A Study of Neural Architectures for General Knowledge Crossword Clue Solving

Rajat Agrawal
ucabra4
18153777

Jack Copsey
ucabops
19149017

Alok C. Suresh
ucabacs
15013377

Daniel A. Williams
ucabdaw
19031356

Abstract

In this paper we explore various NLP-inspired methods for configuring automated crossword solvers. Completing the entries within a crossword is an example of the *reverse dictionary problem*: how can we measure the similarity between a source phrase and a target phrase? Sub-tasks under this problem topic include the retrieval of words based on their definitions, and finding suitable replacements for short phrases in the form of synonyms. We consider techniques including word embeddings and recurrent neural networks, and define several models applicable to the problem. To quantitatively compare performances, we then apply our models to solving crossword puzzles from *The Guardian*. Constraints imposed by the crosswords such as known word lengths provide additional performance gains. The main finding of our paper is that simple pooling methods (e.g., bag-of-words) are more effective than RNN architectures for crossword solving, due primarily to the typically short lengths of crossword clues. We conclude by suggesting several further directions our work could be taken in.

1 Introduction

1.1 Context

As an application of open-domain question answering, the *reverse dictionary problem* seeks to find the closest phrase in semantic meaning (*head*) to a given definition (*gloss*). Reverse dictionaries are frequently used by professional writers to identify suitable words to express a concept. Previous research has concentrated on using information retrieval techniques, such as cosine similarity with *tf-idf* weightings (Hill et al., 2016). Recent advances in neural network architectures for generating word embeddings have had a substantial influence on current approaches to the reverse dictionary problem (Parry, 2018). By generating an embedding-based representation of a gloss's words, one can estimate the head's embedding by searching for the gloss representation's nearest neighbors in the embedding space. Current research has focused on the choice of latent space for embeddings and the methods for computing them, including neural bags of words (*NBOW*) and recurrent neural networks (*RNN*).

Within the reverse dictionary problem, we consider the non-cryptic general knowledge crossword puzzle. Given a set of glosses (clues), one must identify the most suitable head (a word or phrase) that satisfies a length constraint, does not place letters on the board in conflict with intersecting heads and, where applicable, permutes into a provided anagram. The challenges of the general knowledge crossword puzzle stem from variety of clue types. Many clues are synonyms, antonyms or definition phrases, however semantic ambiguity may mean that more than one candidate head satisfies one or more of the aforementioned constraints. Other clues may require domain-specific knowledge that is weakly captured by pre-trained language models, such as geography, literature and music. To generate appropriate candidate heads for such clues, it may be necessary to fine-tune such language models using authoritative encyclopedic sources (e.g. Wikipedia).

In the remainder of this introduction we review the existing literature and formulate a problem statement for our investigation. In Section 2 we describe two neural-based architectures for an improved crossword solver. We present initial performance results in Section 3, followed by a discussion of these results in Section 4. We conclude with final observations in Section 5.

1.2 Relevant Work

Crossword puzzle solving is an NP-complete problem (Purdin and Harris, 1993). This encourages

probabilistic approaches to identifying candidate heads and filling in missing letters. In Purdin and Harris (1993) a genetic algorithm is developed for the *go-words* variant of the crossword puzzle; the algorithm formulation is tied closely to the puzzle format and a counterpart for general knowledge crossword puzzles is not immediately obvious.

Other attempts at crossword puzzle solvers have been informed by information retrieval techniques. In Hage (2016) simple candidate heads are identified with modified binary search, with fuzzy search for more difficult clues and pattern matching for character gap filling, nevertheless the solver struggles with referential clues (e.g. *See 9 across*). Jobin et al. (2017) explore the use of hidden semantic information extracted from clue sentences and entries from the WordNet database[1], although the model struggles with word inflections (e.g. past tense), pop culture references and obscure phrases, non-standard vocabulary (including slang) and heads containing multiple words.

A seminal approach described in Shazeer et al. (1999) and expanded in Littman et al. (2002) formulates a probabilistic constraint satisfaction problem for identifying suitable heads. The latter paper introduces the PROVERB crossword solver, which integrates an ensemble of expert modules to search through different domains of information. Each module's guess and confidence level is merged to produce a list of candidate heads which is fed to the solver. To fill in gaps and filter candidates, an "implicit distribution" module evaluates the character-level probability of a given head, allowing PROVERB to iteratively refine a puzzle's solution.

