NeuralQA: A Usable Library for Question Answering (Contextual Query Expansion + BERT) on Large Datasets

Victor Dibia
Cloudera Fast Forward
vdibia@alumni.cmu.edu

Abstract

Existing tools for Question Answering (QA) have challenges that limit their use in practice. They can be complex to set up or integrate with existing infrastructure, do not offer configurable interactive interfaces, and do not cover the full set of subtasks that frequently comprise the QA pipeline (query expansion, retrieval, reading, and explanation/sensemaking). To help address these issues, we introduce NeuralQA - a usable library for QA on large datasets. NeuralQA integrates well with existing infrastructure (e.g., Elasticsearch instances and reader models trained with the HuggingFace Transformers api) and offers helpful defaults for QA subtasks. It introduces and implements contextual query expansion using a masked language model (MLM) as well as relevant snippets ($RelSnip$) - a method for condensing large documents into smaller passages that can be speedily processed by a document reader model. Finally, it offers a flexible user interface to support workflows for research explorations (e.g., visualization of gradient-based explanations to support qualitative inspection of model behaviour) and large scale search deoployment. Code and documentation for NeuralQA is available on Github.

1 Introduction

The capability of providing *exact* answers to queries framed as natural language questions can significantly improve the user experience in many real world applications. Rather than sifting through lists of retrieved documents, automatic QA (also known as reading comprehension) systems can surface an exact answer to a query, thus reducing the cognitive burden associated with the standard search task. This capability is applicable in extending conventional information retrieval systems (search engines) and also for emergent use cases,
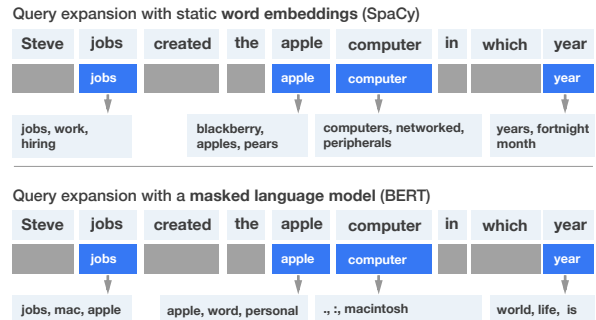

Figure 1: Examples of qualitative results from applying query expansion: (a) Query expansion using SpaCy (Honnibal and Montani, 2017) word embeddings to identify most similar words for each expansion candidate token. This approach yields terms with low relevance (e.g., terms related to work (jobs, hiring) and fruits (apple, blackberry, pears) are not relevant to the current query context) (b) Query expansion using a masked language model (BERT). This approach yields terms that are not contained within the original query (e.g., mac, macintosh, personal) but are, *in general*, relevant to the current query.

such as open domain conversational AI systems (Gao et al., 2018; Qu et al., 2019). For enterprises, QA systems that are both fast and precise can help unlock knowledge value in large unstructured document collections.

