Exploring Transformers for Answering Crossword Clues

Mike Chaberski

NYU Tandon School of Engineering
mac937@nyu.edu

Abstract

The design and implementation of transformer models for answering crossword puzzle clues is explored herein. A word-generating model and a letter-generating model are defined and tuned based on training results. The models are evaluated relative to a baseline non-neural-network algorithm. Model definition, training, and evaluation code is available at https://github.com/mike10004/csgy6953-fp1.

Introduction

Humans solve crossword puzzles with a variety of skills, including language facility, trivia knowledge, and lateral thinking. Transformer models show adeptness at mining training data for obsfuscated connections among samples in tasks such as language translation. Therefore, it is plausible that a transformer trained on crossword clue-answer pairs could discover approximations of the skills required to answers crossword clues correctly.

This work explores the design and implementation of transformer models trained to answer crossword clues. Two models are developed, one that suggests a single word given an input clue, and one that suggests a sequence of letters. The definition and training of these models is detailed herein. The letter model requires engineering beyond the training step, as additional considerations must be made for the generator component that produces sequences of letters in response to a clue.

Related Work

The intuition behind the use of transformers for answering clues is the robust performance of transformer models at the machine translation task (Vaswani et al. 2017). Mapping crossword clues to answers is a sequence-to-sequence translation task, with the caveats that the source sequences (clues) are generally terse and often contain misdirection, and the target sequences (answers) are generally very short relative to the length of the source sequence.

The end-to-end task of solving a complete crossword grid is notably tackled by the Berkeley Crossword Solver, which produced an algorithm that outperformed humans at the American Crossword Puzzle Tournament in 2021 (Wallace et al. 2022). The clue-answering component is a bi-encoder QA model designed to be used iteratively to solve harder clues given probable letters from easier crossing answers. Contemporaneous work from Kulshreshtha et al. 2022 suggests end-to-end crossword grid-solving as a benchmark for Natural Language Processing algorithm capabilities.

The task this study investigates is limited to answering clues (as opposed to populating a grid), which is a type of reverse-dictionary problem addressed by the "MultiRD" model (Zhang et al. 2019). MultiRD is a bidirectional LSTM model with a "multi-channel" approach that integrates linguistic analysis (e.g. morphemes, lexemes, and semantic units).

A statistical approach to answering clues is implemented in Parikh 2019. For a given clue, a database of clue-answer pairs is searched using a character-level similarity metric, and the corresponding answer for the best-matching clue is returned. That method is the baseline against which results of the word and letter models described herein are compared in the test phase.

Dataset

A modified version of the dataset from Kulshreshtha et al. is used for training and evaluation. The modified dataset contains only single-word answers and does not contain cross-referential clues (e.g. "See 46-Across"). The training set has 320K clue-answer pairs and 55K in the validation/test sets.

Model Selection

Model structure and training code was adapted from a transformer implementation for sequence to sequence translation (Hegde 2022). A custom generator component was developed to support ranked answer suggestions. Default hyperparameters for each model were either adopted from Vaswani et al. or inherited from the upstream codebase. Adopted hyperparameters include:

- 8 attention heads;
- embedding size 512;
- 3 encoder layers and 3 decoder layers;
- sinusoidal positional encoding;
- feed-forward network dimension 512;
- Adam optimizer (Kingma and Ba 2017) with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$; and
- dropout rate 0.1.

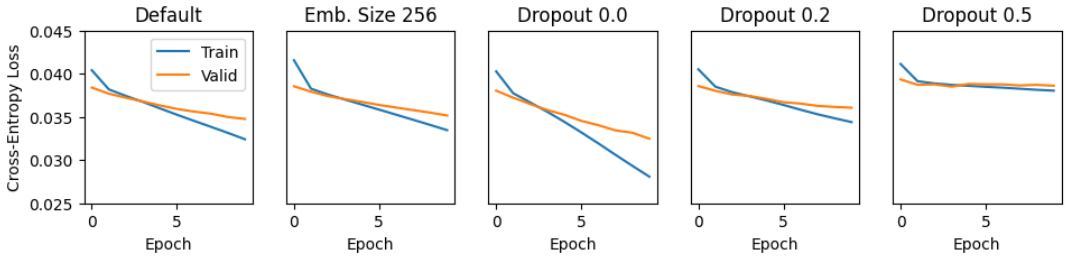

Figure 1: **Word model** training and validation loss curves under varying hyperparameters.