A number of subsequent papers' architectures have been influenced by PROVERB's design. In Radev et al. (2016), CRUCIFORM improves on the performance of PROVERB by exploiting a much larger database of online crossword clues and articles from the English-language Wikipedia. WEBCROW integrates web search functionality with a crossword clue database to identify lists of candidate heads (Ernandes et al., 2005). The web search module retrieves relevant documents, ranks candidate heads by their *tf-idf* scores and filters out morphologically-inappropriate candidates (e.g. excluding adverbs when the head must be a noun). Motivated by PROVERB, WEBCROW also implements an implicit distribution module, expert guess merger and maximum-likelihood grid-filling.

Following Ernandes et al. (2005), in Barlacchi et al. (2014b) the BM25 ranking function replaces *tf-idf* scores for ranking database clue similarities with the target clue, while a logistic regression module converts these scores into more intuitive confidence values. In addition, support vector classifiers have been trained for clue re-ranking (Barlacchi et al., 2014a; Nicosia et al., 2015; Moschitti et al., 2015).

Recent crossword puzzle solvers have explored neural network architectures driven by end-to-end learning. Pre-trained word embeddings and feed-forward deep neural networks have been used to calculate clue similarity scores for WEBCROW (Severyn et al., 2015; Nicosia and Moschitti, 2016). Similarly the models developed by Hill et al. (2016) and Parry (2018) concentrate on individual clues, using pre-trained word embeddings to find the best single-word head for a given gloss.

As a baseline in Hill et al. (2016), Word2Vec embeddings are used with a variant of cosine distance to compute similarity scores between the clue sentence and candidate heads. The authors then concentrate on recurrent neural networks (*RNN*), with the clue sentence as an input and the final stage's output yielding the clue's embedding representation. To improve robustness in handling rare and obscure clues, the training data includes Word2Vec embeddings for approximately $90,000$ single-word heads, sentences from the first paragraphs of the respective Simple English Wikipedia articles (as quasi-glosses), and approximately $850,000$ additional $\{\text{head, gloss}\}$ pairs from four online dictionaries. Crossword clue-answer pairs have been collated from various online sources (primarily *The Guardian* quick crosswords) to evaluate the performance of the baseline and RNN-based models. The models compare favorably with commercial, database-driven systems for long clue sentences and achieve comparable accuracy with shorter clue sentences.

Building on Hill et al. (2016), Parry (2018) explores different RNN structures. To capture contextual dependencies between words, the author replaces the unidirectional long short-term memory network (*LSTM*) with a bidirectional LSTM. The average of each stage's output is considered instead of just the final stage's output, which the author argues increases the influence of earlier gloss words on the output embedding. Finally the byte-pair en-

[1] https://wordnet.princeton.edu/

coding algorithm is used to generate a sub-word language model of the glosses to increase robustness with unseen words. These changes boost performance with respect to Hill et al. (2016).

2 Design

2.1 Data Collection and Storage

In gathering a database of crossword puzzles we have concentrated on the *quick* category of crosswords published by the British newspaper *The Guardian*. Empirically these puzzles are less cryptic than those of *The New York Times* and *La Reppublica*, examined in Hill et al. (2016) and Ernandes et al. (2005) respectively. We have processed $5,000$ puzzles dating from 2002 to 2018, extracting for each puzzle the set of clue identifiers (number and direction), clue words, corresponding answers, lengths of individual tokens in the answers, and positions in the puzzle grid.[2] An example crossword from the dataset is depicted in Figure 1. For the purpose of model evaluation we have used a subset of 500 crossword puzzles, corresponding to $11,851$ individual clue-answer pairs, while for supervised model training we use the remaining $4,500$ puzzles.

Each clue contains at least one gloss, of three possible types: synonymous (containing a definition e.g. '*Dog (5)*'), an anagram (containing a permutation of the answer's letters, e.g. '*Endurance - it's them (anag)*'), or referential (e.g. '*See 8 across*'). For synonymous glosses we pre-process the literal string by removing dates, punctuation, the possessive *'s* and quotation marks, then convert the clue string to lowercase letters and tokenise using white spaces. With anagram glosses we remove non-alphabetic characters before converting the character sequence to lowercase. For each clue we then return a list of the synonym tokens and the anagram characters where applicable. 96.19% of clues contain one gloss while 3.81% of clues contain two or more glosses, e.g. 'generous; type' for *kind*. We retain a list of each individual gloss for downstream filtering tasks, described in greater detail in Section 2.3.