Conventional methods for open domain QA (Yang et al., 2015, 2019) follow a two-stage implementation - (i) a **retriever** that returns a subset of relevant documents. Retrieval is typically based on sparse vector space models such as BM25 (Robertson and Zaragoza, 2009) and TF-IDF (Chen et al., 2017); (ii) a machine reading comprehension model (**reader**) that identifies spans from each document that contain the answer. While sparse representations are fast to compute, they rely on exact keyword match, and suffer from the *vocabulary mismatch* problem - scenarios where the vocabulary used to express a query is different from the vo-
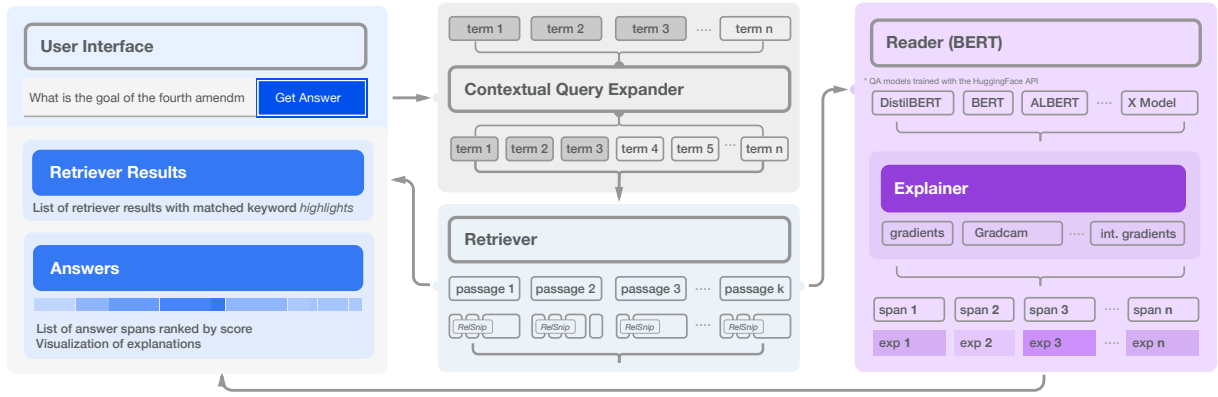
Figure 2: The NeuralQA Architecture is comprised of four primary modules. (a) User interface: enables user queries and visualizes results from the retriever and reader (b) Contextual Query Expander: offers options for generating query expansion terms using an MLM (c) Retriever: leverages the BM25 scoring algorithm in retrieving a list of candidate passages; it also optionally condenses lengthy documents to smaller passages via 3.1 *RelSnip*. (d) Document Reader: identifies answer spans within documents (where available) and provides explanations for each prediction

cabulary used to express the same concepts within the documents. To address these issues, recent studies have proposed neural ranking (Lee et al., 2018; Kratzwald et al., 2019) and retrieval methods (Karpukhin et al., 2020; Lee et al., 2019; Guu et al., 2020), which rely on dense representations.

However, while dense representations show significantly improved results, they introduce additional complexity and latency, which limits their practical application. For example, Guu et al. (2020) require a specialized MLM pretraining regime, as well as a supervised fine-tuning step, to obtain representations used in a retriever. Similarly Karpukhin et al. (2020) use dual encoders in learning a dense representation for queries and all documents in the target corpus. Each of these methods require additional infrastructure to compute dense representation vectors for all documents in the target corpus as well as implement efficient similarity search at run time. In addition, transformer-based architectures (Vaswani et al., 2017) used for dense representations are unable to process long sequences due to their self-attention operations which scale quadratically with sequence length. As a result, these models require that documents are indexed/stored in small paragraphs. For many use cases, meeting these requirements (rebuilding retriever indexes, training models to learn corpus specific representations, precomputing representations for all indexed documents) can be cost-intensive. These costs are hard to justify, given that simpler methods can yield comparable results (Lin, 2019). Furthermore, as reader models are

applied to domain-specific documents, they fail in counter-intuitive ways. It is thus valuable to offer visual interfaces that support debugging or sense-making of results (e.g., explanations for *why* a set of documents were retrieved or *why* an answer span was selected from a document). While several libraries exist to explain NLP models, they do not integrate interfaces that help users make sense of both the retriever and the reader tasks. Collectively, these limitations can hamper experimentation with QA systems and the integration of QA models into practitioner workflows.

Therefore, we introduce the NeuralQA library to help address these limitations. Our contributions are summarized as follows:

- An easy to use, end to end library for implementing QA systems. It integrates methods for query expansion, document retrieval (ElasticSearch[1]), and document reading (QA models trained using the HuggingFace Transformers API (Wolf et al., 2019)). It also offers an interactive user interface for sensemaking of results (retriever + reader). NeuralQA is open source and released under the MIT License.

