Information Sciences 480 (2019) 339–353

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An anatomy for neural search engines

Thiago Akio Nakamura[a,b,*], Pedro H. Calais[b], Davi de Castro Reis[b], André Paim Lemos[a]

[a] Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais, Av. Antônio Carlos, 6627, Belo Horizonte, Minas Gerais 31270-901, Brazil
[b] WorldSense, Av. Afonso Pena, 4000, Belo Horizonte, Minas Gerais 30130-008, Brazil

ARTICLE INFO

Article history:
Received 24 July 2018
Revised 29 November 2018
Accepted 21 December 2018
Available online 22 December 2018

MSC:
68P20
68T50
68U01

Keywords:
Information retrieval
Machine learning
Deep learning
Neural IR
Search engine

ABSTRACT

In this work, we explore the application of modern deep learning techniques to build a neural model centric search engine. We conduct an in-depth discussion under several quantitative and qualitative criteria, comparing the trade-offs of adopting the proposed neural architecture against the successful and mature traditional information retrieval techniques. We show that a full neural architecture, which employs neural models both in the retrieval and ranking phases, offers good scalability, predictability and evolution properties, and discuss under which conditions one can achieve state-of-the-art results. We conclude that deep learning centric systems still require significant more effort to implement and deploy and demand more computational resources, but this work, together with several others in the research community, sheds a light into that path.

1. Introduction

For a little over a decade, the *deep learning* revolution has been dramatically improving how computers perform on a variety of tasks, very often surpassing human capabilities. The tasks range from processing image, video and audio, natural language understanding with text and speech, to game playing and conversational agents, attracting not only the research community but being applied to solve real world problems. The main breakthrough was the ability to train the long-standing artificial neural networks, but now with many more hidden layers. Those extra hidden layers build increasingly higher levels of abstraction of the data at hand, and are able to generate features usually more significant and robust than those handcrafted by scientists and engineers. On the other hand, these new learning algorithms are very data hungry, and the increasingly global connectivity in the digital world, also boosted in the last decade, enabled us to feed them with a plethora of data. Lastly, processing and training from so much data was made possible by both more efficient algorithms and an exponential increase of computing power.

* Corresponding author at: Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais, Av. Antônio Carlos, 6627, Belo Horizonte, Minas Gerais 31270-901, Brazil.
E-mail addresses: akionakamura@ufmg.br (T.A. Nakamura), pedro@worldsense.com (P.H. Calais), davi@worldsense.com (D.d.C. Reis), andrepl@cpdee.ufmg.br (A.P. Lemos).

https://doi.org/10.1016/j.ins.2018.12.041
0020-0255/

Deep neural networks and algorithms really shine when dealing with unstructured data. Traditional machine learning requires explicit feature engineering, and extracting features from unstructured data is a tough task. Alternatively, due to its hierarchical structure and potentially large parameter space, deep neural networks are able to build abstractions and find patterns from the raw data.

The deep neural networks found its glory in image processing and computer vision, where "tricks" like the convolutional networks naturally encode spatial information and lead to reasonably interpretable and visualizable models [36]. Although the neural models for natural language processing (NLP) lack some of the same intuition, it has also been a big research focus. Important advances in the neural NLP models include recurrent neural networks, which encode time-dependency, and bidirectional inputs [50], attention mechanisms [44] and, borrowing from the computer vision community, convolution networks [42]. Furthermore, since these models can fall short in intuition and principled design decisions, there has also been work in the opposite direction, aiming at interpreting the trained network and its results [23].

In this work, we are interested on how deep learning can improve text based information retrieval (IR). Several works in the literature show how the feature composability of deep neural networks are capable of capturing "the deeper semantic meaning of words" [2], an important feature for a search engine, where it has to deal with semantic processing, word sense disambiguation and similarity encoding. In the IR context, deep neural networks learn to encode the information need of a query and the information content of a document, as well as how to match them, and depending on the chosen architecture, the encoding process might capture different semantic information [35]. Hence, we propose a study to look beyond the neural IR models themselves, and their individual ranking quality metric, and investigate a complete neural model based search engine.

More specifically, we build a deep learning centric search engine with the goal of exploring and understanding the benefits and trade-offs of neural based IR over the traditional architecture and algorithms. We fully cover it in Section 3, but, in summary, our proposal is to leverage the power of neural models *both* in the retrieval and ranking phases of a search engine. In the **retrieval phase**, we leverage intermediate representations of **representation-focused** neural models, which allows for an efficient and cost-predicable document retrieval. Next, a final **ranking phase** is provided by employing an **interaction-focused** neural model, which is capable of learning more complex relations between the query and the document, while also overcoming the exact-match limitations of classical techniques

To explore this space, we have chosen a classical information retrieval task, namely ad-hoc search, and implemented fully operational search engines both with classical and neural IR architectures. Having both implementations, we go over aspects of IR systems beyond the ranking quality metrics, broadening the discussion over important practical engineering aspects, usually neglected in the academic literature. As an anecdotal motivation, Netflix famously proposed a 1 million dollar challenge to improve its recommendation system. It ended up choosing not to deploy the winning model in production due to engineering costs. Our final goal is to understand the factors that may lead a team to implement and use one IR architecture over the others. Therefore, we compare our proposed neural architecture with the traditional one, plus some mixed variations, and offer an in-depth discussion over the following dimensions:

Ranking metrics. In the end, nothing else matters if the quality is bad. Is the neural anatomy at least as good as the traditional approaches? We are a going to look at mean average precision (MAP), normalized discounted cumulative gain (nDCG), mean reciprocal rank (MRR) and precision/recall.

Query type. The user can have different goals when web searching. We want to understand if the query type (namely, precision and recall-oriented queries) influences the effectiveness of each architecture.

Latency mean and variance. Low latency is a big deal, but the latency variance and load predictability is also important, especially for large scale systems.

Data need. Training neural IR models usually requires a lot of labeled data, which can be expensive to obtain. Is weak supervision, as a means of acquiring labeled data, enough to bootstrap a good neural model [13]?

Computatio nal cost and resources. What resources each architecture requires? Is the computation cost predictable and controlled?

Engineering costs and maintenance. How hard it is to implement it? What are the existing commercial and/or open source solutions? Furthemore, ideally the model and overall system should be easy to evolve, add new features, or even re-purpose for different IR tasks.

The remainder of this work is organized as follows: Section 2 offers a literature review on classical and neural IR. Section 3 describes our proposed neural anatomy and which properties have driven its design, Section 4 presents our experimental setup and results and Section 5 wraps up the discussion with a more qualitative analysis and shows motivations of future work. Finally, Section 6 concludes this work, summarizing our contributions and learnings.

2. Literature review

The most fundamental IR task is the *ad-hoc* search task, in which the system must respond with a ranked list of documents that is relevant for the information need of the user, usually communicated to the system as an one-off query, e.g., "hotels in New York". One of the ways to rank a set of documents is to compute a relevance score for each document d given a query q. Hence, the task boils down to a *representation* and a *matching* problem, i.e., how do you represent a query and a document in a way that a posterior matching function returns the highest values for the most relevant documents for

a given query q? That relevance scoring process can be generally expressed by:

$$rel(q, d) = f(\Phi_q(q), \Phi_d(d)) \tag{1}$$

where Φ_q and Φ_d are representation mapping functions, i.e., they compute the representation of the query and document, respectively, and f is the matching function based on the interaction between the query and document representations.

We know from experience that the ranking function for any commercial *ad-hoc* search system uses a combination of query-independent signals and query-dependent signals, i.e., it might be something similar to $rel(d, q) = \Psi(d) \times f(\Phi_q(q), \Phi_d(d))$. The query-independent signals $\Psi(d)$ are built out of authority/popularity algorithms, like PageRank [33] or spam detection score [11]. This work leaves these signals aside, as we understand they can be explored orthogonally to the computation of the query-dependent signals.

Next we review some of the classical techniques for text-based IR and the latest neural IR approaches.

