Evaluating QANet on SQuAD 2.0

Parthasarathy Suryanarayanan
psuryan@stanford.edu

Yash Mangilal Jain
yashjain@stanford.edu

Abstract

The QANet architecture [21] was evaluated against the SQuAD [1.1] dataset [12] with impressive results (EM: 82.47 and F1: 89.30). The SQuAD [2.0] dataset [11], the latest revision of SQuAD, tests the ability of a system to not only answer the question whenever possible but also determine when no answer is supported by the paragraph and abstain from answering. Thus, it poses a much more challenging machine comprehension task. The main objective of this work is to evaluate the QANet system on the SQuAD [2.0] dataset, by implementing the architecture from the ground up and to understand the challenges posed by the non-answerability.

1 Introduction

We can define machine comprehension in terms of Question Answering in its most general form [1]. Recently a variety of neural architectures have achieved near-human performance in open domain question answering tasks. These recent advances are broadly of two types – (1) Pre-trained Contextual Embeddings (PCE) based methods such as ELMo[10], Bert[6] etc. and (2) Non-PCE methods. The former offers more of an "off-the-shelf" module that could be employed for specific tasks such as question answering, the latter, even though not being the state of the art (SoTA) any longer (as of Feb 2019), offer more scope for creativity and opportunities for deep learning practitioners to explore different techniques and develop intuitions behind them. One of the successful non-PCE approaches to machine comprehension is the QANet architecture.

The main goal of the QANet architecture is efficiency / speed which also results in performance gain as a nice by-product. The authors of the system demonstrate the efficiency and performance improvements on SQuAD [1.1] dataset. However, suitability of the architecture for the non-answerability setting has not been systematically evaluated to our knowledge. Our major focus is understanding the limitations of QANet against SQuAD [2.0] dataset and investigating methods to augment the architecture, to predict if the question can be answered or not based on the information from the comprehension.

The organization of the paper is as follows. In Section 2 we attempt to show the landscape of related work. Section 3 gives a comprehensive overview of the QANet architecture and our approach for evaluation. Section 4 describes the different experiments we conducted, and discusses our observations. In Section 5, we summarize our main conclusions from this work and describe our vision for the future.

2 Related Work

Prior to QANet, major question answering systems primarily contained either of two key ingredients (1) recurrent units such as LSTM for capturing sequential input and (2) exploiting attention mechanism for capturing long-term interactions. For example, popular non-PCE models such as, Multi-Paragraph Reading Comprehension [4], Gated Self-Matching Networks (R-Net) [19] and Match-LSTM [18] use these ideas. The BiDAF [13] model which we use as the baseline, employs both the techniques, is a hierarchical multi-stage end-to-end network which takes inputs of different granularity (character, word and phrase) to obtain a query-aware context representation using memory-less context-to-query (C2Q) and query-to-context (Q2C) attention. This representation can then be used for different final tasks, such as question answering.

However, due to the sequential nature, (1) can't be parallelized to exploit the hardware and hence is slow. To achieve the speedup, the QANet borrows a neat idea from NMT, proposed in the Transformer[17] architecture: encode the question and context separately using a non-recurrent, therefore faster, encoder. This encoder uses convolution (which models the local features) and self-attention (which models global interaction) as building

blocks. The parallel nature of CNN architectures leads to a significant speed boost, especially given the fact that context passages in SQuAD are really long.

3 Approach

The first of our two step approach is essentially implementing the architecture carefully as specified in the paper. The second step is to perform an error analysis on the development set, understand the strength and weakness of the model come up with a few ideas that might help the performance. We describe the QANet architecture in the following section at a high level. Please refer to the original paper [21] for details.

3.1 Representation

Before we dive deep into the architecture, it is useful to define the terminology and representation. Recall that the QA problem can succinctly be formulated as following: Given a context / passage of n words, $C = \{c_1, c_2..., c_n\}$ and a question / query sentence of m words, $Q = \{q_1, q_2..., q_m\}$, we want to output a span $S = \{c_i, c_{i+1}, ...c_{i+j}\}$ from the original paragraph C if the question is answerable. QANet uses fixed size contexts and questions. In the default implementation, only the first 400 tokens in the given paragraphs are used as context and 50 tokens in the given question are used as query. For each word, first 16 characters are only used to compute the character embeddings. If the context/query is shorter than the length threshold, zero-padding is applied. QANet uses pre-trained GloVe [9] vectors as word and character embeddings for representing both query and context.