- To address the vocabulary mismatch problem, NeuralQA introduces and implements a method for contextual query expansion (CQE), using a masked language model (MLM) fine-tuned on the target document corpus. Early qualitative results show CQE can surface relevant additional query terms

---

[1]ElasticSearch https://www.elastic.co

that help improve recall and require minimal changes for integration with existing retrieval infrastructure.

- In addition, we also implement *RelSnip*, a simple method for extracting relevant snippets from retrieved passages before feeding it into a document reader. This, in turn, reduces the latency required to chunk and read lengthy documents. Importantly, these options offer the opportunity to improve latency and recall, with no changes to existing retriever infrastructure.

Overall, this work complements a line of end to end applications that improve QA system deployment (Akkalyoncu Yilmaz et al., 2019; Yang et al., 2019) and provide visual interfaces for understanding machine learning models (Wallace et al., 2019; Strobelt et al., 2018; Madsen, 2019).

2 The Question Answering Pipeline

There are several subtasks that frequently comprise the QA pipeline and are implemented in NeuralQA.

2.1 Document Retrieval

The first step in the QA process focuses on retrieving a list of candidate passages which are subsequently processed by a reader. Conventional approaches to QA apply representations from sparse vector space models (e.g., BM25, TF-IDF) in identifying the most relevant document candidates. For example, Chen et al. (2017) introduce a end-to-end system combining TF-IDF retrieval with a multi-layer RNN for document reading. This is further improved upon by Yang et al. (2019), who utilize BM25 for retrieval with a modern BERT transformer reader. However, sparse representations are keyword dependent, and suffer from the *vocabulary mismatch* problem in information retrieval (IR); given a query Q and a relevant document D, a sparse retrieval method may fail to retrieve D if D uses a different vocabulary to refer to the same content in Q. Furthermore, given that QA queries are under-specified by definition (users are searching for unknown information), sparse representations may lack the contextual information needed to retrieve the most relevant documents. To address these issues, a set of related work has focused on methods for re-ranking retrieved documents to improve recall (Kratzwald et al., 2019; Wang et al., 2018).

More recently, there have been efforts to learn representations of queries and documents useful for retrieval. Lee et al. (2019) introduce an inverse cloze task for pretraining encoders used to create static embeddings, that are indexed and used for similarity retrieval during inference. Their work is further expanded by Guu et al. (2020) who introduce non-static representations that are learned simultaneous to reader fine-tuning. Finally, Karpukhin et al. (2020) use dual encoders for retrieval: one encoder that learns to map queries to a fixed dimension vector, and another that learns to map documents to a similar fixed-dimension vector (such that representations for similar query and documents are close).

2.2 Query Expansion

In addition to re-ranking and dense representation retrieval, query expansion methods have also been proposed to help address the vocabulary mismatch problem. They serve to identify additional, relevant query terms, using a variety of sources, such as the target corpus, external dictionaries (e.g., Word-Net), or historical queries. Existing research has explored how implicit information contained in queries can be leveraged in query expansion. For example, Lavrenko and Croft (2017); Lv and Zhai (2010) show how a relevance model (RM3) can be applied for query expansion and improve retrieval performance. More recently, (Lin, 2019) also show that the use of a well-tuned relevance model such as RM3 (Lavrenko and Croft, 2017; Abdul-Jaleel et al., 2004) results in performance at par with complex neural retrieval methods.

Word embeddings have been explored as a potential method for query expansion, as well. In their work, Kuzi et al. (2016) train a word2vec (Mikolov et al., 2013) CBOW model on their search corpora and use embeddings to identify expansion terms that are either semantically related to the query as a whole or to its terms. Their results suggest that a combination of word2vec embeddings and a relevance model (RM3) provide good results. However, while word embeddings trained on a target corpus are useful, they are static and do not take into consideration the context of the words in a specific query. Therefore, in this work, we propose an extension to this direction of thought and explore how contextual embeddings produced by an MLM, such as BERT (Devlin et al., 2018), can be applied in generating query expansion terms.

