# Distributional Neural Networks for Automatic Resolution of Crossword Puzzles

**Aliaksei Severyn**[*], **Massimo Nicosia, Gianni Barlacchi, Alessandro Moschitti**[†]
DISI - University of Trento, Italy
[†]Qatar Computing Research Institute, Hamad Bin Khalifa University, Qatar
[*]Google Inc.
{aseveryn,gianni.barlacchi,m.nicosia,amoschitti}@gmail.com

## Abstract

Automatic resolution of Crossword Puzzles (CPs) heavily depends on the quality of the answer candidate lists produced by a retrieval system for each clue of the puzzle grid. Previous work has shown that such lists can be generated using Information Retrieval (IR) search algorithms applied to the databases containing previously solved CPs and reranked with tree kernels (TKs) applied to a syntactic tree representation of the clues. In this paper, we create a labelled dataset of 2 million clues on which we apply an innovative Distributional Neural Network (DNN) for reranking clue pairs. Our DNN is computationally efficient and can thus take advantage of such large datasets showing a large improvement over the TK approach, when the latter uses small training data. In contrast, when data is scarce, TKs outperform DNNs.

## 1 Introduction

Automatic solvers of CPs require accurate list of answer candidates to find good solutions in little time. Candidates can be retrieved from the DBs of previously solved CPs (CPDBs) since clues are often reused, and thus querying CPDBs with the target clue allows us to recuperate the same (or similar) clues.

In this paper, we propose for the first time the use of Distributional Neural Networks to improve the ranking of answer candidate lists. Most importantly, we build a very large dataset for clue retrieval, composed of 2,000,493 clues with their associated answers, i.e., this is a supervised corpus where large scale learning models can be developed and tested. This dataset is an interesting resource that we make available to the research community[1]. To assess the effectiveness of our DNN model, we compare it with the current state of the art model (Nicosia et al., 2015) in reranking CP clues, where tree kernels (Moschitti, 2006) are used to rerank clues according to their syntactic/semantic similarity with the query clue.

The experimental results on our dataset demonstrate that:

(i) DNNs are efficient and can greatly benefit from large amounts of data;

(ii) when DNNs are applied to large-scale data, they largely outperform traditional feature-based rerankers as well as kernel-based models; and

(iii) if limited training data is available for training, tree kernel-based models are more accurate than DNNs

## 2 Clue Reranking Models for CPs

In this section, we briefly introduce the general idea of CP resolution systems and the state-of-the-art models for reranking answer candidates.

### 2.1 CP resolution systems

The main task of a CP resolution system is the generation of candidate answer lists for each clue of the target puzzle (Littman et al., 2002). Then a solver for Probabilistic-Constraint Satisfaction Problems, e.g., (Pohl, 1970), tries combinations of letters that satisfy the crossword constraints. The combinations are derived from words found in dictionaries or in the lists of answer candidates. The latter can be generated using large crossword databases as well as several expert modules accessing domain-specific databases (e.g., movies, writers and geography). WebCrow, one of the

[*] Work done when student at University of Trento

[1] http://ikernels-portal.disi.unitn.it/projects/webcrow/

| Rank | Clue | Answer |
|---|---|---|
| 1 | Actress Pflug who played Lt. Dish in "MASH" | Jo Ann |
| 2 | Actress Pflug who played in "MASH" (1970) | Jo Ann |
| 3 | Actress Jo Ann | Pflug |
| 4 | MASH Actress Jo Ann | Pflug |
| 5 | MASH | Crush |

Table 1: Candidate list for the query clue: *Jo Ann who played Lt. "Dish" in 1970's "MASH"* (answer: Pflug)

best systems (Ernandes et al., 2005), incorporates knowledge sources and an effective clue retrieval model from DB. It carries out basic linguistic analysis such as part-of-speech tagging and lemmatization and takes advantage of semantic relations contained in WordNet, dictionaries and gazetteers. It also uses a Web module constituted by a search engine (SE), which can retrieve text snippets related to the clue.

Clearly, lists of better quality, i.e., many correct candidates in top positions, result in higher accuracy and speed of the solver. Thus the design of effective answer rankers is extremely important.

## 2.2 Clue retrieval and reranking

One important source of candidate answers is the DB of previously solved clues. In (Barlacchi et al., 2014a), we proposed the BM25 retrieval model to generate clue lists, which were further refined by applying our reranking models. The latter promote the most similar, which are probably associated with the same answer of the query clue, to the top. The reranking step is important because SEs often fail to retrieve the correct clues in the first position. For example, Table 1 shows the first five clues retrieved for the query clue: *Jo Ann who played Lt. "Dish" in 1970's "MASH"*. BM25 retrieved the wrong clue, *Actress Pflug who played Lt. Dish in "MASH"*, at the top since it has a larger bag-of-words overlap with the query clue.