2.2 Baseline Model: Neural Bag of Words

As a baseline model we implement an unsupervised NBOW model, using the Word2Vec embeddings pre-trained on the Google News corpus (Mikolov et al., 2013) as described in Hill et al. (2016).

For each clue phrase, we find the embedding vector representation of each token and use mean pooling to yield an estimate of the clue's representation. We then query the Word2Vec model for the most similar vectors to the clue representation, yielding a list of candidate answers. Formally, the pre-trained Word2Vec model defines a mapping σ from a vocabulary W to a 300-dimensional vector space $\mathbb{R}^{300}$ known as the *embedding space*. This mapping can be extended to arbitrary-length sequences of tokens $\mathbf{s} = (s_1, \ldots, s_n) \in W^n$ by mapping over each token separately:

$$\sigma_n : (s_1, \ldots, s_n) \mapsto \begin{bmatrix} \sigma(s_1) \\ \vdots \\ \sigma(s_n) \end{bmatrix} \in \mathbb{R}^{n \times 300}$$

Mean pooling transforms this matrix into a single vector $\bar{\sigma}_n(\mathbf{s}) = \frac{1}{n} \sum_i \sigma(s_i) \in \mathbb{R}^{300}$ by summing along columns, which can then be compared to each of the vectors in the set

$$\sigma(W) = \{\sigma(w) \,|\, w \in W\} \subset \mathbb{R}^{300}$$

using cosine similarity. This defines a ranking for the tokens $w \in W$ according to how similar they are to $\bar{\sigma}_n(\mathbf{s})$; the predicted solution $\hat{t}$ is set to the token with the best rank:

$$\hat{t} = \underset{w \in W}{\operatorname{argmin}} \frac{\bar{\sigma}_n(\mathbf{s}) \cdot \sigma(w)}{\|\bar{\sigma}_n(\mathbf{s})\| \, \|\sigma(w)\|}$$

For all of our models, we solely consider the similarities between vectors in an embedding space for the purposes of suggesting an initial list of candidate answers, prior to crossword constraints being imposed. In particular we have opted for our models not to consult multiple knowledge bases when searching for candidate answers, as models of this type have already been studied extensively both when applied to crossword puzzles in particular and more generally in the context of QA frameworks. See Ernandes et al. (2005), Radev et al. (2016) and Kalyanpur et al. (2012) for in-depth treatments of this topic.

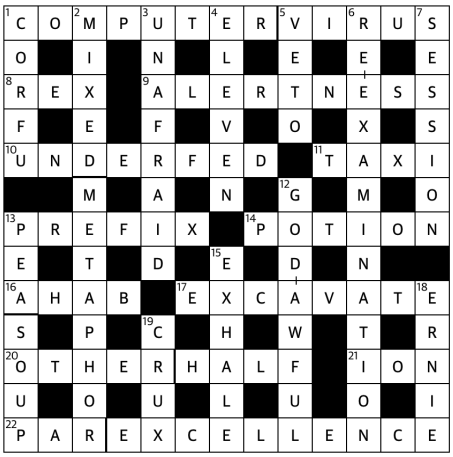
In generating the mean-pooled sentence representation we neglect gloss tokens not found in the vocabulary of the pre-trained Word2Vec model, affecting approximately 32.53% of clues. These tokens fall into three categories: Word2Vec stop words, proper names (i.e. of people or locations) and British English spelling variants.

