CS230

DeepDowNN: Automatic Crossword Clue Generation

Angad Rekhi
Dept. Electrical Engineering
Stanford University
arekhi@stanford.edu

Ting Chia Chang
Dept. Electrical Engineering
Stanford University
tcchang3@stanford.edu

Charlotte Kirk
Dept. Chemical Engineering
Stanford University
ckirk@stanford.edu

Abstract

Crosswords are a very popular type of puzzle, first invented in 1913 and included in almost every newspaper around the world ever since. In this work, we seek to automate the process of novel clue creation for crossword puzzles using neural networks. In particular, we use RNNs as generators of multi-word text (clues). We combine crossword data from the NYT crossword database with definition data from the GCIDE open-source dictionary to explore both word-to-clue (one-to-sequence) and word/definition-to-clue (sequence-to-sequence) models. Our networks are able to combine and regurgitate clues for words they have been trained on, but for words outside of the training set, the clues they generate are generally unrelated to the passed word. We outline possible reasons for this poor performance and provide corresponding paths forward.

1. Introduction

Crosswords are a very popular type of puzzle. Most crosswords published today, such as those published daily in the New York Times, are composed by humans. Creating these puzzles takes considerable time and skill. Programs now exist that can create the word grids used for crosswords and that can choose clues to match the words in the grid from a database of previously used clues. However, generating novel clues for crosswords using an algorithm has not yet been actively explored. We propose the use of recurrent neural networks (RNNs) to automatically generate novel clues for crossword words. We consider two approaches to tackle the problem: using the given word to generate a clue directly (one-to-sequence) and generating a clue based on the definition of the given word (sequence-to-sequence).

2. Related Work

Clue generation is similar to translation and text summarization, in which the goal is to output a sequence relating to an input sequence. Neural machine translation as proposed by Sutskever *et al.* attempts to use a single large neural network to output correct translations [1]. Bahdanau *et al.* adds an attention mechanism on top of a basic encoder-decoder network to improve the performance of the network [2]. The attention model has also been applied to abstractive summarization applications by Rush *et al.* [3]. The slight difference between our problem and the above two applications is that we are not looking to generate the "best" clue for a given word; rather, we are interested in generating novel clues (that are still sensible) for words both inside and outside the training data. This functionality, if achieved, would represent a significant advance beyond today's crossword generation process.

3. Dataset and Features

We obtained word/clue pairs from the New York Times crossword from January 1994 to March 2018; the unfiltered dataset consists of 742,149 pairs. This dataset includes repeated words with clues that are not necessarily identical. We also used the GNU Collaborative International Dictionary of English (GCIDE) open-source dictionary which is derived from the 1913 Webster's Revised Unabridged Dictionary as a source of definitions; we extracted 98,855 word/definition pairs, using only the first non-obsolete definition for every word. It is important to note that in our data, clues do not always correspond to the dictionary definition of the word; we return to this point in the discussion.

We first processed the data by removing punctuation and converting all words to lowercase. Then, we formed word/definition/clue triples by joining the word/definition and word/clue datasets based on the words. To represent words in our data, we used Global Vectors for Word Representation (GloVe) pre-trained on Wikipedia [4] as word embeddings. The GloVe dataset that we used has a vocabulary size of 400K words ("Unfiltered", Table I); however, in order to decrease the number of trainable parameters and achieve reasonable computation time (as explained in the models section), we limited our vocabulary size to a small fraction of the total.

Table I: Filtered Dataset Sizes

Filtering method	Word/Clue Pairs	Word/Definition/Clue Triples
Unfiltered	576,344	251,524
First 5	156,007	57,033
20K Common Words	161,539	33,294
20K + First 5	29,616	5,943
Extraction	576,344	211,038
Extraction + First 5	156,007	**47,801**

Initially, we limited our vocabulary to the words in the 400K dataset that are also in the 20K most commonly used word in the English language (as determined by n-gram frequency analysis of Google's Trillion Word Corpus); this reduced our vocabulary size to just under 20K, as some common words did not have embeddings. But this significantly reduced the number of word/clue/definition triples ("20K Common Words", Table I). Therefore, we decided to extract all word vectors in GloVe as long as they are present in the data; this keeps ~211K triples while still achieving a reduction in the vocabulary size from 400K to ~80K ("Extraction", Table I).

