Deploying deep ranking models for search verticals

Rohan Ramanath, Gungor Polatkan, Liqin Xu, Harold Lee, Bo Hu, Shan Zhou
LinkedIn Corporation
{rramanath,gpolatkan,lxu,hlee,bohu,shzhou}@linkedin.com

1 INTRODUCTION

Capturing the semantic similarity between a query and a set of documents is a well studied problem in the Information Retrieval community [3, 10, 11].

LinkedIn Talent Solutions (LTS) provides innovative tools to help talent searchers (e.g. recruiters, hiring managers and corporations) around the world become more successful at talent acquisition. One important challenge is to translate the criteria of a hiring position into a search query; the searcher has to understand which skills are typically required for a position, which companies are likely to have such candidates, which schools the candidates are most likely to graduate from, etc. Moreover, the knowledge varies over time. As a result, as indicated by LinkedIn search log data [7], often multiple attempts are required to formulate a good query. To help the searcher, LTS search provides advanced targeting criteria called *facets* (i.e. skills, schools, companies, titles and many more). The query can be entered as free text, a facet selection or the combination of the two. This results in queries where semantic interpretation and segmentation becomes important, e.g. in the query "java" or "finance" the searcher could be searching for a candidate whose title contains the word or someone who knows a skill represented by the word. Relying on exact term or attribute match in faceted search for ranking is sub-optimal. In this paper, we investigate a method to improve the solution to the matching and ranking problem rather than focus on the query formulation.

Latent semantic models are commonly used to map a noisy high dimensional query to a low-dimensional representation to make the matching problem tractable [6]. We extend latent semantic models with a deep structure by projecting queries and talent attributes into a shared low-dimensional space where the relevance of a talent given a query is readily computed as the distance between them. In this paper, we propose an architecture in which a neural network scoring a query-member pair is split into 3 semantic pieces such that each piece is scored on a separate system with its own characteristics. Additionally, we implement and experiment with one specific instance of this architecture in production, computing semantic similarity (used in a downstream learning to rank model [5]) using online low-dimensional vector representations in a scalable way (being able to score millions of LinkedIn members) without compromising system performance or site stability.

2 MODELING

The problem of talent search can be formulated as follows; given a query q by a recruiter r, rank a list of candidate LinkedIn members $m_1, m_2, ..., m_d$ in decreasingly relevant order by learning a function (in this case a neural network scoring a query-member pair) , $f : q(r, m_j) \mapsto s_{i,j} \in \mathbb{R}$. For the purposes of this paper, we make the model independent of the recruiter (r). Since our goal is to productionize this function, we study the characteristics of each

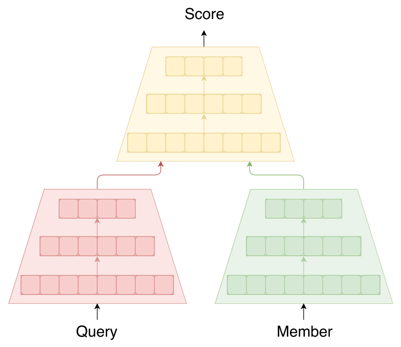

Figure 1: A typical siamese network with an additional crossing network. Although, this is trained as a single network using query log data, the architecture is split into 3 semantic components during inference, which are implemented on separate physical systems

system, and consequently each semantic piece (Figure 1), required to serve a search result in Section 3.

In literature there have been different efforts trying to address similar problems. The models considered [8, 9, 13, 14] were pointwise methods with the focus on learning a function that scores the similarity between the query and a candidate(s). Our architecture in Figure 1 is a generalization of such models. In such a framework, the degree of model complexity of each module is dictated by 1) implementation and serving constraints, 2) requirements specified by a Service Level Agreement (SLA).

One drawback of the models in [8, 9, 13, 14] is that they only consider text data. In LTS search, the query and talent are represented by multiple sources of data (profile picture, education, job history, skills and many more facets) and not just text. The problem of combining heterogeneous data of different modalities adds complexity to the ranking model. In our experiments, we use the late crossing [13] variant of siamese networks [4] since it allows us to compute scores within strict SLAs to be served online.

As shown in Figure 2, the input to the model is a combination of text and facet attributes. Each input layer converts the incoming attribute / text (ngram) from a list of categorical features to a single embedding (via pooling) and the aggregation layer simply stacks embeddings from multiple attributes to one vector. Since the member arm has a richer source of input data, there is more opportunity to learn representative structures. This intuition manifests itself via a deeper and structurally richer (i.e. convolutions) member arm that eventually produces the member representation. The shorter query arm leverages query text and facets selected by the recruiter in the search UI to produce a query representation. The similarity layer (fully-connected, cosine, or any distance function) processes the query and member representations to produce a score that captures semantic similarity.

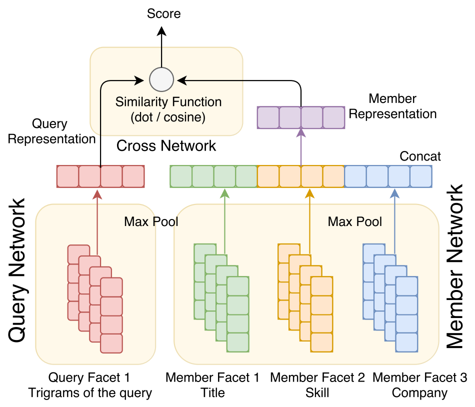

Figure 2: The two arm architecture with a shallow query arm (it is scored at real time) and a deep member arm. Note that the member arm is not only wider but also can be deeper.