## 2.3 Reranking with Kernels

We applied our reranking framework for question answering systems (Moschitti, 2008; Severyn and Moschitti, 2012; Severyn et al., 2013a; Severyn et al., 2013b; Severyn and Moschitti, 2013). This retrieves a list of related clues by using the target clue as a query in an SE (applied to the Web or to a DB). Then, both query and candidates are represented by shallow syntactic structures (generated by running a set of NLP parsers) and traditional similarity features which are fed to a kernel-based reranker. Hereafter, we give a brief description of our models for clue reranking whereas the reader can refer to our previous work (Barlacchi et al., 2014a; Nicosia et al., 2015; Barlacchi et al., 2014b) for more specific details.

Given a query clue $q_c$ and two retrieved clues $c_1$, $c_2$, we can rank them by using a classification approach: the two clues $c_1$ and $c_2$ are reranked by comparing their classification scores: SVM($\langle q, c_1\rangle$) and SVM($\langle q, c_2\rangle$). The SVM classifier uses the following kernel applied to two pairs of query/clues, $p = \langle q, c_i\rangle$ and $p' = \langle q', c'_j\rangle$:

$$K(p, p') = TK(q, q') + TK(c_i, c'_j) + FV(q, c_i) \cdot FV(q', c'_j),$$

where TK can be any tree kernel, e.g., the syntactic tree kernel (STK) also called SST by Moschitti (2006), and FV is the feature vector representation of the input pair, e.g., $\langle q, c_i\rangle$ or $\langle q', c'_j\rangle$. STK maps trees into the space of all possible tree fragments constrained by the rule that the sibling nodes from their parents cannot be separated. It enables the exploitation of structural features, which can be effectively combined with more traditional features (described hereafter).

**Feature Vectors (FV)**. We compute the following similarity features between clues: (i) tree kernel similarity applied to intra-pairs, i.e., between the query and the retrieved clues; (ii) DKPro Similarity, which defines features used in the context of the Semantic Textual Similarity (STS) challenge (Bär et al., 2013); and (iii) WebCrow features (WC), which are the similarity measures computed on the clue pairs by WebCrow (using the Levenshtein distance) and the SE score.

## 3 Distributional models for clue reranking

The architecture of our distributional matching model for measuring similarity between clues is presented in Fig. 1. Its main components are:

(i) sentence matrices $\mathbf{s}_{c_i} \in \mathbb{R}^{d\times|\mathbf{c}_i|}$ obtained by the concatenation of the word vectors $\mathbf{w}_j \in \mathbb{R}^d$ (with $d$ being the size of the embeddings) of the corresponding words $w_j$ from the input clues $\mathbf{c}_i$;

(ii) a distributional sentence model $f : \mathbb{R}^{d\times|\mathbf{c}_i|} \rightarrow \mathbb{R}^m$ that maps the sentence