[2] See `https://github.com/ucabops/robbie` for the reference code used to parse the raw crossword data

Crosswords

Wed 25 Apr 2012 01.00 BST

Quick crossword No 13,092



Across

1 Software program capable of causing harm to digital files (8,5)
8 King — Harrison? (3)
9 Vigilance (9)
10 Too thin? (8)
11 Vehicle for hire (4)
13 'Pre-', for example (6)
14 Concoction — elixir (6)
16 Captain of the whaler *Pequod* (4)
17 Dig out (8)
20 Spouse (5,4)
21 Charged particle (3)
22 Beyond comparison (3,10)

Down

1 Ionian island (5)
2 "A rolling stone is worth two in the bush", for example (5,8)
3 Dauntless (8)
4 Prime number (6)
5 Power to reject legislation (4)
6 Further check (2-11)
7 Sitting (7)
12 Appalling (3-5)
13 Green-coloured starter (3,4)
15 Breathe out (6)
18 It picks Premium Bond winners (5)
19 Pivotal point — cross (4)

Figure 1: Online edition of quick crossword no. 13,092 from *The Guardian*, with clues and corresponding solutions laid out in the grid. Individual tokens are separated either by thick black bars (used to denote spaces) or by hyphens.

Model	Constraint Filter					
	1	2	3	4	5	6
BOW-0						
BOW-1	x					
BOW-2	x	x				
BOW-3	x	x	x			
BOW-4	x	x	x	x		
BOW-5	x	x	x	x	x	
BOW-6	x	x	x		x	
BOW-7	x	x	x	x	x	x
BOW-8	x	x	x		x	x

Table 1: Model Functionality

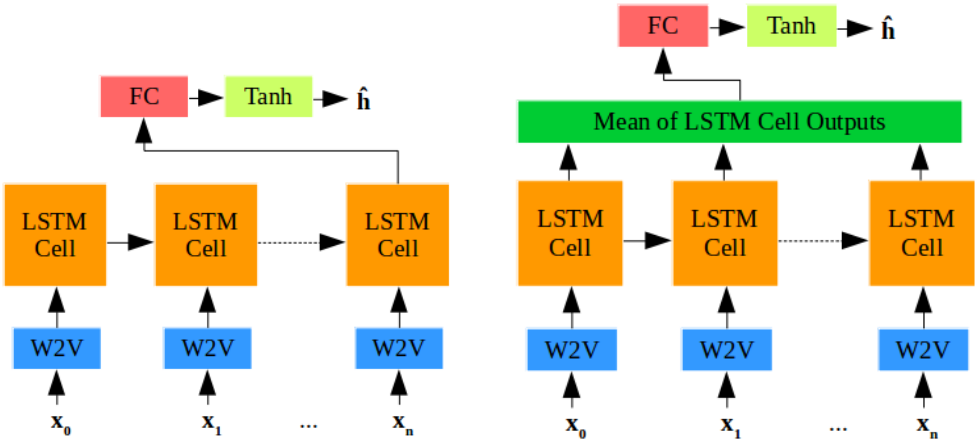
2.3 Enhanced Neural Bag of Words

To enhance the performance of the baseline model, we impose a series of filters that eliminate candidate answers that do not satisfy the puzzle's constraints.

First we remove candidate answers that contain one or more words provided in the clue sentence, since these never appear in the solution. We also remove candidate answers with incorrect lengths; for single-word answers this is equivalent to the answer's string length (e.g. 4 for *BALL*), while with answers composed of multiple words we consider both the total length of the concatenated words as well as the individual word lengths (e.g. 10 and 4,2,4 respectively for *CAFE-AU-LAIT*). If the clue is an anagram we ignore words that are not permutations of the clue's letters. This chain of filters yields a list of compliant candidate answers, which we rank by the cosine distance from the clue representation as before.

For certain entries in the crosswords, the clue may contain two or more gloss sets. In such situations the model performs an individual lookup on each gloss set, retrieving the respective rankings of tokens ordered by cosine similarity with the head token. The model then computes the mean cosine similarity for each candidate solution across the lists, re-ranking the candidate solutions in descending order of mean score. We hypothesise this will strengthen the model's performance on solutions corresponding to homonyms (words with multiple meanings depending on context).

After the final list of candidate answers is computed, we may derive the quality metrics and error statistics as described for the baseline model. In Section 3, we denote the NBOW-based models according to the number of constraint filters imposed on the list of candidate answers. The constraints identified in Table 1 respectively ensure that 1) candidate heads have the correct length, 2) gloss words are excluded, 3) non-anagrams are excluded, 4) candidates incompatible with each gloss are excluded (where multiple clue sets are given), 5) British English spellings are assigned the same embedding vectors as the equivalent American English spelling and 6) multiple-word answers are included.