2.1. Classical IR

One of the first IR solutions was proposed in the1950's, the boolean model: the document and query are represented simply by a boolean vector indicating which words of the vocabulary they contain. The matching function f is simply a boolean operation indicating whether the words in the query are present in the document. It may seem too simple, but that is the basis of the widely and successfully adopted TF-IDF based ranking, which can be seen as an extended boolean model using vector space model to give weights to the words according to its collection-level frequency (IDF) and document-level frequency (TF), proposed in the1970's [22].

Hence, in traditional IR techniques, the functions Φ_q and Φ_d compute a vector representation as TF-IDF (or similar metrics) weights of their terms, and the matching function f of the pair is computed by a closed-form equation, e.g., BM25 [40], considering the overlapping terms. Fancier representations might consider longer sequences of words, instead of individual words, distance between terms, synonyms and non-compositional compounds [18]. Furthermore, oftentimes a document might be comprised of several components, like title, main body, meta tags and anchor texts, and perform feature engineering from these sources, plus the decision of how to weight them in the final f matching function, is not trivial [39].

Another drawback of the classical techniques is that it assumes the term-based retrieval fetches a reasonably complete list of candidate documents. However, because of a well known vocabulary gap, this assumption is not fully accurate [7]. A document can be relevant to a query even without exact matching terms, due to the use of plural, conjugation, synonyms, or semantic relation, to cite a few. Document and query expansions can help solve this problem, and they are full research fields by themselves. To put it in other words, using exact term-matching retrieval leads to a loss of recall right off the bat, since a document containing only "artificial intelligence" will not be retrieved for the query "intelligent machines", for example.

2.2. Neural IR

Handcrafting ranking formulas like BM25 is not an easy task; we are most of times tuning it based on our intuitions and extensive experimentation. Taking advantage of the amount of potential training data gathered from the surge of the Internet, it has become possible to leverage classical machine learning methods to build ranking models. We call those methods that learn how to combine predefined features for ranking by means of discriminative learning as "learning-to-rank" (L2R) methods [25].

While classical L2R methods are capable of learning complex relations between the features, they still rely on a fixed set of features carefully engineered and extracted from the queries and documents. This is where deep neural models can shine. Their deep hierarchical structure build increasing higher levels of abstractions starting from raw data, and generate features usually more significant and robust than those handcrafted by scientists and engineers. Deep learning models designed for information retrieval are typically named *Neural IR*. We recommend the reader a reviewing work for an extensive look at the current neural IR landscape [29].

Besides the capacity of extracting important features and learning complex functions, neural IR models overcome another classical IR limitation: they can support soft matching very naturally, since most text based neural models are fed with word embeddings, which have been shown to capture word relations in several dimensions [6]. This soft matching broadens the query and document encoding, which can lead to a higher recall by weighting in related terms in the scoring function.

That said, although neural models can learn what are the good features for the given task, as well as learn a more complex function of how these features interact, we should not over-romanticize it. Whilst it is true that feature engineering has been simplified in many cases, the model complexity has grown accordingly. Oftentimes, the neural network structure is as specific to a problem as features used to be. Every time a human takes a structural decision that could not be learned, we are essentially hard-coding a feature. Architecture engineering is the new feature engineering.

Next, we describe the main modeling strategies and review import contributions in the Neural IR literature.

2.2.1. Representation-focused models

As aforementioned, the neural network structure is a fundamental part of the process of modeling the problem and, under the IR lenses, it is typically defined by how you model each component of the relevance estimation function. In short, the neural IR literature will vary on which components of Eq. (1) is learned through neural models: Φ, f or both [17,29].

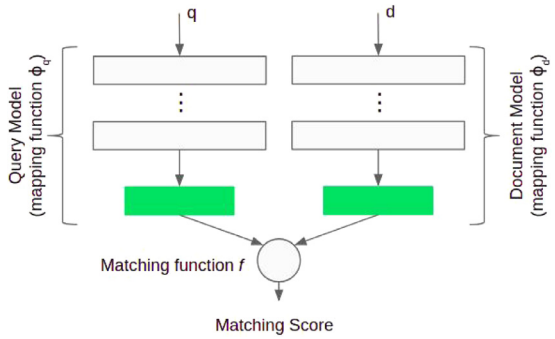

Fig. 1. Representation-focused neural models have separate components for queries and documents, which compute their dense representation. These components can be used separately, and their dense representations can be cached and indexed for fast lookup and nearest neighbours computation. The matching function is usually a simple distance metric function.

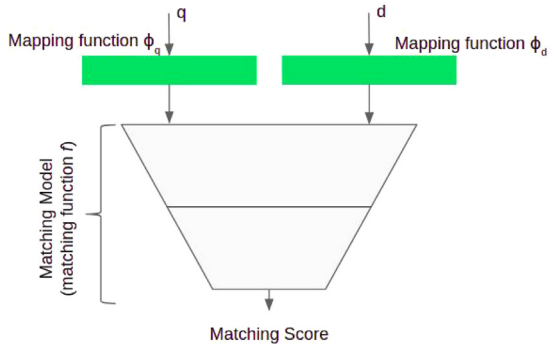

Fig. 2. Interaction-focused neural models have simple representation mapping functions, from which a neural model computes detailed interaction and matching features between the query and the document.

Representation-focused or **distributed** models, as depicted in Fig. 1, can be seen as composed by two separate parts, a *query model,* Φ_q, and a *document model,* Φ_d. The matching function f can be as simple as a vector distance metric, like cosine distance (DSSM, C-DSSM, CLSM, LSTM-RNN [19,34,42]), or yet another neural network (NRM-F [49]).

It is important to note that, even though the query and document models are trained together, they can be used separately. They are capable of independently computing queries and document representations, which allows us to compute and cache those representations offline, leading to a more efficient use at inference time.

2.2.2. Interaction-focused models

Interaction-focused or **local** models, shown in Fig. 2, use simple representation mapping functions, Φ_q and Φ_d, typically just the token indices or their word embeddings, in order to allow detailed interactions between the query and the document. Then a complex matching function f, the neural network, learns to match them based on those detailed interactions. For example, DeepRank [37] has an intricate scheme to detect local matching snippets and compute an aggregated relevance score. The MatchPyramid [36] and K-NRM [47] models build an interaction matrix from the normalized dot product of the word embeddings of the query and document, and apply a deep neural network over it, e.g., a convolutional neural network. The DRMM [17] offers a simpler yet effective structure, where it builds a matching score histogram for each term in the query, passes it through a fully connected network and a gating network weights the importance of each query term in the final score. Finally, the DUET model proposes a two part model, one interaction-focused and another representation-focused, used simultaneously by summing their score [30].

The two types of deep neural models described above lead to different behavior under the IR task. The representation-focused models usually encodes well document and query topicality, whilst interaction-focused models are better at learning query terms density information [17,35].

Having that difference in mind, the neural search engine described next aims at leveraging each type of neural network to be used in the proper stage of a search.

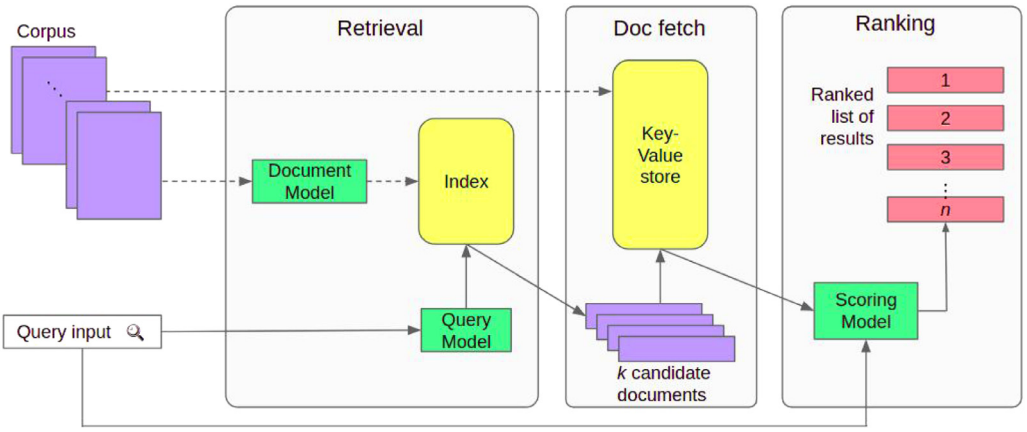

Fig. 3. Anatomy of a general search engine. Dashed lines indicate offline computations, whilst continuous line indicate the online flow at search time.