We also experimented with keeping only the first 5 clues for a given word and definition so that the network would not be confused by too many different clues for the same input, and so that for a fixed amount of computational resources, the network would see a wider variety of words/definitions ("First 5" in Table I). We settled on combining this method with the extraction described above to arrive at a dataset size of ~48K triples, with a vocab size of ~80K.

Finally, we added 3 tokens to the extracted GloVe embedding dataset: a start token, an end token, and a pad token. Our start and end tokens are randomly generated vectors (after setting a random seed, so that we get the same vectors every time the code is run) of length 200 (the dimensionality of the GloVe embeddings that we chose to use), and our pad token is a vector of zeros of the same length. For each clue, we appended a start token before the first word of the sequence and an end token after the last word of the sequence. Pad tokens were used such that the length of each clue would equal the length of the longest clue. We did not add start, end, or pad tokens to our input when the input is a single word (one-to-sequence); when the word's definition is used as input (sequence-to-sequence), we added the word itself to the beginning of the definition and appended

pad tokens at the end to ensure a consistent length across the dataset. Example data with tokens are shown in Table II.

Table II: Example Data with Tokens

Word	Definition	Clues
overripe	overripe matured to excess <p>…	<s> somewhat spoiled <e> <p> … <s> like a mushy banana say <e> <p> … <s> turning brown as a banana <e> <p> …
angel	angel a messenger <p> …	<s> gabriel eg <e> <p> … <s> harp player <e> <p> … <s> theater backer <e> <p> …
spate	spate a river flood an overflow or inundation <p> …	<s> sudden flood <e> <p> … <s> outpouring <e> <p> … <s> inundation <e> <p> …
reagent	reagent a substance capable of producing with another a reaction especially when employed to detect the presence of other bodies	<s> litmus for one <e> <p> … <s> chromatography spray <e> <p> … <s> chemistry lab selection <e> <p> … <s> assaying aid <e> <p> …

4. Methods and Models

We built three different encoder-decoder models (Fig. 1) using Keras [5] with Tensorflow [6] backend. The number shown in each block is the dimension of the output of the layer represented by that block. The single-time step LSTM encoder of the Word-to-Clue model (W-C) takes as input an index into the vocabulary list; its output states are fed into the 21-time step LSTM decoder. The output of the decoder feeds into a fully-connected (dense) layer before a softmax activation over a one-hot vector of length equal to the vocabulary size. Pad tokens are masked (though refer to the discussion section for more details). Categorical cross-entropy loss is used for training.

For the Definition-to-Clue model (D-C), we use a 20-time step bidirectional LSTM encoder. We add an attention mechanism to this model to arrive at a new sequence-to-sequence model (D-CA): the attention block generates a weighted sum of the encoder output at each time step, which is concatenated with the output state of the decoder before passing through a fully-connected layer and onto a softmax activation across the vocabulary. The trainable parameters for all models are about 5.7 million, determined mainly by the final softmax layer (~79663 × 64).

5. Results, Discussion, and Future Work

To quickly get a simple working model from which to proceed, we first trained the three models shown on a small amount of data (2560 W/D/C triples with an 80/20 split). All models achieve low bias when trained long enough on this fraction of the dataset, but exhibit poor variance (Fig. 2, L). Our models already included some degree of regularization (mainly dropout layers), so we looked to scale up the amount of data. We first tuned the learning rate at small scale to ensure efficient use of resources when scaling up (Fig. 2, R). We also confirmed that validation accuracy would increase with the amount of training data (Fig. 3, L).

We then ran two larger-scale experiments with our D-CA network, using 20480 triples (100 epochs) and 40960 triples (30 epochs), both with an 80/20 split. The network trained on more data achieves a smaller variance, but at the expense of bias (Fig. 3, R). We used this network to generate clues by sampling from the softmax-generated distribution over the vocabulary at each time step.

Based on the quantitative results shown in Figs. 2-3 and our qualitative judgment of generated clues (Table III), we find that our networks are able to combine and regurgitate clues for words they have been trained on, but for words outside of the training set, the clues they generate are generally unrelated to the passed word, though they are almost always grammatically correct and "clue-like" in their feel.