(a) Unidirectional LSTM with Fully-Connected Layer

(b) Unidirectional LSTM with Fully-Connected Layer and Mean Pooling

Figure 2: LSTM Architectures

Figure 3: Prediction Generation

2.4 LSTM Models

Inspired by the family of models introduced in Hill et al. (2016), we first consider a unidirectional LSTM that accepts gloss token embeddings as input and yields an output embedding (Figure 2(a)), then identifies the nearest neighbour in the embedding space as the predicted head token (Figure 3). The model relies on the pre-trained Word2Vec embeddings identified in Section 2.2 to encode both the head and gloss tokens, as well as to decode the predicted embedding vector. We use the cosine loss $\cos(h, \hat{h}) = 1 - \frac{h \cdot \hat{h}}{\|h\|\|\hat{h}\|}$ as the objective function, as we seek to minimise the angular difference between the predicted output's embedding $\hat{h}$ and the true head's embedding h. The general approach is similar to that considered in the previous section, but with an RNN architecture used to accumulate tokens into a single vector instead of the mean pooling operation $\bar{\sigma}_n$.

For comparison we investigate the unidirectional LSTM presented in Parry (2018), depicted in Figure 2(b) (*LSTM-U*). The key differences with the previous LSTM architecture are a) mean pooling over the entire set of output vectors instead of using only the final stage's output vector, b) the gloss embeddings are learned in training instead of using pre-trained embeddings, and c) 100-dimensional word embeddings were used instead of 300-dimensional word embeddings. Parry postulates that this introduces an indirect bias towards earlier tokens in the input sequence, and can thus improve the quality of the predicted embedding.

Finally we also consider the bidirectional LSTM architecture (*LSTM-B*) from Parry (2018), which concatenates the final output vectors of forward and backward LSTM cell chains to yield the predicted embedding vector. We hypothesise that this could improve model performance by detecting sequential relations between gloss tokens.

3 Results

We present experimental results in Table 2 and report the following metrics:

- **Correct at 1** (C@1): the number of clues for which the correct answer is identified as the top candidate answer;
- **Median rank** (MR) of the correct answer in list of candidate answers retrieved;
- **Accuracy at 10 and 100** (A@10, A@100): the percentage of clues for which the correct answer is in the top 10 and 100 candidate answers respectively.
- **Duration** of test evaluation for the BOW models, in minutes.

As expected, the baseline model *BOW-0* is the fastest model to evaluate. However, it achieves the

Model	C@1	MR	A@10 / %	A@100 / %	Duration / min
BOW-0	0	43	10.34	23.77	83
BOW-1	1110	14	29.21	47.22	84
BOW-2	1449	13	29.72	47.25	111
BOW-3	1490	13	30.01	47.46	122
BOW-4	1494	**11**	29.47	45.81	155
BOW-5	1501	**11**	29.75	46.23	173
BOW-6	1502	13	30.39	48.02	155
BOW-7	**1636**	**11**	32.58	50.86	175
BOW-8	1618	13	**32.93**	**52.54**	145
LSTM-U	1361	274.5	25.00	42.20	—
LSTM-B	1304	440.5	23.80	39.40	—

Table 2: Results of Baseline, Enhanced NBOW and LSTM Models

lowest performance on all metrics, with no correct candidates at rank 1, median rank of 43, accuracy at 10 of 10.34% and accuracy at 100 of 23.77%. As more constraints are introduced, model performance on each metric generally improves but model evaluation becomes slower. When all constraint filters are included, *BOW-7* achieves the best performance on the number of correct candidates at rank 1 and the median rank of the correct answer, albeit with the longest duration to evaluate. In contrast *BOW-8* (which does not divide clues with multiple definitions into individual gloss sets) achieves the best results for accuracy at 10 and at 100 (32.93% and 52.54% respectively) in a significantly shorter time than *BOW-7*.

The initial LSTM model that we developed yielded poor performance results with an accuracy at 100 of 0.3%. The unidirectional LSTM model with output mean pooling from Parry (2018) performs better than the bidirectional LSTM model on all metrics. However, both are outperformed by nearly all of the BOW models (*BOW-2* through *BOW-8*).