3.2 Model

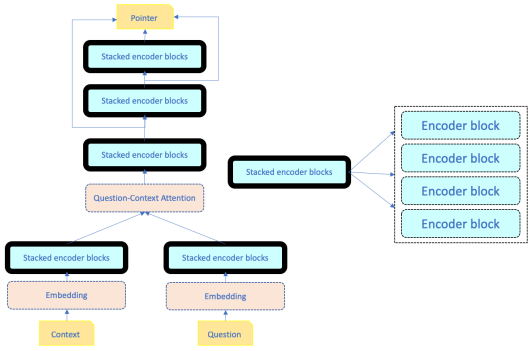


Figure 1: Model

QANet architecture, shown in Figure **??**. But before we examine the layers, we need to look at a key component that is used in several places with the model, the encoder block. The encoder block draws upon many ideas from the Transformer [17]. It is stacked several times and the stack is used in three different places: to encode the question, to encode the context and finally on encode the model 3 times. The encoder block consists of four key layers.

1. *Positional embedding layer:* The position of a word within a sentence carries useful information for the model. In an RNN, this information is trivially encoded as the inputs are presented to the model one step at time. But for non-RNN model, a "positional encoding" is needed to be given explicitly. This layer adds the positional encoding to the input at the beginning of each encoder layer as defined in Transformer
2. *Convolution layer:* This is repeated multiple times and the number varies based on the location of usage of the encounter block. the implementation uses depth-wise separable convolutions [3] which require an order of magnitude less computation to achieve nearly the same result as the normal convolution. This is a key ingredient of the speedup. (Details: kernel / filter size: 7, number of filters: 128)
3. *Self-attention layer:* the multi-head attention mechanism, again from Transformer is used with number of attention heads as 8. Attention key depth is 128 and depth per head is 16.
4. *Feed-Forward layer*

The model consists of 5 key modules: an embedding layer, an encoder layer, a context-query attention layer, a modeling layer and an output layer.

1. **Embedding layer:** combines the word and character embeddings into a singular representation for each word. This is done by,
   (a) Looking up the 300 ($p1$) dimensional pre-trained GloVe embeddings $[x_w]$
   (b) Learning a 200 ($p2$) dimensional embedding for each character in the word using convolution $[x_c]$. The character embeddings are passed through a convolutional layer and a max-pooling layer, as described in [8], to produce 200-dimensional character-level word embeddings. The intuition behind using a character-level embedding is to be able to handle out of vocabulary words (OOV), rare words, misspellings and morphology during test time.
   (c) Concatenating the two vectors into $[x_w; x_c]$ into a vector of size $(p1 + p2)$. Also, a two-layer highway network [14] is also used as optimization to avoid the vanishing gradient problem.
2. **Encoder layer:** After that, the context and the question are passed through encoder layer. This layer uses a single encoder block with 4 convolutions. The output of this layer is of 128 dimensions $(hidden\_size)$ as the input is immediately converted to this dimension by a one-dimensional convolution. **Note:** In our implementation, we convolved down from 500 dimension to $hidden\_size$ in the embedding layer itself.
3. **Context-Query Attention layer:** The output from the encoders are fed to the Context-Query attention layer which combines context and query and produces a representation for each word in context. This is similar to the previous reading comprehension models, such as [20] and [2]. The purpose of this layer is to calculate two quantities.
   (a) *Context-to-query attention (A):* First calculate pair wise similarities between words in the encoded context (C) and encoded query (Q). The similarity measure is a trilinear function as described in [20] $f(q, c) = W0[q, c, q \odot c]$. This would yield a matrix S of dimensions n x m. $W0$ is a trainable variable. After normalizing each row using $softmax$, we get a different matrix $\overline{S}$. The context-to-query-attention is calculated as $A = \overline{S} \cdot Q^T$
   (b) *Query-to-context attention (B):* Column-normalizing S yields $\overline{\overline{S}}$. The query-to-context attention is calculated as $B = S \cdot \overline{\overline{S}}^T \cdot C^T$
4. **Model encoder layer:** The input to this layer at each position is $c, a, c \odot a, c \odot b$ where a and b are rows from attention matrices A and B calculated above. This layer consists of 3 stacked encoder blocks that share weights and the number of encoder blocks within each stack is 7. The number of convolutions within each block is 2.
5. **Output layer:** The key layer for prediction is the output layer. Similar to method described in [13], the optimization function is formulated as follows. Let