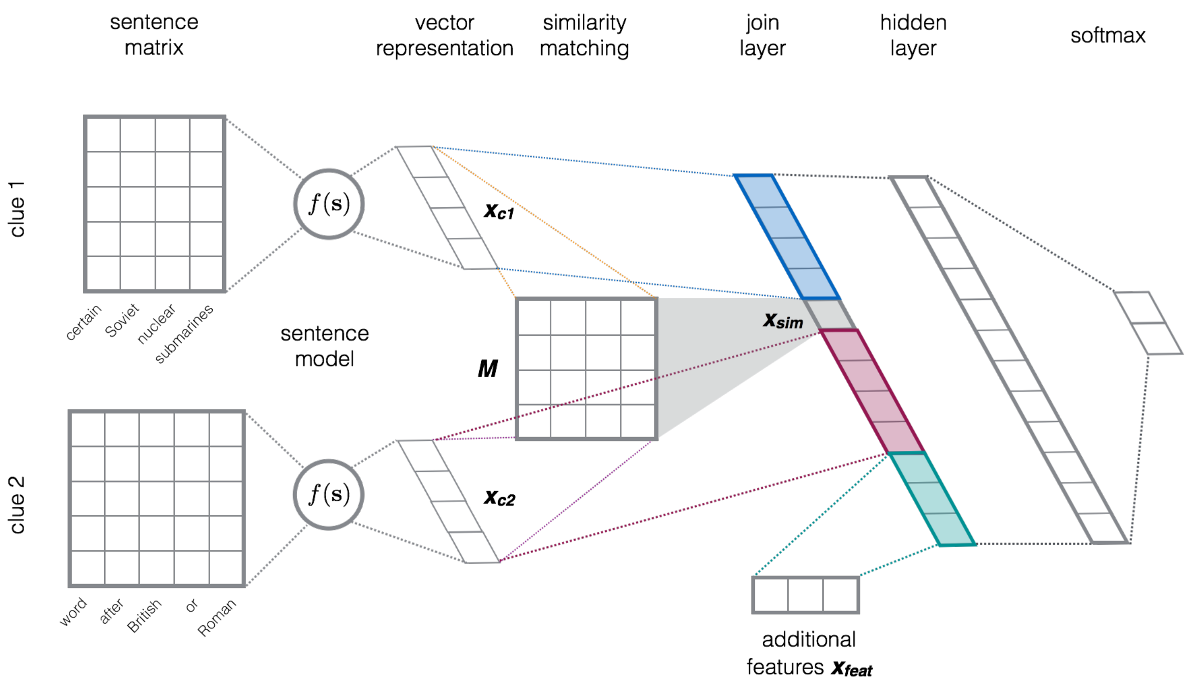

Figure 1: Distributional sentence matching model for computing similarity between clues.

matrix of an input clue $\mathbf{c}_i$ to a fixed-size vector representations $\mathbf{x}_{c_i}$ of size $m$;

(iii) a layer for computing the similarity between the obtained intermediate vector representations of the input clues, using a similarity matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ – an intermediate vector representation $\mathbf{x}_{c_1}$ of a clue $\mathbf{c}_1$ is projected to a $\tilde{\mathbf{x}}_{c_1} = \mathbf{x}_{c_1}\mathbf{M}$, which is then matched with $\mathbf{x}_{c_2}$ (Bordes et al., 2014), i.e., by computing a dot-product $\tilde{\mathbf{x}}_{c_1}\mathbf{x}_{c_2}$, thus resulting in a single similarity score $x_{\text{sim}}$;

(vi) a set of fully-connected hidden layers that models the similarity between clues using their vector representations produced by the sentence model (also integrating the single similarity score from the previous layer); and

(v) a *softmax* layer that outputs probability scores reflecting how well the clues match with each other.

The choice of the sentence model plays a crucial role as the resulting intermediate representations of the input clues will affect the successive step of computing their similarity. Recently, distributional sentence models, where $f(\mathbf{s})$ is represented by a sequence of convolutional-pooling feature maps, have shown state-of-the-art results on many NLP tasks, e.g., (Kalchbrenner et al., 2014; Kim, 2014). In this paper, we opt for a simple solution where $f(\mathbf{s}_{\mathbf{c}_i}) = \sum_i \mathbf{w}_i / |\mathbf{c}_i|$, i.e., the word vectors, are averaged to a single fixed-sized vector $\mathbf{x} \in \mathbb{R}^d$. Our preliminary experiments revealed that this simpler model works just as well as more complicated single or multi-layer convolutional architectures. We conjecture that this is largely due to the nature of the language used in clues, which is very dense and where the syntactic information plays a minor role.

Considering recent deep learning models for matching sentences, our network is most similar to the models in Hu et al. (2014) applied for computing sentence similarity and in Yu et al.(2014) (answer sentence selection in Question Answering) with the following differences:

(i) In contrast to more complex *convolutional* sentence models explored in (Hu et al., 2014) and in (Yu et al., 2014), our sentence model is composed of a single averaging operation.

(ii) To compute the similarity between the vector representation of the input sentences, our network uses two methods: (i) computing the similarity score obtained by transforming one clue into another using a similarity matrix $\mathbf{M}$ (explored in (Yu et al., 2014)), and (ii) directly modelling interactions between intermediate vector representations of the input clues via fully-connected hidden layers (used by (Hu et al., 2014)).

## 4 Experiments

Our experiments compare different ranking models, i.e., BM25 as the IR baseline, and several rerankers, and our distributional neural network (DNN) for the task of clue reranking.

### 4.1 Experimental setup

**Data.** We compiled our crossword corpus combining (i) CPs downloaded from the Web[2] and (ii) the clue database provided by Otsys[3]. We removed duplicates, fill-in-the-blank clues (which are better solved by using other strategies) and clues representing anagrams or linguistic games.

We collected over 6.3M pairs of clue/answer and after removal of duplicates, we obtained a compressed dataset containing 2M unique and standard clues, with associated answers, which we called CPDB. We used these clues to build a Small Dataset (SD) and a Large Dataset (LD) for reranking. The two datasets are based on pairs of clues: query and retrieved clues. Such clues are retrieved using a BM25 model on CPDB.