4 Discussion

4.1 Comparison of BOW and LSTM Models

When evaluated on a test set of 500 crossword puzzles, the best-performing models are *BOW-7* and *BOW-8*, outperforming both the unidirectional *LSTM-U* and bidirectional *LSTM-B* models from Parry (2018). While *LSTM-U* and *LSTM-B* may achieve a higher number of correct predictions at rank 1 than *BOW-1*, they achieve lower performance on all other metrics, performing particularly worse on the median rank. Low median rank suggests that the LSTM models predict the rank of the correct head either very well or very poorly (hence the worse accuracy at ranks 10 and 100). This phenomenon is likely to have been exacerbated by the models' training conditions: *LSTM-U* and *LSTM-B* were both trained for ten epochs due to computational resource limitations, however it is likely that performance on the validation set would improve further if the LSTM models were trained for more epochs.

Nonetheless, given these results, the LSTM architectures may be unnecessarily complex when compared with the NBOW models. The relatively short lengths of token sequences involved in general knowledge crossword clues do not require sophisticated infrastructure for learning and tracking long-term dependencies between gloss words, so a simple NBOW model may suffice.[3] Despite using a different evaluation methodology the results of Hill et al. (2016) appear to confirm that NBOW models perform better than RNN-based models on the crossword clue resolution task.

4.2 BOW Model Choices

Among the NBOW models, the best performing models both used the first three filters (length, gloss words, anagrams) and the last two filters (British English spellings and multiple-word answers). The first three serve to eliminate incorrect candidates, while the last two compensate for the idiosyncrasies of the pre-trained Word2Vec model with respect to certain true heads. Notably using the fourth filter (filtering by multiple gloss sets) does not improve all performance metrics. For *BOW-7*

[3]For this reason we explicitly ruled out exploring attention-based models for this task.

the number of correctly predicted heads at rank 1 and the median rank is slightly higher than for *BOW-8*, however for an analogous pair of models (*BOW-5* and *BOW-6*) only median rank improves when this filter is applied. This may be due to the ineffectiveness of the filter in the presence of clues comprised of more than two synonyms. In these cases there are increasing numbers of lists over which a common intersection of words must be found, thus making it more likely that the correct answer candidate could be eliminated during this process. In addition, model evaluation is significantly longer for models using this filter, hence it is of negligible practical benefit to apply this filter.

4.3 LSTM Model Choices

Our initial attempt at implementing a unidirectional LSTM performed poorly. We speculate that this was because pre-trained embeddings were used for the gloss token representations, as opposed to learning the gloss embeddings. Both Hill et al. (2016) and Parry (2018) train input embeddings on a varied collection of information sources, including cryptic clues, dictionary definitions and Wikipedia article abstracts. This likely enhances a model's lexical range and explains their models' improved performance on general knowledge clues.

5 Conclusions

5.1 Summary of Results

In this paper we have investigated two classes of models for the automated solving of crossword puzzles: NBOW and RNN. We found that even the simplest NBOW implementations matched the performance of RNNs in most metrics, whereas NBOW models designed to take into consideration extra constraints significantly outperformed RNNs in our experiments. This combined with the shorter evaluation times and absence of training required to develop NBOW models makes it the superior choice for solving crosswords.

The short lengths of typical crossword clues suggests that order-dependence is not important when making predictions, which goes at least some way to explain why the LSTM-based models do not overwhelmingly outperform the NBOW models.

We were limited in our experiments to only training the LSTMs for 10 epochs each, due to the time required to make batch predictions. The accuracy achieved improved on each epoch and showed no obvious signs of slowing. Given greater computational resources it would be interesting to repeat our experiments with the LSTMs trained for more epochs, and check whether the model parameters rapidly converge to a limit or if performance surpasses that of bag-of-words within a reasonable number of epochs.