$$P^1 = \textit{softmax}(W_1[M_0; M_1]) \qquad P^2 = \textit{softmax}(W_2[M_0; M_2])$$

   be the probabilities of each position in the context being start and end of the answer span, respectively. Where W_1 and W_2 are trainable weights and M_0, M_1, M_2 are outputs from the three model encoders. Now the objective function could be defined as the sum of negative log probabilities of predicted distributions indexed by true start and end indices, averaged over the training instances.

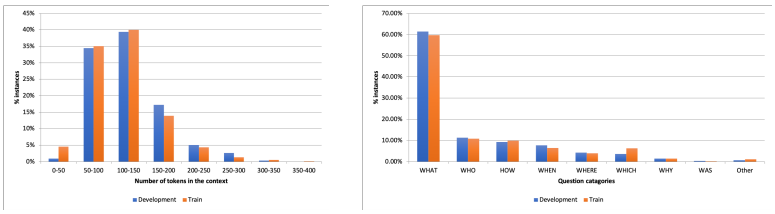
$$Log(\theta) = -1/N \sum_{i}^{N} [log(P^1_{y^1_i}) + (P^2_{y^2_i})]$$

   Where y^1_i and y^2_i are start end positions of the i^{th} training instance from ground truth. The model outputs probabilities for all pairs similar to other pointer networks for reading comprehension. Through an efficient method (such as using dynamic programming), the best pair of spans (s, e) that maximize $p^1_s p^1_e$ have to be chosen as the prediction. Ultimately these probabilities are turned into a span consisting of a start index and an end index, indicating the location of the answer in the context paragraphs. Since, the answer is essentially a contiguous phrase in the context, predicting start and end index is sufficient.

4 Experiments

4.1 Data

We now describe the SQuAD [2.0] dataset[11], customized for default final project in more detail, starting with some general statistics. The training, development and test sets consist of 129941, 5951 and 5918 questions respectively. As noted earlier, the question and context are truncated to 50 and 400 tokens respectively. Since pure Transformer based architectures have been known not to do well on very long-range dependencies [5], we take note of the distribution of the context length for the development set in Figure 2



(a) Context lengths **(b)** Question categories

Figure 2: Dataset distributions

The dataset contains approximately 50% unanswerable questions due to a range of linguistic phenomena such as negation, antonyms, entity swapping etc. However, while the dataset provides high quality questions and answers, many of them tend to be of $wh-$ type. Figure 2 shows the distribution of these question types. Also, there is some repetition of answers—of the 5951 questions in the development set 2848 questions are answerable (47.8%) with 2596 distinct answers; there are 194 repeated answers (7.4%).

4.2 Evaluation

We use the standard evaluation metrics for SQuAD - EM (Exact Match) and F1 measure. The baseline for our implementation is the Bi-DAF model supplied as part of the default final project.