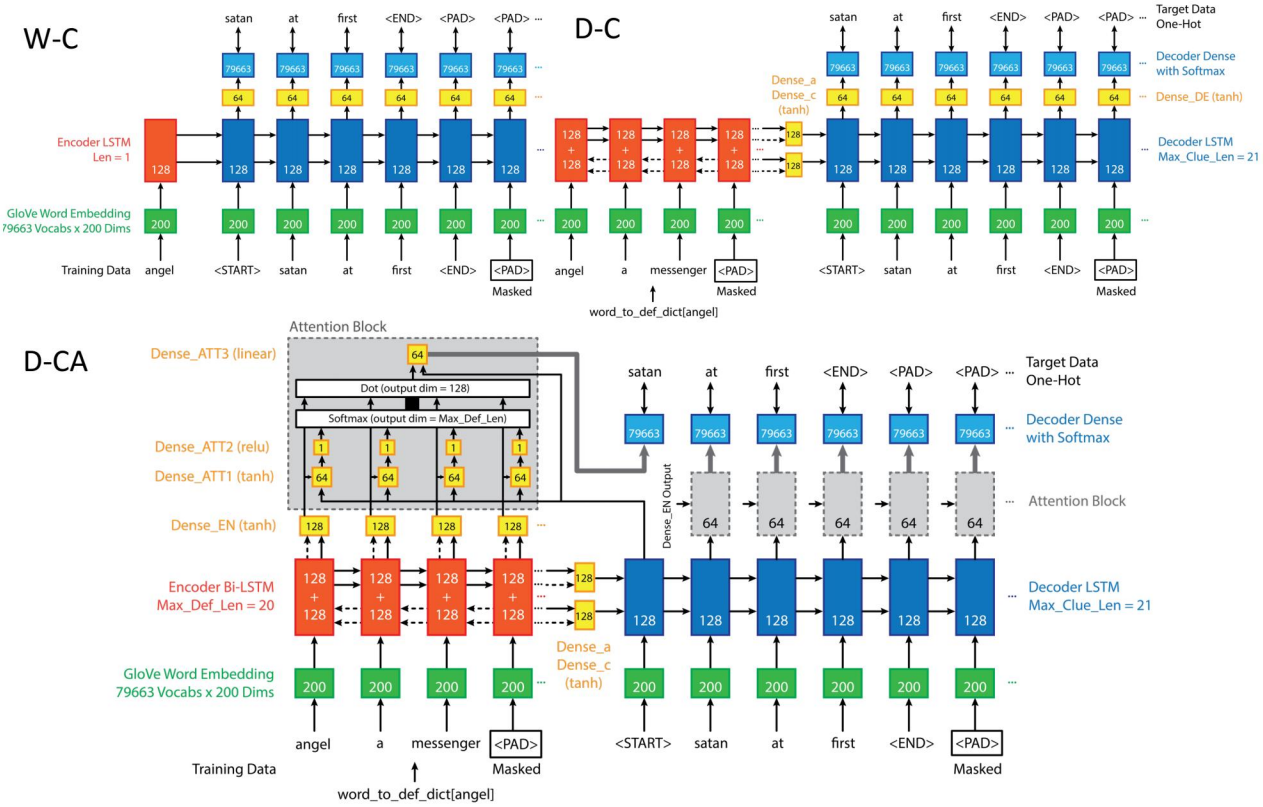

Fig. 1. Word-to-Clue model (W-C), Definition-to-Clue Model (D-C), and Definition-to-Clue with Attention model (D-CA). Please zoom in for details as space is limited.

We've identified several possible reasons for this poor performance. First, in our dataset, definitions and clues often do not refer to the same meaning of the word; if the word's GloVe embedding does not have enough (or the right) information to overcome this challenge, then there may not be enough information in the input to generate a meaningful output clue. This problem is aggravated by the fact that our source of word definitions is a century-old dictionary, while the crossword clues are from the last 20 years of puzzles, making it all the more likely that clues and definitions often don't match. Moreover, multiple different clues for the same word/definition pair might be confusing the networks, which is why we limited the frequency of any given word in our dataset to 5 (which may include repeat clues).

In addition to definition-clue mismatch, there are issues surrounding accuracy and loss that must be resolved in any future work. It seems that although we implemented masking layers in Keras, the calculation of training and validation accuracy still includes <PAD> tokens. This reduces the dynamic range of the accuracy metric. The use of a BLEU score-like accuracy metric might actually be better-suited to this problem than the simple categorical accuracy that we used here.

Finally, because our performance ended up being data-limited and resource-limited, a simple way to boost performance would be to include unknown word tokens in order to increase data size while decreasing vocabulary size (and therefore the number of trainable parameters), while perhaps implementing pointing [7] to be able to copy words not in the vocabulary.