2.2.1 Document Reading

Recent advances in pretrained neural language models, such as BERT (Vaswani et al., 2017) and GPT (Radford et al., 2019), have enabled robust contextualized representation of natural language, which, in turn, have enabled significant performance increases on the QA task. Each QA model (reader) consists of a base representation and an output feedforward layer which produces 2 sets of scores: (i) scores for each input token that indicate the likelihood of an answer span starting at the token offset, and (ii) scores for each input token that indicate the likelihood of an answer span ending at the token offset.

3 NeuralQA System Architecture

In this section, we review the architecture for NeuralQA, as well as design decisions and supported workflows. The core modules for NeuralQA (Figure 2) include a user interface, retriever, expander, and reader. Each of these modules are implemented as extensible python classes (to facilitate code reuse and incremental development), and are exposed as REST API endpoints that can be either consumed by 3rd party applications or interacted with via the NeuralQA user interface.

3.1 Retriever

The retriever supports the execution of queries on an existing Elasticsearch instance, using the industry standard BM25 scoring algorithm.

3.1.1 Condensing Passages with $RelSnip$

In practice, open corpus documents can be of arbitrary length (sometimes including thousands of tokens) and are frequently indexed for retrieval *as is*. On the other hand, document reader models have limits on the maximum number of tokens they can process in a single pass (e.g., BERT-based models can process a maximum of 512 tokens). Thus, retrieving large documents can incur latency costs, as a reader will have to first split the document into manageable *chunks*, and then process each *chunk* individually. To address this issue, NeuralQA introduces $RelSnip$, a method for constructing smaller documents from lengthy documents. $RelSnip$ is implemented as follows: For each retrieved document, we apply a highlighter (Lucene Unified Highlighter), which breaks the document into fragments of size k_{frag} and uses the BM25 algorithm to score each fragment as if they were individual

documents in the corpus. Next, we concatenate the top n fragments as a new document, which is then processed by the reader. $RelSnip$ relies on the simplifying assumption that fragments with higher match scores contain more relevant information. As an illustrative example, $RelSnip$ can yield a document of 400 tokens (depending on k_{frag} and n) from a document containing 10,000 tokens. In practice, this can translate to 25x increase in speed.

3.2 Expander

3.2.1 Contextual Query Expansion

Contextual Query Expansion (CQE) relies on the assumption that a masked language model (MLM) that has been fine-tuned on the target document corpus contains implicit information on the target corpus that can be exploited in identifying relevant query expansion terms. Ideally, we want to expand tokens, such that additional tokens serve to increase recall, while adding minimal noise and without significantly altering the semantics of the original query. We implement CQE as follows:

First, identify a set of expansion candidate tokens. For each token t_i in the query t_{query}, we use the SpaCy (Honnibal and Montani, 2017) library to infer its part of speech tag $t_{i_{pos}}$ and apply a filter f_{rule} to select a subset as a candidate for expansion $t_{candidates}$. Next, we construct intermediate versions of the original query, in which each token in $t_{candidates}$ is masked, and an MLM (BERT) predicts the top n tokens that are contextually most likely to complete the query. These predicted tokens $t_{expansion}$ are then added to the original query as expansion terms.

To minimize the chance of introducing spurious terms that are unrelated to the original query, we find that two quality control measures are useful. First, we leverage confidence scores returned by the MLM and only accept expansion tokens above a certain threshold (e.g., $k_{thresh} = 0.5$) where k_{thresh} is a hyperparameter. Secondly, we find that a conservative filter in selecting token expansion candidates can mitigate the introduction of spurious terms. Our filter rule f_{rule} currently only expands tokens that are either nouns or adjectives $t_{i_{pos}} \in (noun, adj)$; tokens for other parts of speech are not expanded. Finally, the list of expansion terms are further cleaned by the removal of duplicate terms, punctuation, and stop words. Figure 1 shows a qualitative comparison of query expansion terms suggested by a static word em-

bedding and a masked language model for a given query.