4.3 Experiments

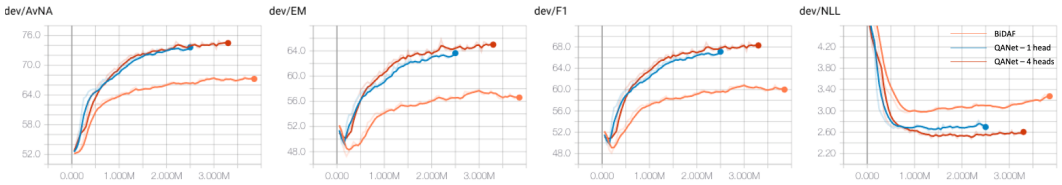


Figure 3: Experiments

In this section, we describe the experiments conducted by us to evaluate the performance of the QANet model on the SQuAD [2.0] dataset. The Figure 3 shows our main experimental runs. All the experiments were run with the hyper parameters specified in the paper except for the following changes. The $hidden\_size$ of the model was set to 96 instead of 128 due to hardware limitations. Our hyper-parameter search was limited to changes in number of heads for the multi-headed attention. We ran two experiments with 1 and 4 attention heads respectively. The hardware used for the experiment is NV6 class of GPU machines provided by Microsoft Azure with 8GB of video memory. It was observed that close 7.9GB was used. The baseline BiDAF model was run for 30 epochs. The two QANet experiments were manually stopped before 30 epochs upon observing the stabilization of dev NLL (negative log-likelihood loss). While the BiDAF was run with batch size of 64, the QANet was run with batch size of 32 due to memory constraints. The typical wall-clock time per epoch for the three models were 16, 53 and 80 minutes respectively. We found that variable context length batches (where sentences are padded to max_len) ran about 30% (2) faster than padding all sentences across batches to 400 tokens.

Also, as specified in the paper, we use three different regularization techniques.

1. L2 weight decay on all trainable kernels with $\lambda = 3e^{-7}$.
2. Dropout with $drop\_prob = 0.1$ after every other layer. Additionally dropout is applied word embedding and character embedding with $drop\_prob$ of 0.1 and 0.05 respectively.
3. Stochastic depth dropout (layer dropout) on every sublayer in an Encoder Block. For an Encoder Block with L total sublayers, during training, the $l-th$ sublayer has probability $p_l = 1 - (l/L) * drop\_prob$ of survival (i.e., earlier sublayers are more likely to survive). For the question and context encoders, $L = 6$. For each of the stacked encoder blocks within the model encoder, $L = 28$.

It is to be noted that as a departure from the QANet paper, we chose not to implement any data-augmentation due to time-constraints.

4.4 Results

Our model achieves competitive scores of $60.57/64.44$ of EM and $F1$ respectively on test set, in the non-PCE SQuAD [2.0] leaderboard. Also, the model performs considerably well on the development set with the scores of

$65.68/68.67(EM/F1)$ in the leaderboard. The differences in development and test scores could be attributed to slight differences in data distribution as our implementation had the same hyper parameters as the paper. We failed to observe any dramatic speedup over BiDAF baseline. We suspect that this is due to a combination of memory-intensive nature of the architecture on our limited hardware and some suboptimal implementation.

4.5 Analysis

Answerability: The main feature SQuAD [2.0] dataset is unanswerable questions. SQuAD [2.0] questions cannot be answered by learning simple heuristics such as word overlap or type matching [12]. To deal with unanswerable cases, systems must learn to identify a wide range of linguistic phenomena such as negation, antonymy and entity changes between the passage and the question. Therefore we expect the performance of the base QANet over SQuAD [2.0] to be worse than SQuAD [1.1]. Comparing the QANet results against the BiDAF (refer Figure 4), we find that QANet attempts to answer 62% questions from the development set and predicts the answerability wrong 28% of the time (6% worse than BiDAF). Of these incorrect answerability predictions, the model is biased towards answering a question (23%) rather than refraining (5%). Based on this headroom, we note that any improvement answerability prediction will improve overall performance considerably.

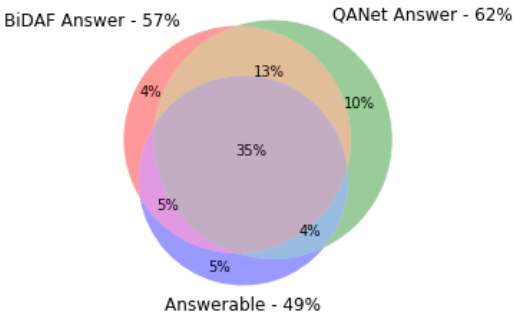


Figure 4: Answerability prediction

Attention at work: As expected, whenever a question is answered correctly by the system, the ever-dependable attention mechanism again seems to perform well inside the QANet architecture. We visualized the attention distribution in the last encoder block of the first stacked encoder block in the model encode layer. Given the question *Who was the most influential researcher among those grappling with the deficit of work surrounding the complexity posed by algorithmic problems?*, the tokens *Alan, Turing* from the context get the highest levels of attention as shown in Figure 5