5.2 Extensions

In this report we have focused on predicting the solutions to individual entries in a crossword puzzle. Additional constraints can be enforced by considering the intersections between entries. For example in Figure 1, entry 14 across (*'POTION'*) intersects three other entries:

- entry 12 down (*'GOD-AWFUL'*)
- entry 6 down (*'RE-EXAMINATION'*)
- entry 7 down (*'SESSION'*)

Often these intersections are crucial in forcing a unique solution to a crossword puzzle, particularly in cases where a single clue admits multiple synonyms of the same length. More formally, these intersections introduce strong conditional dependencies between entries, where definite knowledge of one solution rules out a class of possible solutions for the entries it intersects. A simple way to model these dependencies is through using a *directed acyclic graph*: letting the entries in a crossword puzzle be denoted by $x_1, \ldots, x_n$, where we admit an arbitrary ordering of the entries amongst x_i, we can construct a suitable graphical model as follows:

- For each index i from 1 to n in turn:
- Find the maximal set of entries $S_i \subseteq \{x_1, \ldots, x_{i-1}\}$ such that every $y \in S_i$ intersects x_i at one or more tile(s)
- Form arrows between the entries $y \in S_i$ and x_i, directed from y to x_i

If we choose an ordering whereby the entries $\{x_i\}$ in the across direction are ordered earlier in the sequence than the entries $\{x_j\}$ in the down direction, then this procedure typically simplifies to introducing dependencies $x_j \,|\, \{x_{i'}\}$ for the subset of entries $x_{i'} \in \{x_i\}$ intersecting x_j.[4] Following this

[4]Note that in some quick crossword puzzles from *The Guardian*, one set of tiles can be shared between multiple entries in the same direction. For example, in quick crossword no. 10,873 the word *'PAPER'* is shared between the two separate answers *'GREEN PAPER'* and *'WHITE PAPER'*.

principle for the example crossword in Figure 1, a factor $P(x_{1\text{ down}} \mid x_{1\text{ across}}, x_{8\text{ across}}, x_{10\text{ across}})$ is introduced for the first entry in the down direction, among others. When evaluating the model on a crossword puzzle, this would correspond to first generating predictions for all entries in the across direction, and then generating predictions for all entries in the down direction making use of the tiles already completed.

An important requisite for this technique is that the estimated probability distributions $P(x_i)$ on individual entries are close to ground truth, so that good choices are made for the conditional distributions $P(x_i \mid S_i)$ which can be heavily dependent on S_i. If the model often makes incorrect predictions on individual entries, then in fact imposing the intersections between words can *decrease* overall accuracy, by ruling out correct solutions from ever being selected. We have opted not to pursue this line of inquiry here, as our models' accuracies on individual entries did not reach a sufficiently high level. For a treatment of automated crossword solvers where the effects of intersections are imposed, see Littman et al. (2002) (cf. constraint satisfaction).

As another possible extension to our work, we note that we have only considered a single word embedding function, namely the Word2Vec model pre-trained on the Google News corpus, specified in Mikolov et al. (2013). While providing good baseline performance across a number of disparate tasks (Baroni et al., 2014), this model does have several flaws when applied to crosswords.

First, Word2Vec by default does not handle homonyms particularly well. Words which are spelt the same but have different meanings, rather than being represented as distinct vectors, are collapsed into a single vector in the embedding space. This is problematic for synonymous clues in crosswords, which typically are between one and four words in length making disambiguation of homonyms from context difficult. This shortcoming is addressed in AdaGram (Bartunov et al., 2016), an extension of Word2Vec's skip-gram model that supports multiple meanings for a single word. Using AdaGram it would be possible to return a ranking of solutions under each possible sense of the clue words, providing a bigger pool of likely candidate answers and thereby improving the median ranks of the models' predictions.

Second, Word2Vec has a limited vocabulary and is unable to recognise certain clue words and solutions from *The Guardian* quick crosswords. Clue words can be handled by simply being ignored if missing from the vocabulary, although handling solutions is slightly more involved. The approach taken in Ernandes et al. (2005) is to predict heads missing from the vocabulary using a joint probability over tetragrams, estimated from the corpus of crossword answers, and maximising this probability over known characters imposed by other solutions. We propose that an alternative would be to use fastText (Bojanowski et al., 2016) as the embedding model. Instead of grouping together words that appear in the same contexts by assigning vectors with high cosine similarities, fastText groups together character n-grams appearing in the same contexts. This can be employed when making predictions from clue words by considering all the candidate words returned and forming valid tokens from the most probable n-grams. For instance, this could be used to predict the solution *'WHALING'* even if the word does not appear in the embedding space vocabulary, given that 1) *'WHALE'* does appear in the vocabulary and 2) it's returned as a highly-ranked synonym of the clue words (e.g. *'CATCHING AQUATIC MAMMAL WITH HARPOON'*), by virtue of combining the tetragrams *'WHAL'* and *'LING'*.