3.3 Reader

The reader module implements an interface for predicting answer spans, given a query and context documents. Underneath, it loads any QA model trained using the HuggingFace Transformers API (Wolf et al., 2019). Documents that exceed the maximum token size for the reader are automatically split into chunks with a configurable stride and answer spans provided for each chunk. All answers are then sorted, based on an associated score (start and end token softmax probabilities). Finally, each reader model provides a method that generates gradient-based explanations (Vanilla Gradients (Simonyan et al., 2013; Erhan et al., 2009; Baehrens et al., 2010)).

3.3.1 User Interface

The NeuralQA user interface (Figure 3) seeks to aid the user in performing queries and in sense-making of underlying model behaviour. As a first step, we provide a visualization of retrieved document highlights that indicate what portions of the retrieved document contributed to their relevance ranking. Next, following work done in AllenNLP Interpret(Wallace et al., 2019), we implement gradient-based explanations that help the user understand what sections of the input (question and passage) were most relevant to the choice of answer span. We do not use attention weights, as they have have been shown to be unfaithful explanations of model behaviour (Jain and Wallace, 2019; Serrano and Smith, 2019) and not intuitive for end user sensemaking. We also implement a document and answer tagging scheme that indicates the source document from which an answer span is derived.

NeuralQA is scalable, as it is built on industry standard OSS tools that are designed for scale (ElasticSearch, HuggingFace Transformers API). We have tested deployments of NeuralQA on docker containers running on CPU machine clusters which rely on ElasticSearch clusters. The UI is responsive and optimized to work on desktop as well as mobile devices.

3.4 Configuration and Workflow

NeuralQA implements a command line interface for instantiating the library, and a declarative approach for specifying the parameters for each module. At run time, users can provide a command line argument specifying the location of a configuration YAML file.[2] If no configuration file is found in the provided location and in the current folder, NeuralQA will create a default configuration file that can be subsequently modified. As an illustrative example, users can configure properties of the user interface (views to show or hide, title and description of the page, etc.), retriever properties (a list of supported retriever indices), and reader properties (a list of supported models that are loaded into memory on application startup).

3.4.1 User Personas

NeuralQA is designed to support use cases and personas at various levels of complexities. We discuss two specific personas briefly below.

Data Scientists: Janice, a data scientist, has extensive experience applying a collection of machine learning models to financial data. Recently, she has started a new project, in which the deliverable includes a QA model that is skillful at answering factoid questions on financial data. As part of this work, Janice has successfully fine-tuned a set of transformer models on the QA task, but would like to better understand how the model behaves. More importantly, she would like to enable visual interaction with the model for her broader team. To achieve this, Janice hosts NeuralQA on an internal server accessible to her team. Via a configuration file, she can specify a set of trained models, as well as enable user selection of reader/retriever parameters. This workflow also extends to other user types such as hobbyists, entry level data scientists or researchers who want an interface for qualitative inspection of custom reader models on custom documents indices.

Engineering Teams: Candice manages the internal knowledge base service for her startup. They have an internal ElasticSearch instance for search, but would like to provide additional value via QA capabilities. To achieve this, Candice provisions a set of docker containers running instances for NeuralQA and then modifies the frontend of their current search application to make requests to the NeuralQA REST API end point and serve answer spans.