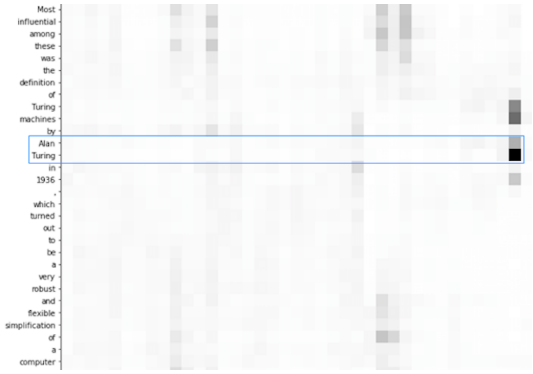


Figure 5: Attention at work
See appendix for the un-cropped figure.

5 Conclusion

In this work, we implemented the QANet model for machine reading comprehension. We evaluated the model against the SQuAD [2.0] dataset and analyzed the results. Prior to the recent PCE methods, QANet was the SoTA system for question answering. QANet is a complex model with more than 130 layers. Implementing the architecture correctly in a short span time based on the brief descriptions in the paper was a huge challenge. Our motivation behind choosing this project was to carefully study the key ideas (such as, attention mechanism, pointer networks etc.) from the recent innovations of neural architectures that the QANet is built upon. We hoped to gain hands-on experience by implementing the architecture "from the scratch". We believe we have accomplished this learning goal. However, due to time constraints, we could not further explore adapting QANet architecture for SQuAD [2.0] dataset. At the outset, we want to pursue ideas along two different dimensions:

1. *Answerability prediction improvements:* Previous works, including the framework supplied as part of the baseline implementation for default final project, usually predict an additional "no-answer" probability to detect unanswerable cases. Given a context of N tokens, this model predicts start/end probabilities for $N + 1$ positions. The start/end probabilities in the extra position are used for answerability prediction. Other approaches include adding auxiliary losses to the system [7] or encoding the no-answer option within the model [4].
2. *General accuracy improvements for answerable questions:* Borrowing some ideas from Transformer-XL architecture [5] to improve accuracy of QANet, especially for handling long-range dependencies could prove valuable.

Further, evaluating against domain-specific datasets such as CliCR [16] to further explore the different comprehension skills (co-reference, tracking, logical etc.) needed by the the system as described in [15] would be another future direction.

Acknowledgments

We are immensely grateful to Chris Chute for his help in interpreting the paper, providing guidance in debugging and in finding a critical bug in the persistence of model parameters to disk. We greatly appreciate Abi See for taking time outside the office hours and providing valuable direction. We would also like to thank Stephanie Dong for her help with the Azure.

References

[1] Christopher JC Burges. Towards the machine comprehension of text: An essay. *TechReport: MSR-TR-2013-125*, 2013.

[2] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[4] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *arXiv preprint arXiv:1710.10723*, 2017.

[5] Zihang Dai*, Zhilin Yang*, Yiming Yang, William W. Cohen, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Language modeling with longer-term dependency, 2019.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Minghao Hu, Yuxing Peng, Zhen Huang, Nan Yang, Ming Zhou, et al. Read+ verify: Machine reading comprehension with unanswerable questions. *arXiv preprint arXiv:1808.05759*, 2018.

[8] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[9] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[10] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[11] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[12] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[13] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[14] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[15] Saku Sugawara, Yusuke Kido, Hikaru Yokono, and Akiko Aizawa. Evaluation metrics for machine reading comprehension: Prerequisite skills and readability. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 806–817, 2017.

[16] Simon Šuster and Walter Daelemans. Clicr: a dataset of clinical case reports for machine reading comprehension. *arXiv preprint arXiv:1803.09720*, 2018.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[18] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

[19] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 189–198, 2017.

[20] Dirk Weissenborn, Georg Wiese, and Laura Seiffe. Making neural qa as simple as possible but not simpler. *arXiv preprint arXiv:1703.04816*, 2017.

[21] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.