For creating SD, we used 8k clues that (i) were randomly extracting from CPDB and (ii) satisfying the property that at least one correct clue (i.e., having the same answer of the query clue) is in the first retrieved 10 clues (of course the query clue is eliminated from the ranked list provided by BM25). In total we got about 120K examples, 84,040 negative and 35,960 positive clue[4].

For building LD, we collected 200k clues with the same property above. More precisely we obtained 1,999,756 pairs (10$\times$200k minus few problematic examples) with 599,025 positive and 140,0731 negative pairs of queries with their retrieved clues. Given the large number of examples, we only used such dataset in classification modality, i.e., we did not form reranking examples (pairs of pairs).

[2]http://www.crosswordgiant.com

[3]http://www.otsys.com/clue

[4]A true reranker should be built using pairs of clue pairs, where the positive pairs are those having the correct pair as the first member. This led to form 127,109 reranking examples, with 66,011 positive and 61,098 negative pairs. However, in some experiments, which we do not report in the paper, we observed that the performance both of the simple classifier as well as the true reranker were similar, thus we decided to use the simpler classifier.

**Structural model.** We use SVM-light-TK[5], which enables the use of structural kernels (Moschitti, 2006). We applied structural kernels to shallow tree representations and a polynomial kernel of degree 3 to feature vectors (FV).

**Distributional neural network model.** We pre-initialize the word embeddings by running the `word2vec` tool (Mikolov et al., 2013) on the English Wikipedia dump. We opt for a skipgram model with window size 5 and filtering words with frequency less than 5. The dimensionality of the embeddings is set to 50. The input sentences are mapped to fixed-sized vectors by computing the average of their word embeddings. We use a single non-linear hidden layer (with rectified linear (ReLU) activation function) whose size is equal to the size of the previous layer.

The network is trained using SGD with shuffled mini-batches using the Adagrad update rule (Duchi et al., 2011). The batch size is set to 100 examples. We used 25 epochs with early stopping, i.e., we stop the training if no update to the best accuracy on the `dev` set (we create the dev set by allocating 10% of the training set) is made for the last 5 epochs. The accuracy computed on the `dev` set is the Mean Average Precision (MAP) score. To extract the DNN features we simply take the output of the hidden layer just before the softmax.

**Evaluation.** We used standard metrics widely used in QA: the Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP).

### 4.2 Results

Table 2 summarizes the results of our different reranking models trained on a small dataset (SD) of 120k examples and a large dataset (LD) with 2M examples.

The first column reports the BM25 result; the second column shows the performance of SVM perf (SVM$_p$), which is a very fast variant of SVM, using FV; the third column reports the state-of-the-art model for crossword clue reranking (Nicosia et al., 2015), which uses FV vector and tree kernels, i.e., SVM(TK).

Regarding the other systems: DNNM$_{SD}$ is the DNN model trained on the small data (SD) of 120k training pairs; SVMp(DNNF$_{LD}$) is SVM perf trained with (i) the features derived from

[5]`http://disi.unitn.it/moschitti/Tree-Kernel.htm`

**Training classifiers with the Small Dataset (SD)** (120K instances)

| | BM25 | SVMp | SVM(TK) | DNNM$_{SD}$ | SVMp(DNNF$_{LD}$) | SVM(DNNF$_{LD}$,TK) |
|---|---|---|---|---|---|---|
| MRR | 37.57 | 41.95 | 43.59 | 40.08 | 46.12 | 45.50 |
| MAP | 27.76 | 30.06 | 31.79 | 28.25 | 33.75 | 33.71 |

**Training classifiers with the Large Dataset (LD)** (2 million instances)

| | BM25 | SVMp | SVM(TK) | DNNM$_{LD}$ | SVMp(DNNF$_{LD}$,$-$FV) | SVMp(DNNF$_{LD}$) |
|---|---|---|---|---|---|---|
| MRR | 37.57 | 41.47 | – | 46.10 | 46.36 | 46.27 |
| MAP | 27.76 | 29.95 | – | 33.81 | 34.07 | 33.86 |

Table 2: SVM models and DNN trained on 120k (small dataset) and 2 millions (large dataset) examples. Feature vectors are used with all models except when indicated by $-$FV

DNN trained on a large clue dataset $LD$ and (ii) the FV; and finally, SVM(DNNF$_{LD}$,TK) is SVM using DNN features (generated from LD), FV and TK. It should be noted that:

(i) SVM$_p$ is largely improved by TK;

(ii) DNNM$_{SD}$ on relatively small data delivers an accuracy lower than FV;

(iii) if SVM$_p$ is trained with DNNM$_{LD}$, i.e., features derived from the dataset of 2M clues, the accuracy greatly increases; and

(iv) finally, the combination with TK, i.e., SVM(DNNF$_{LD}$,TK), does not significantly improve the previous results.