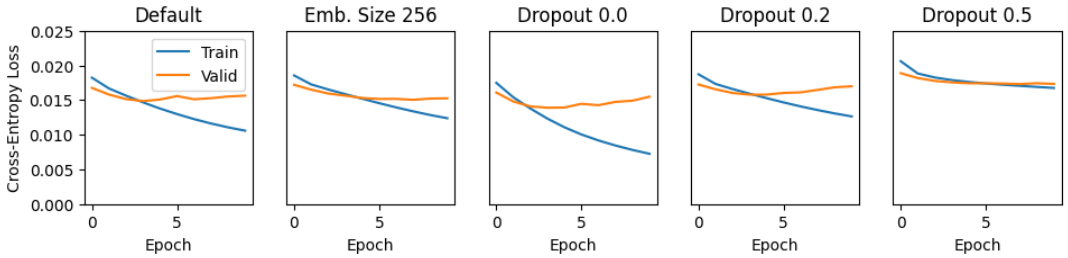

Figure 2: **Letter model** training and validation loss curves under varying hyperparameters.

The use of cross-entropy for the loss function and learning rate of 10^{-4} were inherited from the upstream codebase. Learning rates of greater magnitude were tested but in each case the loss did not trend toward convergence. The batch size of 128 was likewise taken from the upstream codebase.

Models were trained for 10 epochs. Initial results of training both the word model and the letter model showed a propensity for overfit. As such, different methods of preventing overfit by varying hyperparameters were attempted. These included regularization by varying dropout rates and reducing model dimensionality by decreasing embedding size to 256. A dropout rate of zero was also tested to determine whether dropout was beneficial for the task.

Word Model

Training and validation loss curves for the word model are shown in Figure 1. All of the curve pairs show undesirable characteristics. In the case of the default parameters and the reduced dimensionality, growing divergence between training and validation loss is evident. In the case of dropout, at a dropout rate of zero, the model achieves lower validation loss than under the default rate of 0.1. At greater dropout rates, the loss converges early at a loss value greater than under the default rate.

The model trained under default parameters and the model trained with zero dropout show similar overfit propensity, but the zero-dropout model achieves lower loss. Thus the zero-dropout model, with 110 million trainable parameters, was selected as the optimal word model for subsequent consideration.

Letter Model

Training and validation loss curves for the letter model are shown in Figure 2. As with the word model, all of the curve pairs show undesirable characteristics. Under the default parameters, the model shows substantial overfit and validation loss has increasing trajectory at later epochs. Overfit is also evident in the loss curves of the model with reduced dimensionality, but that model shows a better overall trend, which aligns with the intuition that the small target token space of the letter model would be accommodated better by a lower-dimension model.

Examination of the loss curves for models trained under varying dropout rates shows that a dropout rate of zero induces substantial overfit and an increasing validation loss trend, similar to the default model but magnified. At dropout rate 0.5 there is no evidence of overfit, but the loss appears to have converged early.

The reduced overfit but greater minimum validation loss (0.0173) of the 0.5-dropout model was weighed against the greater overfit and minimum validation loss of 0.0152 for the reduced-dimensionality model. The latter model, with 25 million trainable parameters, was selected as optimal for subsequent consideration.

Generator Strategy

Given a sequence-to-sequence transformer model, a sequence generator is the component that accepts as input a source sequence and an empty or partial output sequence and produces the next element of the output sequence. The generator is iteratively applied to produce a complete sequence. A custom generator needed to be developed for the letter model to decide what action to take at each iteration, whereas the word model always stops after generating a single word.

For each input, the generator produces an array of values corresponding to possible next tokens. The values indi-

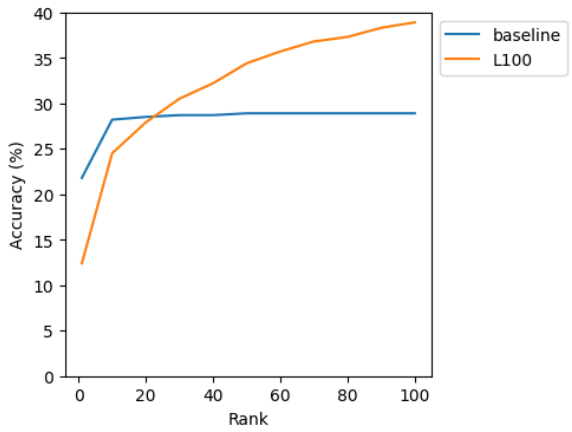

Figure 3: **Word model** accuracy using the L100 generator strategy.