A final extension that could be considered is adding a decoder to the RNN architecture. An encoder-decoder LSTM functions by first combining a sequence of tokens into a single vector representation (referred to as the encoder), and then converting the vector back into a new sequence of predicted tokens (the decoder), not necessarily the same length as the original sequence (Cho et al., 2014). This could facilitate handling of unseen multiple-word answers, which are not supported by the encoder-only LSTM architecture. We can thus draw on a wider vocabulary than that of the pre-trained language models.

Acknowledgments

The authors wish to thank Guardian News & Media Ltd. for their permission to re-use crosswords 10,000–15,000 under their Open Licence scheme, and Jack Parry for repository access and helpful insights into his LSTM model implementations.

References

Gianni Barlacchi, Massimo Nicosia, and Alessandro Moschitti. 2014a. Learning to rank answer candidates for automatic resolution of crossword puzzles. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 39–48.

Gianni Barlacchi, Massimo Nicosia, and Alessandro Moschitti. 2014b. A retrieval model for automatic resolution of crossword puzzles in italian language. In *The First Italian Conference on Computational Linguistics CLiC-it 2014*, page 33.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. *Don't count, predict* a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 238–247.

Sergey Bartunov, Dmitry Kondrashkin, Anton Osokin, and Dmitry P. Vetrov. 2016. Breaking sticks and ambiguities with adaptive skip-gram. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 130–138.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv:1607.04606v2*.

Kyunghyun Cho, Bart van Merri enboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Marco Ernandes, Giovanni Angelini, and Marco Gori. 2005. Webcrow: A web-based system for crossword solving. In *AAAI*, pages 1412–1417.

Sam Hage. 2016. Regis fillbin: A crossword puzzle solver. Master's thesis, Middlebury College.

Felix Hill, Kyunghyun Cho, Anna Korhonen, and Yoshua Bengio. 2016. Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics*, 4:17–30.

AR Jobin, Anand G Menon, Ashwin Sekhar, and Vinay Damodaran. 2017. Key to crossword solving: Nlp. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 929–933. IEEE.

Aditya Kalyanpur, Branimir K Boguraev, Siddharth Patwardhan, J William Murdock, et al. 2012. Structured data and inference in deepqa. *IBM Journal of Research and Development*, 56(3.4):10–1.

Michael L Littman, Greg A Keim, and Noam Shazeer. 2002. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(1-2):23–55.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Alessandro Moschitti, Massimo Nicosia, and Gianni Barlacchi. 2015. Sacry: Syntax-based automatic crossword puzzle resolution system. In *Proceedings of ACL-IJCNLP 2015 System Demonstrations*, pages 79–84.

Massimo Nicosia, Gianni Barlacchi, and Alessandro Moschitti. 2015. Learning to rank aggregated answers for crossword puzzles. In *European Conference on Information Retrieval*, pages 556–561. Springer.

Massimo Nicosia and Alessandro Moschitti. 2016. Crossword puzzle resolution in italian using distributional models for clue similarity. In *IIR*.

Jack Parry. 2018. Finding the answers with definition models. *arXiv preprint arXiv:1809.00224*.

Titus DM Purdin and Geoff Harris. 1993. A genetic-algorithm approach to solving crossword puzzles. In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, pages 263–270.

Dragomir Radev, Rui Zhang, Steve Wilson, Derek Van Assche, Henrique Spyra Gubert, Alisa Krivokapic, MeiXing Dong, Chongruo Wu, Spruce Bondera, Luke Brandl, et al. 2016. Cruciform: Solving crosswords with natural language processing. *arXiv preprint arXiv:1611.02360*.

Aliaksei Severyn, Massimo Nicosia, Gianni Barlacchi, and Alessandro Moschitti. 2015. Distributional neural networks for automatic resolution of crossword puzzles. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 199–204.

Noam M Shazeer, Michael L Littman, and Greg A Keim. 1999. Solving crossword puzzles as probabilistic constraint satisfaction. In *AAAI/IAAI*, pages 156–162.