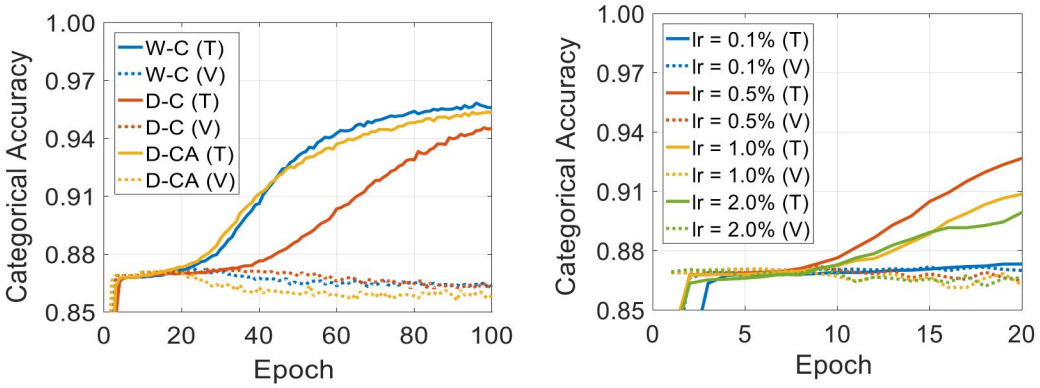

Fig. 2. L: Accuracies vs epoch for all 3 models. R: Learning rate optimization using D-CA model.

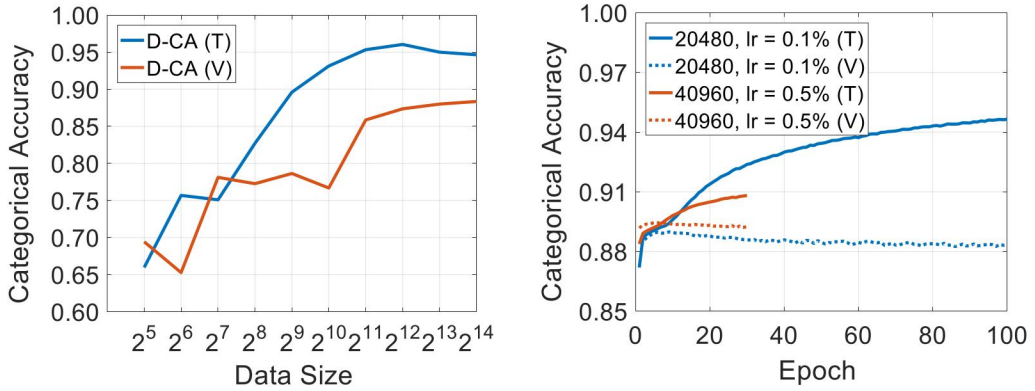

Fig. 3. L: Accuracies vs dataset size for D-CA model. R: D-CA accuracies vs epoch for large datasets.

Table III: Generated Clues from D-CA

Word	Generated Clues	Top $\hat{y}_1$ Values and Corresponding Words
overripe*	turning like a mushy banana, like spoiled, not bright	45% like, 21% turning
angel*	magical harp, jockey cordero, non non jew, gabriel garcia marquez novel genre	19% jockey, 7.9% harp
spate	giving indian, napoleons method, head on an egg, serious offering	2.8% big , 2.7% like
reagent	dirty physics, kind of food, base acting, kitchen gizmo, jock	8.3% one, 4.7% kind

*In training set

6. Contributions

C. Kirk found the NYT crossword dataset and scraped it for word-clue pairs. A. Rekhi found, downloaded, and scraped the GCIDE open-source dictionary for word-definition pairs. T. C. Chang guided the construction of the first few models. All members worked together to format the data, take the models from a rudimentary state to complex sequence-to-sequence models running on AWS instances, and write the final report.

References

[1] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," *Adv. in Neural Information Processing Systems*, 2014.

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *Intl. Conf. on Learning Representations*, 2015.

[3] A. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *Empirical Methods in Natural Language Processing*, 2015.

[4] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. EMNLP*, 2014.

[5] F. Chollet *et al.*, "Keras," 2015. Available: keras.io.

[6] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," 2015. Available: tensorflow.org.

[7] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proc. 55th Annual Meeting ACL*, pp. 1073–1083, July 2017.