In summary, when a dataset is relatively small DNNM fails to deliver any noticeable improvement over the SE baseline even when combined with additional similarity features. SVM and TK models generalize much better on the smaller dataset.

Additionally, it is interesting to see that training an SVM on a small number of examples enriched with the features produced by a DNN trained on large data gives us the same results of DNN trained on the large dataset. Hence, it is desired to use larger training collections to build an accurate distributional similarity matching model that can be then effectively combined with other feature-based or tree kernel models, although at the moment the combination does not significantly improve TK models.

Regarding the LD training setting it can be observed that:

(i) the second column shows that adding more training examples to SVM$_p$ does not increase accuracy (compared with SD result);

(ii) DNNM$_{LD}$ delivers high accuracy suggesting that a large dataset is essential to its training; and

(iii) again SVM$_p$ using DNN features deliver state-of-the-art accuracy independently of using or not additional features (i.e., see $-$FV, which excludes the latter).

## 5 Conclusions

In this paper, we have explored various reranker models to improve automatic CP resolution. The most important finding is that our distributional neural network model is very effective in establishing similarity matching between clues. We combine the features produced by our DNN model with other rerankers to greatly improve over the previous state-of-the-art results. Finally, we collected a very large dataset composed of 2 millions clue/answer pairs that can be useful to the NLP community for developing semantic textual similarity models.

Future research will be devoted to find models to effectively combine TKs and DNN. In particular, our previous model exploiting Linked Open Data in QA (Tymoshenko et al., 2014) seems very promising to find correct answer to clues. This as well as further research will be integrated in our CP system described in (Barlacchi et al., 2015).

## Acknowledgments

This work was supported by the EC project CogNet, 671625 (H2020-ICT-2014-2). The first author was supported by the Google Europe Doctoral Fellowship Award 2013.

## References

Daniel Bär, Torsten Zesch, and Iryna Gurevych. 2013. DKPro similarity: An open source framework for text similarity. In *Proceedings of ACL (System Demonstrations)*.

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*. Association for Computational Linguistics.

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*.

Gianni Barlacchi, Massimo Nicosia, and Alessandro Moschitti. 2015. SACRY: Syntax-based automatic crossword puzzle resolution system. In *Proceedings of 53nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Beijing, China, July. Association for Computational Linguistics.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620, Doha, Qatar, October. Association for Computational Linguistics.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159.

Marco Ernandes, Giovanni Angelini, and Marco Gori. 2005. Webcrow: A web-based system for crossword solving. In *In Proc. of AAAI 05*, pages 1412–1417. Menlo Park, Calif., AAAI Press.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *NIPS*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, June.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. Doha, Qatar.

Michael L. Littman, Greg A. Keim, and Noam Shazeer. 2002. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(12):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 26*, pages 3111–3119.

Alessandro Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *ECML*, pages 318–329.

Alessandro Moschitti. 2008. Kernel methods, syntax and semantics for relational text categorization. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 253–262, Napa Valley, California, USA.

Massimo Nicosia, Gianni Barlacchi, and Alessandro Moschitti. 2015. Learning to rank aggregated answers for crossword puzzles. In Allan Hanbury, Gabriella Kazai, Andreas Rauber, and Norbert Fuhr, editors, *Advances in Information Retrieval - 37th European Conference on IR Research, ECIR, Vienna, Austria. Proceedings*, volume 9022 of *Lecture Notes in Computer Science*, pages 556–561.

Ira Pohl. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(34):193 – 204.

Aliaksei Severyn and Alessandro Moschitti. 2012. Structural relationships for large-scale learning of answer re-ranking. In *Proceedings of ACM SIGIR*, New York, NY, USA.

Aliaksei Severyn and Alessandro Moschitti. 2013. Automatic feature engineering for answer selection and extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 458–467, Seattle, Washington, USA, October. Association for Computational Linguistics.

Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. 2013a. Building structures from classifiers for passage reranking. In *CIKM*.

Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. 2013b. Learning adaptable patterns for passage reranking. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 75–83, Sofia, Bulgaria, August. Association for Computational Linguistics.

Kateryna Tymoshenko, Alessandro Moschitti, and Aliaksei Severyn. 2014. Encoding semantic resources in syntactic structures for passage reranking. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 664–672, Gothenburg, Sweden, April. Association for Computational Linguistics.

Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. 2014. Deep learning for answer sentence selection. *CoRR*.