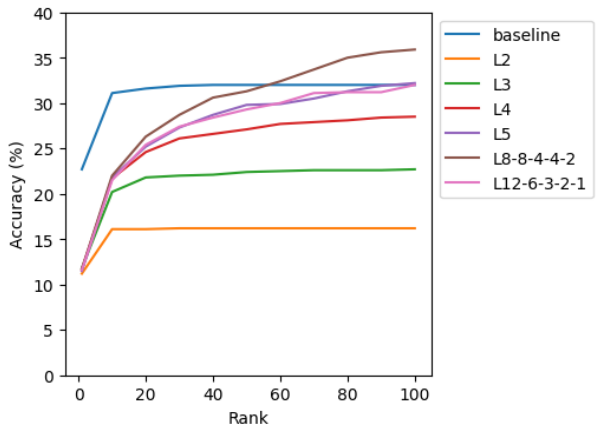

Figure 4: **Letter model** accuracy for multiple generator strategies.

cate relative likelihood. The generator implementation applies the softmax function to these values to map them to a probability distribution.

To produce a list of candidate answers for a given clue, a variation on beam search is performed (Wu et al. 2016). In beam search, the tree of sequence possibilities is traversed and the resulting answers are ranked by their aggregate probability.

At each node of the tree, the possible next tokens are considered in order of probability, descending, up to a limit parameter (number of "beams"). A given tree search strategy may be identified by the sequence of these limit parameters, so a strategy that looks at the top 10 tokens, then top 5, then top 3, would be identified by the string L10-5-3. For brevity, the last number in the identifying sequence is assumed for all subsequent decision nodes, so a strategy that considers 2 tokens at every node is identified by the string L2. (Because the sequences the word model may generate are at most one token, only the L100 strategy is evaluated for the word model.)

Performance under different strategies for generating answers is shown in Figure 3 and Figure 4 by evaluating their performance in producing correct answers. Because this generator strategy is exponential in runtime by nature, evaluation was performed on a random 1000-pair subset of the validation set. The letter model was evaluated on a subset that contains only answers up to 5 letters in length. Performance is measured by calculating the percentage of correct answers at incremental ranks. For example, the accuracy at rank 10 is the percentage of clue-answer pairs for which the correct answer is among the top 10 candidate answers.

Both models are outperformed by the baseline at rank less than 20. The word model shows a generally positive trend, and the fact that it is achieving near 40% accuracy at rank 100, when the size of the answer set is about 50K, shows that the model is indeed learning effectively.

With respect to the letter model, the lower-order strategies do not achieve significant accuracy, and their accuracy with respect to rank has is bounded by their limited search space. (Thus the accuracy curves level off at a certain point.) The higher-order strategies achieve some success, outperforming the baseline at ranks above 30. Strategies that are broadly less greedy, meaning they consider more tokens at greater depths in the search tree, appear to outperform greedier strategies with greater initial degrees. In particular, strategy L5 performs well considering its low-degree search space, and its efficiency recommends it as the optimal strategy for subsequent evaluation in the test phase. While it is not the best performer in this assessment, its broader consideration of tokens at high depths means greater generalizability, which is desirable for the test phase.

Results

For the test phase, subsets of the test set were constructed, with 10K pairs for the word model and 1K pairs for the letter model. The letter model set includes only pairs with answers of at most 5 letters. The results are shown in Figure 5.

The performance difference in the baselines is presumably because clues with shorter answers are more easily matched by the baseline algorithm. The word model outperforms the baseline at ranks above 20, and the letter model outperforms the baseline at ranks above 40.

Conclusion

The models achieve success in outperforming the baseline, but the baseline is a low bar. Comparison to the MultiRD model or a Word2Vec-based algorithm (Mikolov et al. 2013) with a more robust word similarity metric would provide a better gauge of relative performance. (Performance of the MultiRD model could not be reproduced at the accuracy level reported by Zhang et al., so it was deemed unfair to include as a baseline herein.)