3. Neural anatomy

In this section, having reviewed the basic neural IR models, we discuss more about their properties and how they fit in a search engine architecture. Furthermore, we present how to instantiate each of the components that constitutes our neural search engine.

3.1. Motivation and contributions

A large scale search engine operates over a large corpus, usually in the order of millions, or even billions of documents. At this level, careful optimizations are needed in order to work efficiently. More specifically, when given a query, the scoring function cannot be ran for every document in the corpus, even if that function is relatively cheap. A solution often used is the so-called multi-stage retrieval, where we start with a highly scalable method to retrieve a set of candidate documents and, at each stage, we re-rank and reduce the current set, refining the results. The ranking function at each consecutive stage typically gets more sophisticated and, usually, more expensive.

A typical large scale search engine anatomy, with a single retrieval stage, follows the scheme illustrated in Fig. 3. Each of the three phases can be implemented with different data structures and associated algorithms, and the popular choices have been highly optimized over the years [46]. The flow at search time goes as follows:

1. **Query input**: the query is parsed into an internal representation used to consult the index.
2. **Retrieval phase**: given the query representation, the index returns a set of candidate document IDs. An inverted index is perhaps the most popular data structure used in the retrieval step. In an inverted index, for each vocabulary term in the corpus, we keep the list of document IDs that contain that term. At retrieval time, only those document IDs which have at least one of the query terms are retrieved.
3. **Document fetch phase**: given the candidate document IDs, we collect the document contents and features needed to score them, according to the scoring model, e.g., title, body, keywords, PageRank [33], etc. This phase is typically comprised of an in memory key-value store for fast lookup.
4. **Ranking phase**: finally, the scoring function is ran only for the returned set of candidate documents, which is several orders of magnitude smaller than the full corpus, and the top n ranked results are returned to the user. This is where the TF-IDF and BM25 ranking functions are successfully applied for a long time [22,40], and also where learn-to-rank models are typically used [25].

Most of the neural IR work in the literature, reviewed in Section 2.2, focuses on the ranking phase only, i.e., building a neural model that performs well at ranking a given set of documents. In parallel, there has been proposals to build an efficient retrieval phase with the use of neural models and nearest-neighbors algorithms, but they have been used out of the scope of *ad-hoc* search, and without a follow up re-ranking [7].

To summarize, in our view, the main neural IR works in the literature are limited due to one or more design decisions, which are listed next:

- The retrieval phase was ran by a classical algorithm with a posterior neural re-ranking [13,17];
- There was no retrieval phase, the model was tested directly against a labeled set of documents, which is not realistic in a real-world application [19,30,34,37,42,47,49].
- It considers only the document title when scoring, whereas ideally it should leverage the longer and richer document body, or multiple fields [19,34,42,47];

Our proposed full neural architecture aims to address all aforementioned limitations. We employed two different neural networks, one for the retrieval phase (DAN) and another for the ranking phase (DRMM), and they are described in more detail in Sections 4.2 and 4.3, respectively. However, it is important to note that developing and improving the neural models per-se to get the best possible ranking quality is not the main focus of this work. We are interested on how to build a fully neural search engine, and how it performs beyond the hard ranking quality metrics. Therefore, the authors opted for implementing what they judged to be the most consolidated, accessible and easily reproducible neural models in the literature, which are also a reference for several recent works [28,37]. The authors do acknowledge those new models show important improvements, but they are yet to be consolidated, and developing more elaborate models is a work orthogonal to this, which should eventually lead to better ranking quality metrics.

Additionally to the potentially better ranking capabilities of neural models, a full neural approach brings a lot of regularities to the computational cost of retrieval and ranking. All documents and queries are represented by vectors of fixed size and the number of documents retrieved can be controlled. These properties have several practical implications. First, they allow for predictable resource usage and predictable latency, which are very important concerns for the industry [9,21]. Secondly, they also result in arguably better software portability and composability. Finally, a deep learning centric approach gives clear direction on how to bring most of the new features into the system, without a need to revisit its engineering concerns. In this work, we assume all those as desirable properties even at the cost of other aspects, such as interpretability.

Next, we concretely describe how to instantiate each component of the neural search engine anatomy, following Fig. 3.

3.2. Retrieval phase

As aforementioned, in the neural IR literature, it is common to use the proposed models only to re-rank a given document set retrieved by a traditional technique. There are two major drawbacks of implementing the ranking phase with a neural model and keeping a classical retrieval phase. First, the retrieval is based only on exact term matching, which will potentially lead to a loss of recall right of the bat, as discussed in Section 2.1. Secondly, the classical inverted index has an inherent unpredictability of not knowing in advance how many documents a given term will retrieve. If a query term is common, it will be present in many documents of the corpus, resulting in a large retrieved set, which also increases cost of the following phases.

These limitations motivated us to also use a neural model for the retrieval phase. Our proposal is that with a **representation-focused** neural model, one can achieve the scalability needed in the retrieval phase by leveraging the intermediate representations. With the trained model, the document dense representations can be kept in memory and, given the query representation at retrieval time, a simple nearest-neighbors algorithm can be used to retrieve the candidate set.

As depicted in Fig. 3, the retrieval phase has three important components, which are concretely described next.

3.2.1. Query and document models

We need to train a **representation-focused** model to achieve an efficient retrieval phase, since we can leverage its intermediate representations. As depicted in Fig. 1, this model has two components, a **query model**, to learn a query representation, and a **document model**, to learn a document representation.

Note how each component can have inputs comprised of several fields. For example, the document can be composed by the title, the body and anchor texts. It is important that, in the end, both the final query and document representations share the same dimensionality, so we can directly compare their distances.

The training goal of this model should be to minimize the distance between the query and relevant document representations, and cosine distance is the most typically used.

3.2.2. Index

With the trained document model, we can compute the document dense representation of all the documents in the corpus and have them indexed for a fast lookup. On retrieval time, the query representation, computed by the query model, is used to probe this index and retrieve the nearest documents.

Ideally, when looking for the nearest neighbors, we should compute the distance for all documents in the corpus, but that can be prohibitively expensive for a big corpus. Therefore, we opt to use an approximate nearest neighbors library, specifically the open source Annoy library [5] (there are several alternatives [3,26]). Annoy gives us the flexibility of tuning the index for latency, memory use and accuracy with two hyper-parameters, namely the number of trees of the index and the number of nodes searched during lookup.

For a given set of hyper-parameters, having a fixed-size representation for documents results in a demand of memory that only depends on the number of elements in the corpus, and probing the index should have a constant and predictable cost. This design allows us to use readily available optimizations for nearest-neighbors algorithms, like product quantization, instead of having to tune the myriad of problem dependent techniques and manage the gigabytes of compressing and indexing text [46].

3.3. Document fetch phase

The Annoy index returns a set of document IDs, and we need to fetch the content of those documents in order to score them for the given query. We chose to use Redis, an in-memory key-value store popularly used by the industry [27].

The values stored in Redis are already processed and ready to be used by the scoring model, that way we save that computation from being ran at search time. Since the ranking model, as described next, follows the *interaction-focused* modeling, the indexed document content here is typically very simple, for example, a list of token indices in the title/body of the document.

3.4. Ranking phase

Several works in the literature have shown **interaction-focused** models work well for Neural IR, often beating traditional ranking algorithms [17,30,36]. The fine grained interaction between the query and the document allows the model to learn complex relations between them, as well as capture more information from the soft term-matching capabilities of word embeddings. These models are, however, expensive to run, which make them fit to be used only in the *ranking phase* of a search engine, dealing only with the candidate documents set, instead of the entire corpus.

In conclusion, besides the hyper-parameters of the neural network models themselves, the whole system has a few other tuning knobs, namely the accuracy/latency/memory trade-off of the approximate nearest neighbors algorithm and the number k of candidate documents to retrieve for scoring. We believe these hyper-parameters are largely general, and a good setup can be easily adapted to be used for several different IR problems, without much problem-specific fine-tuning.