3.5 Related Work

QA systems that integrate deep learning models remain an active area of research and practice. For

---

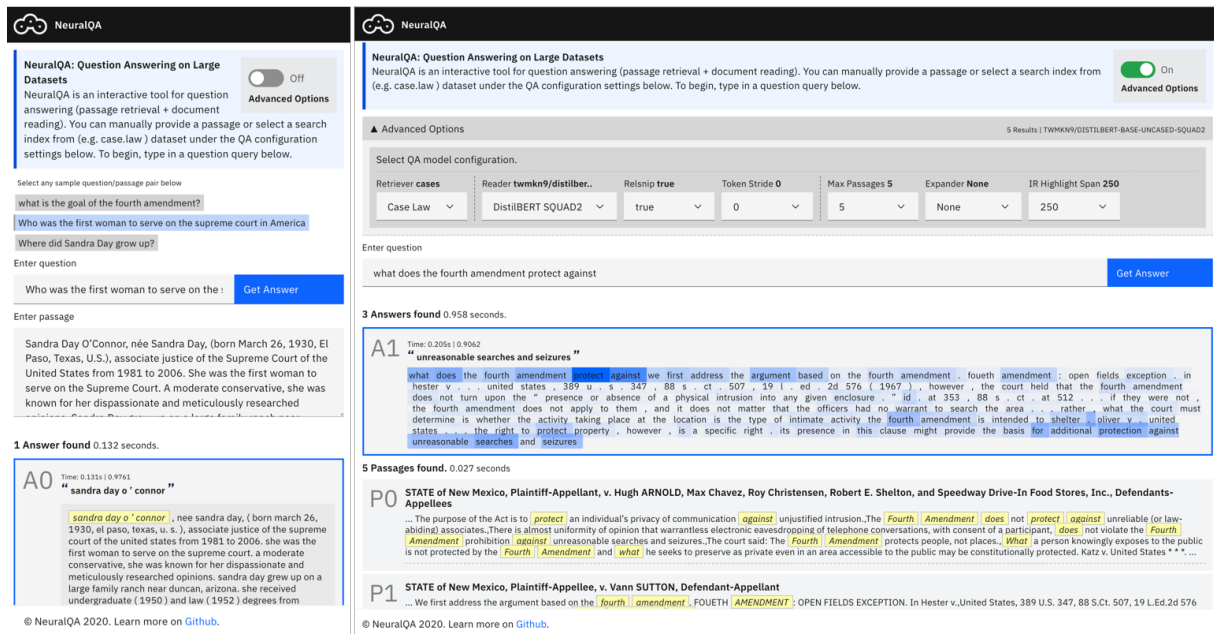[2] A sample configuration file can be found on Github.

Figure 3: The NeuralQA user interface is optimized for web and mobile. a.) Basic view (mobile) for closed domain QA, i.e., the user provides a question *and* passage. Any configured defaults are also displayed. b.) Advanced options view (desktop mode) for open domain QA. The user can configure the retriever (e.g., # of returned documents, toggle $RelSnip$, highlight size k_{frag}), set expander and reader parameters (BERT reader model, token stride size)). View also shows a list of returned documents (P0,P1) with highlights that match the query; a list of answers (A0) with a gradient-based explanation of which tokens impact the selected answer span.

example, Allennlp Interpret (Wallace et al., 2019) provide a demonstration interface and sample code for interpreting a set of Allennlp models across multiple tasks. Similarly, Chakravarti et al. (2019) provide a gRPC-based orchestration flow for QA. However, while these projects provide a graphical user interface (GUI), their installation process is complex and requires specialized code to adapt them to existing infrastructure, such as retriever instances. Several open source projects also offer a programmatic interface for inference (see Hugging-Face pipelines[3]), as well as joint retrieval paired with reading (see Deepset Haystack[4]).

NeuralQA makes progress along these lines, by providing an extensible code base, a low-code declarative configuration interface, and a visual interface for sensemaking of results. It supports a local development workflow (via the pip package manager [5]) and scaled containerization (we provide a Dockerfile). We believe this ease of use can serve to remove barriers to experimentation for

researchers, and accelerate the deployment of QA interfaces for experienced teams.

