to translate significantly more accurately.

Due to limited resources, we were unable to train models with hidden state size larger than 100 or number of layers larger than 2; it has been shown that such improvements can yield very sizeable increases in neural network performance. Thus, we hypothesize that if access to large GPU

computing resources is obtained, much more powerful and accurate models could be trained using an expanded corpus generated as explained above.

Finally, we're very interested in exploring applications of our text-correction model to other languages and evaluate its performance; if the text-correction scheme can be extended to other languages, this model could be both an important pre- and post-processing step in the machine translation pipeline.

References

[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[2] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[3] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[4] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015.

[5] Christopher D. Manning Minh-Thang Luong, Hieu Pham. Effective approaches to attention-based neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[6] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. *EMNLP*, 2013.

[7] Oriol Vinyals Ilya Sutskever and Quoc V. Le. Sequence to sequence learning with neural networks. *NIPS*, 2014.

[8] Caglar Gulcehre Fethi Bougares Holger Schwenk Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014.

[9] Kyunghyun Cho Dzmitry Bahdanau and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

[10] Roland Memisevic Sebastien Jean, Kyunghyun Cho and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *ACL*, 2015.

[11] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[14] NUS Natural Language Processing Group, 2015.

[15] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.