4. Experiments

We have chosen a classical ad-hoc search task to test against our proposed neural architecture, and we present the experimental setup and results in this section.

4.1. Dataset and setup

We used the TREC Web Track datasets from 2009 to 2012 in our experiments. The TREC Web Track series aims at creating a labeled dataset to explore and evaluate large-scale web retrieval technologies. With 50 queries in each year, for a total of 200 queries, there are on average 369 human-labeled relevance judgments per query. Each query-document pair is judged on a three-point scale as being “relevant”, “highly relevant” or “not relevant”. These judged documents come from the ClueWeb09 Category B collection, a large scale heterogeneous collection with 50 million English web documents, which will serve as our document corpus.

The TREC Web Track datasets provide too few query-document examples to support the full training of a deep neural network, and therefore they were only used as the test dataset. To train the neural networks, we used a weak supervision strategy [13]: by collecting the query-document relevance score from a well established IR technique, the pseudo-labeler, we train the neural model to learn-to-rank according to these weak-labels. The queries used for the weak supervised training are drawn from the AOL query logs [38]. This collection is comprised of real user submissions to AOL search engine from March 1st, 2006 to May 31, 2006. We filter out URL-like patterns and numbers-only queries. Furthermore, we only keep those that occurred at least twice in the collection, filtering out a lot of noisy queries. That results in 3.3M distinct queries, from which we randomly select 80% for training and 20% for validation.

The pseudo-labeler used was the Elasticsearch implementation of BM25, with $b = 0.4$ and $k1 = 0.9$, ICU analyzer and English stop words filter, from AWS Elasticsearch Service, version 5.1. It was used the *multi match query* with cross-fields scoring over the document title and body, directly extracted from the raw HTML documents. The choice of using Elasticsearch was motivated by its high commercial availability, scalability and open-sourceness [16]. This is also the classical IR technique used as the baseline for our experiments.

For each query in the training/validation set, we collected the top 16 documents scored by the pseudo-labeler. Furthermore, aiming to have a more diverse and distinguishable set of documents, we also randomly selected other 16 documents from the top 1000 documents scored by the pseudo-labeler, resulting in 32 pseudo-relevance scores per query. We have filtered out the spammy documents from the collection using the Waterloo Fusion spam scorer with the threshold of 60% [11]. Note that all neural models are trained and validated using the same dataset, as described here.

The data was processed and built using Spark’s Transformers Pipeline [48]. The Web ARChive (WARC) files containing the web documents were parsed using `jwat-warc` library, and the HTML files were parsed using the `jsoup` parser, from which the full raw text and title are extracted. To clean up the text, it was used the same strategy adopted by the pre-trained word-embeddings, FastText, where all words were lower-cased and the special characters `([.\!?,'/()])` removed. The output of the pre-processing pipeline are TF-Records, each containing a document, represented by a cleaned title and body strings and a globally unique ID. The queries are cleaned in the same manner.

The TensorFlow Dataset API was used to feed the models during training, which were defined using TensorFlow and Keras [1]. The text tokenization and token-to-index mapping were built within the TensorFlow graph, so that the final model is able to work directly with string inputs. The final trained models were served with TensorFlow-Serving during the experiments [32]. We used the Annoy library as our approximate nearest neighbors algorithm [5] and Redis as the key-value store for the document fetch phase [27].

In both neural models we trained, which are described next, we use word embeddings provided by the FastText algorithm [6]. A few reasons motivated that decision: (1) the use of subword information makes not only semantically similar

words closer, but also morphologically similar words closer, which might be important for the soft term matching capabilities; (2) it can naturally deal with out-of-vocabulary words or typos; and (3) open availability of pre-trained embeddings for several languages, although only English is being used here. We used the English model trained from Wikipedia made available by the FastText team. The model has an embedding of 300 dimensions and its weights were not further adapted during training.

4.2. Retrieval model

For the retrieval model, as described in Section 3.2.1, we want to learn a query model and a document model, which we train to minimize the cosine distance between the query representation and the relevant documents representation.

Taking into account that the document can have multiple fields, the neural network chosen to encode each text field, i.e., query, document title and document body, was the Deep Averaging Network (DAN), motivated by its simplicity and successful use in the Universal Sentence Encoder [10,20]. In other words, to compute the tensor representation of each field, we take the average of the word embeddings, excluding stop words, and follow with a multi-layer feed-forward network, where the final layer's output is the field dense representation.

Inspired by the framework NRM-F, which incorporates multiple fields into a single document representation [49], we also concatenate the document title and body tensors to create a single document-level representation. We further apply a single feed-forward network over the concatenated tensor to properly combine the title and body importance and reduce the dimensionality to the proper size.

We tested [*relu*[1], *softsign*] activation functions for the hidden layers and [*tanh*[1], *softsign*] for the final representation layers. We experimented with [1[1], 3, 5] hidden layers for the DAN sub-models, [300[1], 512] vector representation sizes and [*mse*[1], *hinge*] loss functions. The Glorot normal initializer was used to initialize the weights.

Finally, we trained this model with both the pointwise and pairwise techniques [25]. We initially judged a simpler classification problem from the pointwise perspective would be good, since this model is going to be used only on the retrieval phase, without using its score for ranking. Our experiments showed, however, that the pairwise training leads to better results.

4.3. Ranking model

Motivated by its relative simplicity and good performance, we used the Deep Relevance Matching Model (DRMM) [17] as our ranking model. Following the best architecture of the original paper, we used the $DRMM_{LCH \times IDF}$ model, with the log-count value for the matching histogram and IDF weights for the term gating network. The IDF values were computed from the entire ClueWeb09 corpus.

We tested the neural network with the following configurations: [30, 60[1]] number of buckets, including the exact-match bucket, [5, 20[1]] hidden units in the histogram feed forward layer and with [*mse*[1], *hinge*] loss functions. The feed forward layers use the *tanh* activation function. The matching histograms is computed by matching the query terms and the document body terms with their normalized embedding vectors, therefore the matching scores range from [-1, 1]. Query and body were padded or truncated to fixed lengths of 10 and 1000, respectively. The query padding terms are masked out of the network final scoring, and the matching weight of padding terms is zero.

This is effectively a learn-to-rank problem and, as previous work suggests to be better [13], we used the pairwise strategy during training according to the score of the pseudo-labeler, rounded to the integer part, so that we only pair documents with a minimally significant difference in score.

4.4. Experimental results

The results presented here were drawn by running the queries from the TREC Web Track challenge of 2009–2012, for which there are, on average, 369 human-labeled relevance judgments per query. With 50 queries from each year, the test set has a total of 200 queries. In order to compare how the proposed neural architecture performs, we implemented four different setups:

1. **BM25**: traditional BM25 from Elasticsearch with top n results from a *multi-match* query over the title and body with equal weights. The index was setup with $b = 0.4$ and $k1 = 0.9$, ICU analyzer and English stop words filter;
2. **BM25-DRMM**: a mixed approach where we retrieve k candidate documents with Elasticsearch's BM25 algorithm and re-rank using the DRMM model, getting the top n, where $n \leq k$;
3. **ANN**: we take the top n results directly from Annoy's approximate nearest neighbours algorithm, ranked by the distances, effectively without a second step for re-ranking the retrieved documents;
4. **ANN-DRMM**: the full neural anatomy, using Annoy to retrieve k candidate documents and DRMM to rank them, selecting the top n, where $n \leq k$.

[1] Final selected value due to better validation metrics.

Table 1
Performance metrics for the different architectures and query types. Statistically significant improvements or degradations with respect to BM25, at the 0.05 level using the paired *t*-test, are indicated (+/-).