The initial objectives from the project proposal included attempting to utilize constraints such as target sequence length and known letters, but given the complexity of the initial tasks of model training and generator implementation, pursuing those objectives adequately would have resulted in

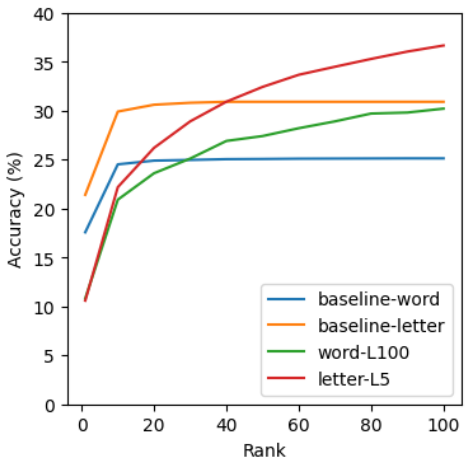

Figure 5: Accuracy of selected models and strategies on test subsets.

more cursory investigation of the core methods that were explored herein. Use of positional encoding to influence output sequence length (Takase and Okazaki 2019) seems like a promising way to achieve better accuracy at lower ranks. The known-letters constraint can be seen as a specific instance of the general problem of unmasking partially-masked sequences given an input context, which is tackled by Raffel et al. 2023, among others.

The fundamental model training task proved difficult. It is possible that propensity toward overfit is an inherent characteristic of crossword clue/answer datasets, relative to datasets for language translation on which transformers have shown robustness. Clue-answer pairs are inherently asymmetric in terms of sequence length, whereas language-to-language pairs are more balanced. This suggests consideration of alternative positional encodings for the answer sequences in the letter model; given their short length, perhaps a sinusoidal encoding is not appropriate. (Tests that completely removed the letter model's positional encoding from the target sequence were performed, but the results were poor.) Kulshreshtha et al. claim that no major transformer architecture supports character-level outputs, so this was likely a more complex task than initially assumed.

While regularization was attempted through reduction of model dimensionality and increasing dropout, other techniques, such as data augmentation (perhaps by randomly masking some tokens in the clue sequences) are worth trying. Scheduling learning rate adjustments differently is also worthy of investigation; Vaswani et al. schedule the learning rate with a warmup period and gradual decrease. They train for many more epochs with a much larger dataset, but even for this dataset, non-constant learning rate schedules are worthy of consideration.

Overall, the results herein show that the models did learn effectively, though imperfectly. Though there is no conclusive evidence that the transformer architecture is optimal for the clue-answering task, relative to other types of encoder-decoder models, several avenues of further research are promising.

References

Hegde, C. 2022. Deep Learning Demos. https://github.com/chinmayhegde/dl-demos. Accessed: 2024-04-16.

Kingma, D. P.; and Ba, J. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980.

Kulshreshtha, S.; Kovaleva, O.; Shivagunde, N.; and Rumshisky, A. 2022. Down and Across: Introducing Crossword-Solving as a New NLP Benchmark. arXiv:2205.10442.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781.

Parikh, P. 2019. Crossword Solver. https://github.com/pncnmnp/Crossword-Solver. Accessed: 2024-04-23.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2023. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683.

Takase, S.; and Okazaki, N. 2019. Positional Encoding to Control Output Sequence Length. arXiv:1904.07418.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 6000–6010. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510860964.

Wallace, E.; Tomlin, N.; Xu, A.; Yang, K.; Pathak, E.; Ginsberg, M.; and Klein, D. 2022. Automated Crossword Solving. arXiv:2205.09665.

Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; Klingner, J.; Shah, A.; Johnson, M.; Liu, X.; Łukasz Kaiser; Gouws, S.; Kato, Y.; Kudo, T.; Kazawa, H.; Stevens, K.; Kurian, G.; Patil, N.; Wang, W.; Young, C.; Smith, J.; Riesa, J.; Rudnick, A.; Vinyals, O.; Corrado, G.; Hughes, M.; and Dean, J. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144.

Zhang, L.; Qi, F.; Liu, Z.; Wang, Y.; Liu, Q.; and Sun, M. 2019. Multi-channel Reverse Dictionary Model. arXiv:1912.08441.