4 Conclusion

In this paper, we presented NeuralQA - a usable library for question answering on large datasets. NeuralQA is useful for developers interested in qualitatively exploring QA models for their custom datasets, as well as for enterprise teams seeking a flexible QA interface/API for their customers. NeuralQA is under active development, and roadmap features include support for a Solr retriever, and implementation of additional query expansion methods such as RM3 (Lavrenko and Croft, 2017). Future work will also explore empirical evaluations of our contextual query expansion implementation to better understand its strengths and limitations.

5 Acknowledgments

---

[3]HugginFace pipelines https://huggingface.co/transformers/main_classes/pipelines.html

[4]Deepset Haystack https://github.com/deepset-ai/haystack/

[5]NeuralQA on PyPI https://pypi.org/project/neuralqa/

References

Nasreen Abdul-Jaleel, James Allan, W Bruce Croft, Fernando Diaz, Leah Larkey, Xiaoyan Li, Mark D Smucker, and Courtney Wade. 2004. Umass at trec 2004: Novelty and hard. *Computer Science Department Faculty Publication Series*, page 189.

Zeynep Akkalyoncu Yilmaz, Shengjin Wang, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Applying BERT to document retrieval with birch. In *Empirical Methods in Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 19–24, Hong Kong, China. Association for Computational Linguistics.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. 2010. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831.

Rishav Chakravarti, Cezar Pendus, Andrzej Sakrajda, Anthony Ferritto, Lin Pan, Michael Glass, Vittorio Castelli, J William Murdock, Radu Florian, Salim Roukos, and Avi Sil. 2019. CFO: A framework for building production NLP systems. In *EMNLP-IJCNLP 2019: System Demonstrations*, pages 31–36, Hong Kong, China. Association for Computational Linguistics.

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

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

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2009. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1.

Jianfeng Gao, Michel Galley, and Lihong Li. 2018. Neural approaches to conversational ai. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1371–1374.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Sarthak Jain and Byron C Wallace. 2019. Attention is not explanation. *NAACL 2019*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Bernhard Kratzwald, Anna Eigenmann, and Stefan Feuerriegel. 2019. RankQA: Neural question answering with answer re-ranking. In *ACL*, pages 6076–6085, Florence, Italy. Association for Computational Linguistics.

Saar Kuzi, Anna Shtok, and Oren Kurland. 2016. Query expansion using word embeddings. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pages 1929–1932.

Victor Lavrenko and W Bruce Croft. 2017. Relevance-based language models. In *ACM SIGIR Forum*, volume 51, pages 260–267. ACM New York, NY, USA.

Jinhyuk Lee, Seongjun Yun, Hyunjae Kim, Miyoung Ko, and Jaewoo Kang. 2018. Ranking paragraphs for improving answer recall in open-domain question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 565–569, Brussels, Belgium. Association for Computational Linguistics.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*.

Jimmy Lin. 2019. The neural hype and comparisons against weak baselines. In *ACM SIGIR Forum*, volume 52, pages 40–51. ACM New York, NY, USA.

Yuanhua Lv and ChengXiang Zhai. 2010. Positional relevance model for pseudo-relevance feedback. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 579–586.

Andreas Madsen. 2019. Visualizing memorization in rnns. *Distill*. Https://distill.pub/2019/memorization-in-rnns.

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.

Chen Qu, Liu Yang, Minghui Qiu, W Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. 2019. Bert with history answer embedding for conversational question answering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1133–1136.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc.

Sofia Serrano and Noah A. Smith. 2019. Is attention interpretable? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2931–2951, Florence, Italy. Association for Computational Linguistics.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. M. Rush. 2018. Seq2Seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models. *ArXiv e-prints*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. 2019. AllenNLP Interpret: A framework for explaining predictions of NLP models. In *Empirical Methods in Natural Language Processing*.

Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerry Tesauro, Bowen Zhou, and Jing Jiang. 2018. R 3: Reinforced ranker-reader for open-domain question answering. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718*.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 2013–2018.