Query count		All 200	Precision-oriented 71	Recall-oriented 129
MAP	BM25	0.0935	**0.1054**	0.0871
	BM25-DRMM	**0.0955**	0.1046	**0.0906**
	ANN	0.0169$^-$	0.0144$^-$	0.0183$^-$
	ANN-DRMM	0.0474$^-$	0.0474$^-$	0.0474$^-$
MRR	BM25	0.4128	0.4466	0.3943
	BM25-DRMM	0.4840$^+$	**0.4773**	0.4877$^+$
	ANN	0.2454$^-$	0.1984$^-$	0.2711$^-$
	ANN-DRMM	**0.4933**$^+$	0.4626	**0.5101**$^+$
nDCG@20	BM25	0.1686	0.1967	0.1533
	BM25-DRMM	**0.2050**$^+$	**0.2146**	**0.1997**$^+$
	ANN	0.0638$^-$	0.0581$^-$	0.0670$^-$
	ANN-DRMM	0.1639	0.1604	0.1658
P@20	BM25	0.2192	0.2257	0.2156
	BM25-DRMM	**0.2614**$^+$	**0.2600**	**0.2621**$^+$
	ANN	0.0843$^-$	0.0807$^-$	0.0863$^-$
	ANN-DRMM	0.1970	0.1757	0.2086
R@20	BM25	0.0729	**0.1044**	0.0557
	BM25-DRMM	**0.0801**	0.0966	**0.0710**$^+$
	ANN	0.0209$^-$	0.0167$^-$	0.0231$^-$
	ANN-DRMM	0.0555$^-$	0.0526$^-$	0.0571
F1@20	BM25	0.1094	**0.1428**	0.0885
	BM25-DRMM	**0.1226**	0.1409	**0.1117**
	ANN	0.0335	0.0277	0.0364
	ANN-DRMM	0.0866	0.0810	0.0897

On all experiments, following the works in the literature, we used $k = 2000$ for the retrieval set, where appropriate, from which the top $n = 1000$ were selected after the respective ranking. We present the MAP and MRR for the entire list of results, and cut at 20 for nDCG, precision and recall.

4.4.1. All queries result

Table 1 summarizes the quality metrics for each architecture, and first we take a look into the "all" column, with the metrics for all 200 test queries.

Firstly, the BM25-DRMM architecture shows improvements over the BM25 implementation on all metrics. It is important to note that even though the retrieval is exact-match based, the retrieved documents most probably have other significant terms related to the query. E.g., if searching for "artificial intelligence", some retrieved documents most probably also contain terms like "machine learning" or "neural network". Whilst the pure BM25 will ignore those related terms, the neural model will capture these semantic relationships, explaining why it has learned to rank according to the weak labeler scoring, while also offering improvements through its soft-matching capabilities. This shows the high potential of neural IR, where neural models indeed are capable of surpassing the classical techniques at the ranking task, result also supported by several works in the literature [13,17,30,49].

The ANN architecture generally performs worse on every metric. This indicates how learning-to-rank with a **representation-focused** model is not suitable to get reasonable results in an ad-hoc search problem, especially when dealing with long documents, which can be noisy and cover several related topics. With this type of modeling, the representation loses detailed information, so important for the relevance matching problem, and only holds coarse grained semantic information about the document content. As previously reviewed, existing works in the literature obtain good results by considering only the document title and effectively without a retrieval phase, only scoring a labeled set of test query-document pairs. Hence, we conclude the secondary step for ranking the set of candidate documents is essential to the neural search engine anatomy.

On the other hand, the performance of the full neural architecture, ANN-DRMM, when compared to the BM25 baseline, is worse on the MAP metric, but better for MRR and comparable in nDCG@20. Aiming to understand better in which particular cases the neural anatomy performs better or worse than the classical approach, we looked at the individual queries in the test set, the focus of the next section.

4.4.2. Breaking down by query type

The literature provides a good framework to understand what are the user goals when web searching [41], and they can be broadly classified into "precision-oriented" or "recall-oriented". Precision-oriented queries are those for which usually a single correct result is enough to satisfy the user need, e.g., they can be navigational, when the user looks for an authoritative site or submits a question with a single unambiguous answer, as in "Obama family tree". For recall-oriented queries,

Table 2
Ranking quality metrics by varying retrieval size for the ANN-DRMM architecture. The greater the retrieval size, the better result we get, with predictable increase in cost.

k	MAP	MRR	nDCG@20	P@20	R@20	F1	Mean Latency	Latency CV
100	0.0236	0.3874	0.1062	0.1232	0.0309	0.0494	**0.139** ± **0.001**	0.0448 ± 0.0022
1000	0.0439	0.4633	0.1567	0.1902	0.0537	0.0838	1.271 ± 0.003	0.0222 ± 0.0011
2000	0.0474	0.4933	0.1639	0.1970	0.0555	0.0866	2.545 ± 0.006	0.0220 ± 0.0011
3000	0.0524	0.4889	0.1645	0.2035	0.0578	0.0900	3.823 ± 0.009	0.0207 ± 0.0010
4000	0.0547	**0.4970**	0.1688	0.2078	0.0590	0.0919	5.101 ± 0.011	0.0192 ± 0.0009
5000	0.0553	0.4906	0.1690	0.2093	0.0592	0.0923	6.375 ± 0.014	0.0192 ± 0.0009
6000	0.0557	0.4792	0.1693	0.2111	0.0589	0.0921	7.642 ± 0.017	0.0189 ± 0.0009
7000	**0.0576**	0.4811	**0.1742**	**0.2187**	**0.0607**	**0.0950**	8.923 ± 0.019	**0.0182** ± **0.0009**

the user usually benefits from several sources, like when searching for general information about a topic or an open-ended question, e.g., “diabetes education”.

We asked three colleagues to blindly and independently classify the test queries into either precision-oriented or recall-oriented. The final query classification was decided through majority vote, where we got a 0.534 Fleiss’ Kappa agreement score. The results of each set were analyzed separately, and they can be seen on the two rightmost columns of Table 1. When comparing between the precision-oriented and recall-oriented columns, we see how the difference in ranking quality are flipped when comparing the classical and neural architectures. Within the same architecture, whilst ANN and ANN-DRMM perform better on recall-oriented queries than on precision-oriented queries, the opposite is true for the classical BM25. The mixed BM25-DRMM architecture has comparable performance in both cases.

We hypothesize this difference is due to the fact that, for precision-oriented queries, the exact string match is essential for retrieving good candidate documents. For example, the ANN-DRMM architecture suffered one of the worst regressions for the query “alexian brothers hospital” (from 0.6226 to 0.0842 MAP). We see sensible results for other institutions, as rehabilitation center, veterinarian or retirement home, or for the history of the congregation, but in the end none of them matter for this query. Diversely, a query with great improvement was “uss yorktown charleston sc” (from 0.0209 to 0.1177 MAP), where the results range from its Wikipedia page and other history sites, to expositions and even paintings of the ship. In the case of this recall-oriented query, this diverse set of results is highly beneficial [24].

Despite the lower precision of the neural architecture, we think this loss in precision may not be catastrophic from the user perspective. While the MAP score, which considers the entire $n = 1000$ ranked results, is considerably worse for the ANN-DRMM architecture, the difference is not as significant on the other metrics. The reciprocal rank (MRR), for example, only cares about the first relevant result, and note how it is actually higher for the ANN-DRMM architecture. Cutting nDGC at 20, a more reasonable range from the user perspective, has a comparable performance.

Therefore, we can conclude that although the ANN has low precision, it does retrieve good enough candidates to capture at least a few relevant results, and the DRMM scoring is capable of ranking those results high in the list. A more detailed study is needed to understand if the experience is degraded from the user perspective [14].

4.4.3. Retrieval size impact

Another important aspect we wanted to understand is how the retrieval step influences the quality of the final result. For that, we run more experiments only for the complete neural architecture, ANN-DRMM, varying the size of the retrieved candidate set, *k*. As shown on Table 2, as we increase the retrieval size, all ranking metrics improve, except for MRR.

It is also important to point out how the latency increases linearly with *k*, and that is mainly due the ranking phase, which has to score each of the *k* candidate documents. This shows the importance of having a good retrieval phase, which allows the use of a smaller *k*, and improving the efficiency when probing a trained neural network, reducing the time taken on the ranking phase, which is further discussed in Section 5.3.