6 Appendices

Attention mechanism

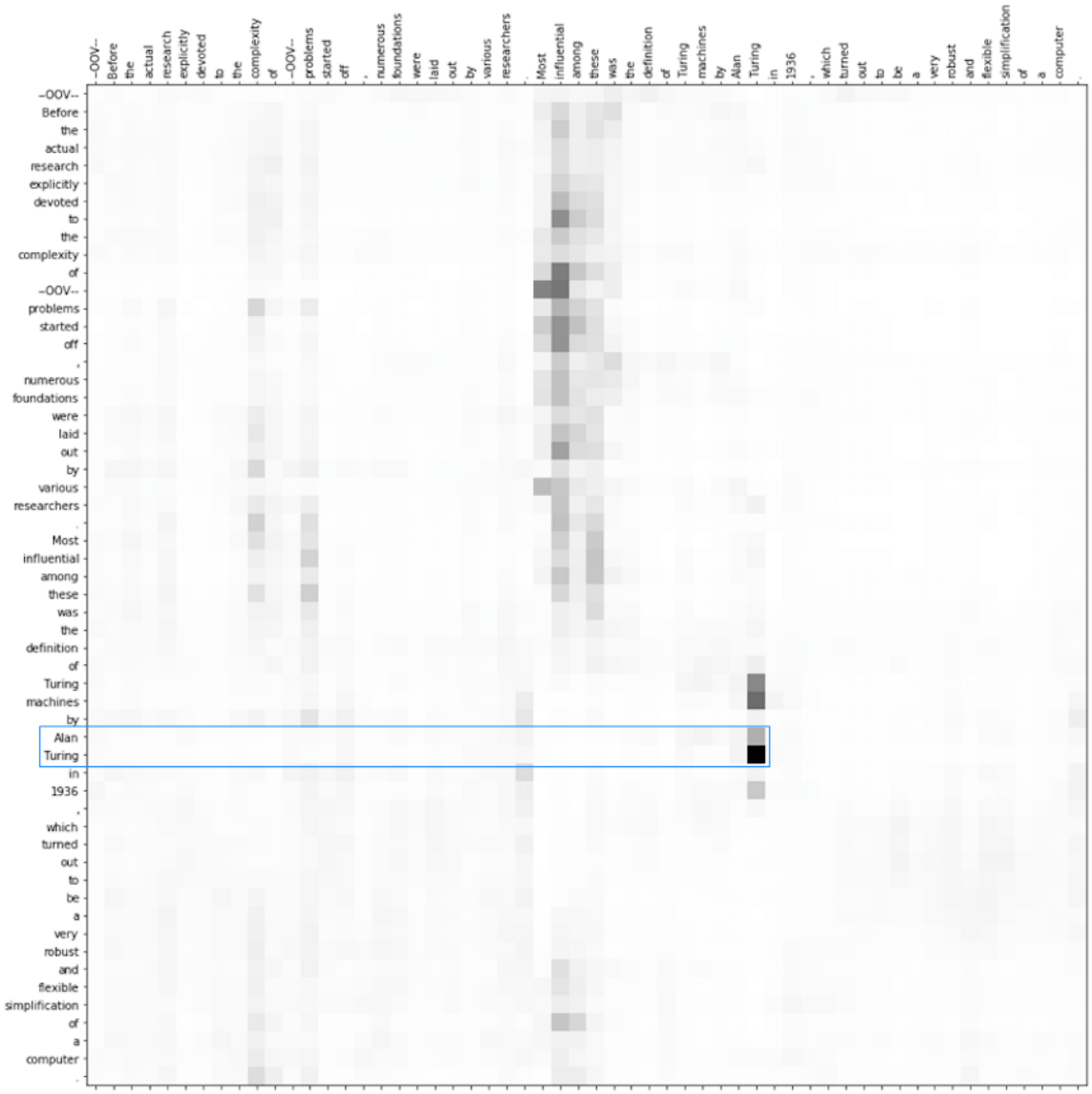


Figure 6: Attention mechanism at work
Question: Who was the most influential researcher among those grappling with the deficit of work surrounding the complexity posed by algorithmic problems?
Answer: Alan Turing