On the other hand, another important conclusion of this experiment, is how the neural architecture is controllable and predictable. The typical inverted index retrieves all the documents that contains at least one of the terms in the query, and we cannot know beforehand how big that will be. One can put a cap on this size to limit response time, as Google’s original paper cap of 40,000, on the cost of possibly having suboptimal results [8]. Several heuristics for pruning the retrieval set exist, aiming to minimize the suboptimality [45]. In a full neural IR architecture, we can control the size of the retrieved set, which brings up our ranking quality metrics, at the expense of longer and predictable response time.

4.5. Latency profiles

To illustrate how the latency profile changes on each architecture, note Fig. 4: for each of the 200 test queries and each of the four architectures, we plot how long it took to run them by the mean IDF of the query terms, excluding stopwords. The IDF is a measure of how rare a word is; the lower its value, the more common it is. Therefore, a low term IDF is expected to be present in several documents, whilst a high IDF term is expected to be on few documents.

As expected, Fig. 4 shows an increase on latency and variance for lower IDF queries, for both the BM25 and BM25-DRMM architectures. That happens because, the lower the IDF, the bigger it is the retrieved set, which also takes longer to rank.

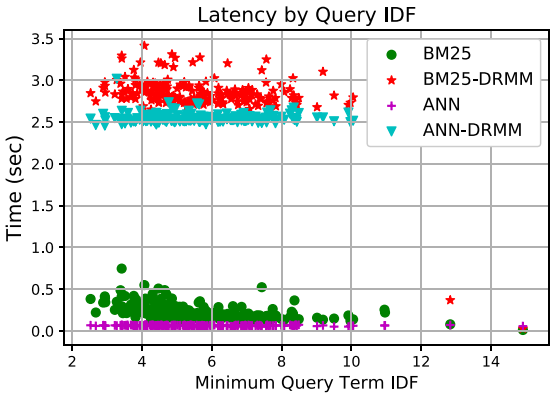

Fig. 4. BM25 and BM25-DRMM architectures present a highly query-dependent latency profile. Queries with common terms take longer and have a greater variance on the response time. The ANN and ANN-DRMM architectures presents a stable latency profile (best viewed in colors).

Table 3
Latency profile on different architectures. The ANN-DRMM architecture has constant and predictable latency, contrary to the BM25 architecture.

	mean (sec)	std (sec)	CV
Total BM25	0.228 ± 0.011	0.098 ± 0.008	0.430 ± 0.021
BM25 Retrieval	0.338 ± 0.013	0.115 ± 0.009	0.342 ± 0.017
Redis Fetch	0.416 ± 0.010	0.088 ± 0.200	0.213 ± 0.011
DRMM Rank	2.078 ± 0.023	0.200 ± 0.015	0.096 ± 0.005
Total BM25-DRMM	2.834 ± 0.035	0.303 ± 0.024	0.107 ± 0.005
Total ANN	**0.064 ± 0.001**	**0.005 ± 0.001**	0.080 ± 0.004
Annoy Retrieval	0.068 ± 0.001	0.009 ± 0.001	0.132 ± 0.006
Redis Fetch	0.363 ± 0.006	0.055 ± 0.004	0.151 ± 0.008
DRMM Rank	2.111 ± 0.001	0.008 ± 0.001	0.004 ± 0.001
Total ANN-DRMM	2.545 ± 0.006	0.056 ± 0.004	**0.022 ± 0.001**

Note how the latency in the ANN and ANN-DRMM architectures have a negligible influence from the query IDF, exactly because we control the retrieval size.

Table 3 shows a more detailed latency profile for the four architectures we tested, where we estimated the mean, standard deviation and coefficient of variance of the latency. Both in the BM25-DRMM and ANN-DRMM architectures, we breakdown the latency for each of the retrieval, document fetch and ranking phases. We also present the estimation error, considering the sample size of 200, the size of test queries set.

The BM25 experiment is, on average, one order of magnitude faster than the BM25-DRMM and ANN-DRMM. That is a result of years of research and investments on optimizing algorithms, data structures and infrastructure [46]. Note how the longest step in both the BM25-DRMM and ANN-DRMM architectures is the ranking one, showing how a good retrieval step is important, because actually running the ranking function on the entire corpus or a large candidate set is unfeasible. The ANN architecture is yet another order of magnitude faster, but as we have seen before, its ranking quality is low. More importantly, following our discussion of controllability and predictability, note how ANN and ANN-DRMM have a coefficient of variance one order of magnitude lower than BM25, showing how the regularities of neural models and approximate nearest neighbours algorithms bring a lot of stability, a desirable property for large scale and production grade systems.

5. Discussion

In this section we wrap up the discussion over the obtained results under a more qualitative view, going over the questions that motivated our work, as laid out in Section 1. We also point out important future work, for both academia and the industry at large, in order to move towards the development of more deep learning centric systems.

5.1. Query type and other IR tasks

The experimental results from Section 4 show that the neural models tend to produce better rankings for recall-oriented queries, where the soft-matching capabilities brings greater benefits. Conversely, the loss of fine grained information when encoding a text document into a fixed-length vector representation leads to a loss in performance for precision-oriented queries.

We consider that the major challenge here is representing a long document, which can be noisy and cover several related topics, into a low dimension fixed-length vector. A lot of work has been done in sentence level representation, from a

Table 4
Memory requirements and monthly cost per architecture, where C is the number of documents in the corpus and $\hat{d}$ is the average length of the documents.

Index / Model	Size (GB)	Space Complexity	In-memory	BM25	BM25-DRMM	ANN	ANN-DRMM
Elasticsearch	311	$O(C * \hat{d})$		✓	✓		
Annoy	90	$O(C^*ntrees)$	✓			✓	✓
Redis	130	$O(C * \hat{d})$	✓		✓	✓	✓
Query Model	1.5	$O(1)$	✓			✓	✓
DRMM	1.5	$O(1)$	✓		✓		✓
		Monthly cost (USD)		870	2070	2400	2400

simple word embedding sum to the Universal Sentence Encoder [10]. However, when trying to represent a long document with hundreds or thousands of words, as used in this work, these techniques still fall short. We believe this is one of the main obstacles for our proposed neural search engine, where in the retrieval step, we need to have a generic document representation capable of holding fine grained information and matching independent queries, especially for the precision-oriented cases. Learning good document-level embeddings is currently one of the main challenges for the NLP community.

Some alternatives to work around this limitation arise. e.g., we can combine two different retrieved sets, or the final ranked sets, one from traditional and another from neural architectures, getting the best of both worlds [15]. Or we can introduce yet another step into the architecture, classifying the query type beforehand and using the appropriate model/algorithm to retrieve and rank the results. These solutions lead to a mixed system, beating to purpose of this work of studying a full neural model centric system, and were not investigated further.

Alternatively, one can work on paragraph or sentence level representation, instead of the full document [35]. This would require bigger indices and a more complex mapping between the several paragraphs per relevant document, but would allow for a finer representation. This is one direction we would like to investigate in future work.

In addition, neural IR is a very hot research topic as of now. It remains a challenge and is still an open question in the literature how to engineer or explain the neural network capabilities for word sense disambiguation, synthesis, semantic processing and similarity encoding, so important an IR task. We would like to explore more of these neural capabilities in future work. Nonetheless, there is a constant wave of new neural network structures to learn more intuitive representations, as well as improving ranking quality, even if they might lack in interpretability [28,37]. Furthermore, some works in the literature focus on leveraging the solid progress of the traditional IR community, either with the infrastructure, e.g., by incorporating a neural network into a Lucene search engine [43], or by mixing the carefully engineered and effective features into the model [31]. More recent work showcase the capabilities of neural IR models across a search session [12]. In any case, as shown in this work, neural IR tend to have better semantic matching, more stable performance, and feature robustness [35], and although it might not be there yet, we believe it will outperform the current technology in the short term.

Finally, we believe the neural architecture can be useful to other types of IR tasks, such as question answering/selection or paraphrase detection, particularly because they deal with smaller texts, usually only a few sentences long. As we understand, the same neural architecture is flexible enough to work with any other text-based IR without much specialized tunning, requiring only the techniques familiar to the machine learning community, making it more accessible from the engineering perspective. Exploring its capabilities on other problems is also an important future work.

5.2. Data need

It is important to point out that the BM25 approach needs no training data at all, since it is a closed formula ready to go, which makes it very accessible. That said, accessing and handling data has never been easier. Even if human-annotated data is still expensive, supported by some previous results, we have shown that neural models are able to learn and improve upon traditional techniques by using a weak supervision approach [13]. That is good news, specially for smaller companies that can not count with a plethora of data and query/activity logs, democratizing access to state-of-the-art models.

Finally, even though weak supervision is great to bootstrap the model, better data will lead to better models. We believe the neural models, as presented here, could still achieve great results on precision-oriented queries, if fed with better data. Rich real-world click-logs, for example, would teach the neural network that some types of queries, like well defined institutions or names, needs a finer matching, semantic similarity is not enough, and that would lead the models to allocate more bits to represent those cases.

5.3. Computational cost and resources

We tried to keep the cost of serving as similar as possible on all experiments, but the in-memory allocation requirements of the neural architecture scales up the pricing. Table 4 summarizes the requirements of each component and which are needed for each architecture.

Specifically, we used the Elasticsearch Service cluster from AWS, version 5.1, with two node instances of type `m4.2xlarge` and 512 GB general purpose SSD each. To serve the neural architecture, we opted for a single machine that loads TensorFlow-Serving, Redis and Annoy all locally. We used a `x1e.4xlarge` instance, which has 16 CPU cores and 488 GB of RAM. The costs on Table 4 are approximate for on-demand machines on AWS `us-east-1a` region, as of July 2018. Note we chose not to serve the TF models using accelerated computing from a GPU or TPU, since that represents a significant increase in cost, although it could notably reduce the ranking time.

Besides the consistent latency profile discussed in Section 4.5, the neural anatomy has predictable and partially controllable memory requirements. The Annoy index size grows linearly with the number of documents in the corpus, no matter their lengths. Furthermore, the number of trees used when building the index offers a trade-off between index size and accuracy.

On the other hand, the Redis index size depends not only on number of documents, but also on their lengths. Depending on the neural model chosen for the ranking phase, one can pad/truncate the document fields, since many neural models expect fixed-size inputs.

We believe reducing the cost of accelerated computing and inference time optimizations like NVIDIA's TensorRT and Tensor Processing Units are important steps for the industry, and more advancements on that direction are going to allow a broader use of deep learning centric systems.

5.4. Engineering cost

Building and maintaining a machine learning platform in-house is still a big challenge. The work of training a model and testing it on a test dataset is far from being enough. We still need to have a serving infrastructure, production grade robustness, be able to experiment with several neural network models and conduct tests, all of which the industry is still very actively working on. This can lead to a lot of glue code, which is hard to maintain and evolve, and require a lot of re-work.

Working with traditional and well established techniques, on the contrary, is not only easier to implement, due to the vast literature and real-case examples, but also easier to find open source solutions and SaaS/IaaS providers.

That said, we believe the neural IR architecture provides a standard way of adding new features and evolving the system. For example, a document can be composed of several other text fields, like anchor texts and clicked queries. Under a neural framework, adding new fields can be done in a standardized way, without special treatment for each case [49], since, in the end, everything boils down to a fixed length vector. On the other hand, under a traditional framework, each new field can change requirements and performance in different ways, due to different sizes or vocabulary distribution, demanding a detailed individual study for each case.

Finally, the deep learning community has grown a lot in recent years and, consequently, several libraries, frameworks and services have been created. We believe it is a matter of time for the industry to find a good common ground of standards, cross-platforms compatibility and a way of sharing models similar to how we share libraries today. There has been great advancements with several commercial cloud services providers focusing on ML platforms, which will definitely facilitate access for smaller engineering teams. Furthermore, transfer learning has been an important aspect for the research community. For example, the most popular word embedding models, Word2Vec, GloVe and FastText, have been made freely available online, for several languages. More recent efforts for sharing complete models and promoting transfer learning have been made, and we believe that is going to be a great tool not only for the research community, but mainly the industry, freeing up engineering teams from building everything from scratch [4,10].

6. Conclusions

In this work, we have demonstrated a novel way of developing a search engine using modern neural network techniques, with strong repercussions on how we deploy, scale and evolve such a system. We have implemented and evaluated a fully neural search engine architecture, where both the retrieval and ranking phases are provided by neural models, each with an appropriate design for high-scalability and good relevance matching capability, respectively.

Our implementation allowed us to evaluate the neural IR architecture under a more practical lenses and consider several engineering aspects, often neglected in the academic literature, whose focus is on the mathematical foundations and neural network architectural decisions. Compared to traditional large scale search engines, we find the neural search engine still have a higher cost from the latency and memory usage perspective, but they do offer better predictability and stability, important concerns for the industry.

Furthermore, building a neural search engine requires little understanding of the complex data structures trade-offs which are intertwined with the ranking work in a distributed inverted list setup. Therefore, pretty much the same neural search engine setup can be used for several IR tasks, with different corpus characteristics, with little special tunning necessary, only requiring the techniques familiar to the machine learning community for training the neural network models.

On what concerns the ranking quality work, by only applying consolidated and easily reproducible neural IR models, we find that the neural search engine can already achieve state-of-the-art results. We further observe the neural IR show more important progress on recall-oriented queries, where the user goal is broad and a diverse set of results is important.

There is still important developments to be done, e.g., on document-level embedding representation, in particular, and transfer learning and neural network inference time optimizations, in general. Moreover, the neural NLP models still lack in intuition interpretability and debuggability, important aspects of commercial solutions. Besides all that, we believe deep learning centric systems are going to be more and more important in the coming years, and this work is one significant step towards exploring and understanding that world.

Acknowledgments

This research was supported by WorldSense. We are thankful for the provided resources, and also for the insight and expertise from our colleagues that greatly assisted the research.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ins.2018.12.041.

References

[1] M. Abadi, et al., TensorFlow: large-scale machine learning on heterogeneous systems, 2015, https://www.tensorflow.org/, Software available from tensorflow.org.
[2] A. Abdulkader, A. Lakshmiratan, J. Zhang, Introducing deeptext: facebooks text understanding engine, https://code.fb.com/core-data/introducing-deeptext-facebook-s-text-understanding-engine/, 2016.
[3] M. Aumüller, E. Bernhardsson, A. Faithfull, Ann-benchmarks: a benchmarking tool for approximate nearest neighbor algorithms, in: C. Beecks, F. Borutta, P. Kröger, T. Seidl (Eds.), Similarity Search and Applications, Springer International Publishing, Cham, 2017, pp. 34–49, doi:10.1007/978-3-319-68474-1_3.
[4] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, J.W. Vaughan, A theory of learning from different domains, Mach. Learn. 79 (1–2) (2010) 151–175, doi:10.1007/s10994-009-5152-4.
[5] E. Bernhardsson, Annoy, 2013, https://github.com/spotify/annoy.
[6] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Transactions of the Association for Computational Linguistics 5 (2017) 135–146.
[7] L. Boytsov, D. Novak, Y. Malkov, E. Nyberg, Off the beaten path: Let's replace term-based retrieval with k-nn search, in: Proceedings of the Twenty-Fifth ACM International on Conference on Information and Knowledge Management, 2016, pp. 1099–1108, doi:10.1145/2983323.2983815.
[8] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Comput. Netw. ISDN Syst. 30 (1–7) (1998) 107–117, doi:10.1016/S0169-7552(98)00110-X.
[9] J.D. Brutlag, H. Hutchinson, M. Stone, User preference and search engine latency, in: Proceedings of the Quality and Productivity Research Section JSM, 2008.
[10] D. Cer, Y. Yang, S.-Y. Kong, N. Hua, N. Lyn Untalan Limtiaco, R.S. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, R. Kurzweil, Universal sentence encoder, EMNLP demonstration, 2018 arxiv: 1803.11175.
[11] G.V. Cormack, M.D. Smucker, C.L. Clarke, Efficient and effective spam filtering and re-ranking for large web datasets, Inf. Retr. 14 (5) (2011) 441–465, doi:10.1007/s10791-011-9162-z.
[12] M. Dehghani, S. Rothe, E. Alfonseca, P. Fleury, Learning to attend, copy, and generate for session-based query suggestion, in: Proceedings of the ACM on Conference on Information and Knowledge Management, ACM, New York, NY, USA, 2017, pp. 1747–1756, doi:10.1145/3132847.3133010.
[13] M. Dehghani, H. Zamani, A. Severyn, J. Kamps, W.B. Croft, Neural ranking models with weak supervision, in: Proceedings of the Fortieth International ACM SIGIR Conference on Research and Development in Information Retrieval, 2017, pp. 65–74, doi:10.1145/3077136.3080832.
[14] A. Fan, L. Chen, G. Chen, A multi-view semi-supervised approach for task-level web search success evaluation, Inf. Sci. 430–431 (2018) 554–566, doi:10.1016/j.ins.2017.12.003.
[15] M. Ganzha, M. Paprzycki, J. Stadnik, Combining information from multiple search engines preliminary comparison, Inf. Sci. 180 (10) (2010) 1908–1923, doi:10.1016/j.ins.2010.01.010. Special Issue on Intelligent Distributed Information Systems
[16] C. Gormley, Z. Tong, Elasticsearch: The Definitive Guide, first ed., O'Reilly Media, Inc., 2015.
[17] J. Guo, Y. Fan, Q. Ai, W.B. Croft, A deep relevance matching model for ad-hoc retrieval, in: Proceedings of the Twenty-Fifth ACM International on Conference on Information and Knowledge Management, 2016, pp. 55–64, doi:10.1145/2983323.2983769.
[18] B. He, J.X. Huang, X. Zhou, Modeling term proximity for probabilistic information retrieval models, Inf. Sci. 181 (14) (2011) 3017–3031, doi:10.1016/j.ins.2011.03.007.
[19] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, L. Heck, Learning deep structured semantic models for web search using click-through data, in: Proceedings of the Twenty-Second ACM International Conference on Information & Knowledge Management, 2013, pp. 2333–2338, doi:10.1145/2505515.2505665.
[20] M. Iyyer, V. Manjunatha, J. Boyd-Graber, H. Daumé III, Deep Unordered Composition Rivals Syntactic Methods for Text Classification, Association for Computational Linguistics, 2015, pp. 1681–1691, doi:10.3115/v1/P15-1162.
[21] M. Jeon, S. Kim, S.-w. Hwang, Y. He, S. Elnikety, A.L. Cox, S. Rixner, Predictive parallelization: taming tail latencies in web search, in: Proceedings of the Thirty-Seventh International ACM SIGIR Conference on Research 38; Development in Information Retrieval, 2014, pp. 253–262, doi:10.1145/2600428.2609572.
[22] K.S. Jones, A statistical interpretation of term specificity and its application in retrieval, J. Doc. 28 (1972) 11–21.
[23] J. Li, X. Chen, E. Hovy, D. Jurafsky, Visualizing and understanding neural models in NLP, in: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, 2016, pp. 681–691, doi:10.18653/v1/N16-1082.
[24] J. Li, Y. Wu, P. Zhang, D. Song, B. Wang, Learning to diversify web search results with a document repulsion model, Inf. Sci. 411 (2017) 136–150, doi:10.1016/j.ins.2017.05.027.
[25] T.-Y. Liu, Learning to rank for information retrieval, Found. Trends Inf. Retr. 3 (3) (2009) 225–331, doi:10.1561/1500000016.
[26] Y. Liu, H. Wei, H. Cheng, Exploiting lower bounds to accelerate approximate nearest neighbor search on high-dimensional data, Inf. Sci. (2018), doi:10.1016/j.ins.2018.07.005.
[27] T. Macedo, F. Oliveira, Redis Cookbook, O'Reilly Media, Inc., 2011.
[28] R. McDonald, G. Brokos, I. Androutsopoulos (Eds.), Deep Relevance Ranking Using Enhanced Document-Query Interactions, 2018.
[29] B. Mitra, N. Craswell, Neural models for information retrieval, CoRR (2017) arXiv: 1705.01509.
[30] B. Mitra, F. Diaz, N. Craswell, Learning to match using local and distributed representations of text for web search, in: Proceedings of the Twenty-Sixth International Conference on World Wide Web, 2017, pp. 1291–1299, doi:10.1145/3038912.3052579.
[31] G. Nguyen, L. Soulier, L. Tamine, N. Bricon-Souf, DSRIM: a deep neural information retrieval model enhanced by a knowledge resource driven representation of documents, CoRR abs/1706.04922 (2017).

[32] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, J. Soyke, Tensorflow-serving: flexible, high-performance ML serving, CoRR (2017) arXiv: 1712.06139.
[33] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: bringing order to the web, Technical Report 1999-66, Stanford InfoLab, 1999.
[34] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, R. Ward, Deep sentence embedding using long short-term memory networks: analysis and application to information retrieval, IEEE/ACM Trans. Audio Speech Lang. Proc. 24 (4) (2016) 694–707, doi:10.1109/TASLP.2016.2520371.
[35] L. Pang, Y. Lan, J. Guo, J. Xu, X. Cheng, A deep investigation of deep IR models, CoRR abs/1707.07700 (2017).
[36] L. Pang, Y. Lan, J. Guo, J. Xu, S. Wan, X. Cheng, Text matching as image recognition, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016, pp. 2793–2799.
[37] L. Pang, Y. Lan, J. Guo, J. Xu, J. Xu, X. Cheng, Deeprank: a new deep architecture for relevance ranking in information retrieval, in: Proceedings of the ACM on Conference on Information and Knowledge Management, 2017, pp. 257–266, doi:10.1145/3132847.3132914.
[38] G. Pass, A. Chowdhury, C. Torgeson, A picture of search, in: Proceedings of the First International Conference on Scalable Information Systems, 2006, doi:10.1145/1146847.1146848.
[39] S. Robertson, H. Zaragoza, M. Taylor, Simple BM25 extension to multiple weighted fields, in: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management, 2004, pp. 42–49, doi:10.1145/1031171.1031181.
[40] S.E. Robertson, S. Walker, Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval, in: Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1994, pp. 232–241.
[41] D.E. Rose, D. Levinson, Understanding user goals in web search, in: Proceedings of the Thirteenth International Conference on World Wide Web, 2004, pp. 13–19, doi:10.1145/988672.988675.
[42] Y. Shen, X. He, J. Gao, L. Deng, G. Mesnil, Learning semantic representations using convolutional neural networks for web search, in: Proceedings of the Twenty-Third International Conference on World Wide Web, 2014, pp. 373–374, doi:10.1145/2567948.2577348.
[43] Z. Tu, M. Crane, R. Sequiera, J. Zhang, J. Lin, An exploration of approaches to integrating neural reranking models in multi-stage ranking architectures, CoRR abs/1707.08275 (2017).
[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 5998–6008.
[45] Q. Wang, C. Dimopoulos, T. Suel, Fast first-phase candidate generation for cascading rankers, in: Proceedings of the Thirty-Ninth International ACM SIGIR Conference on Research and Development in Information Retrieval, 2016, pp. 295–304, doi:10.1145/2911451.2911515.
[46] I.H. Witten, A. Moffat, T.C. Bell, Managing Gigabytes (2Nd ed.): Compressing and Indexing Documents and Images, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
[47] C. Xiong, Z. Dai, J. Callan, Z. Liu, R. Power, End-to-end neural ad-hoc ranking with kernel pooling, in: Proceedings of the Fortieth International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, 2017, pp. 55–64, doi:10.1145/3077136.3080809.
[48] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, in: Proceedings of the Second USENIX Conference on Hot Topics in Cloud Computing, 2010, p. 10.
[49] H. Zamani, B. Mitra, X. Song, N. Craswell, S. Tiwary, Neural ranking models with multiple document fields, in: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, 2018, pp. 700–708, doi:10.1145/3159652.3159730.
[50] W. Zaremba, I. Sutskever, Learning to execute, CoRR abs/1410.4615 (2014).