3 SYSTEM DETAILS

There are 3 sections of the scoring flow serving a query-member pair: (1) Offline distributed processing (e.g. Hadoop, Spark) for member network, (2) Online REST Service [2] for query network, and (3) Galene [12], the LinkedIn search-as-a-service infrastructure for cross network as a final scoring.

The offline distributed piece is used mostly for member network processing. Since the member profiles (education, job history, skills and many more facets) are known offline, we pre-compute the member representation using Tensorflow [1] on Hadoop, compress and store this resultant vector in the forward index of the searcher. One can tolerate infrequent updates to the member representation because the member profile information is relatively static. Additionally, since the member representation is evaluated offline, we can experiment with more aggressive architectures (and depth) for the member arm.

The online REST service is responsible for processing each search request. The query is evaluated and processed on the fly to extract query features like trigrams of text and search facets such as skill, title, company. The query network uses this as the input to produce a query representation as the output. Since the module is scored at real time and has tight SLAs, the network complexity is limited by the time to score. To simplify the discussion, let us assume we just have one attribute, i.e. title (t) on both the query and member side. Everything that follows can be easily extended to any number and types of attributes. A key-value store is used to store attribute, facet vectors, i.e. one vector for every title t_i (or one vector for every *ngram* if we consider *ngrams* of the raw text as the query feature). The search frontend parses the query (the tagged textual query and selected facet) to determine all the titles targeted by the viewer. The vectors corresponding to all the targeted titles are retrieved and the query arm of the network is evaluated in the search frontend. The resultant query representation is then inserted into the query meta data in the call to the search backend. Although evaluating the query arm can be computationally expensive (depending on the depth), this happens only once for a search request unlike the member arm of the network. An alternate solution could involve pre-computing and storing the query representation for the head queries and then directly retrieving them. However, further analysis of the query distribution did not reveal a power law, mostly because of the complexity introduced by facets and their interaction with the free-form text query.

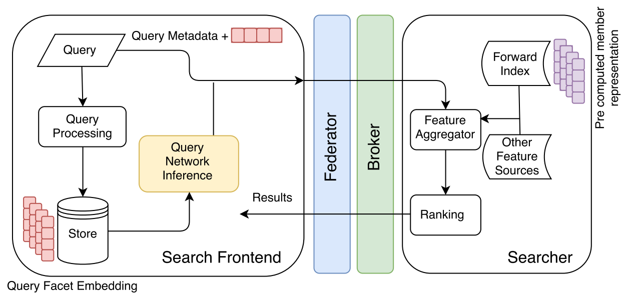

Figure 3: Implementation of the query and member similarity within the search infrastructure.

The third piece of the production pipeline is the LinkedIn search-as-a-service infrastructure, Galene. The offline generated member representation and REST service generated query representation are unified on the search nodes where the final piece of the scorer is evaluated. Galene is built over Lucene and most concepts discussed here will apply to other search frameworks. Sankar and Makhani [12] give a good overview of the Galene stack and the life cycle of a search query is shown in Figure 3. An important design decision in Galene that provides context to this work is that the backend (federator, broker and searcher in Figure 3) should be self-sufficient and is not allowed to make external service calls. This design allowed for the search backend to be run against a suite of integration tests that evaluate the quality of the search index and ranking models before deployment. A side affect of this design is that it prevents one from using an external key-value store to store the pre-computed member representation. At request time, once the members have been retrieved for the query q, each member's representation (via the forward index) along with the query representation (via the request to the backend) are evaluated via the similarity layer in the searcher to produce a score for every query-member pair. This is then used as a feature in the ranking model.

4 RESULTS AND CONCLUSION

The system (Section 3) was ramped to 100% of the LTS traffic and the feature generated by the network (top 3 most important features in the model) is currently being used to rank search results. In terms of system performance, there was no statistically significant difference in the latency (p50, p90, p99) of the search backend. Scoring the query network on the search frontend added $3ms$ (p99) to the latency which was well within the SLA requirements since it needed to be computed only once per request.

To summarize our contributions, (1) We demonstrate the use of NN based embeddings to improve the relevance of results, (2) We propose an architecture that can be scored and leveraged by a real time production service, (3) We show that our system scales without any performance impact.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[2] Joe Betz and Moira Tagle. 2013. Rest.li: RESTful Service Architecture at Scale. (Feb. 2013). Retrieved December 20, 2017 from https://engineering.linkedin.com/architecture/restli-restful-service-architecture-scale

[3] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. 2007. Measuring semantic similarity between words using web search engines. *www* 7 (2007), 757–766.

[4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1994. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*. 737–744.

[5] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. ACM, 129–136.

[6] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391.

[7] Viet Ha-Thuc, Ye Xu, Satya Pradeep Kanduri, Xianren Wu, Vijay Dialani, Yan Yan, Abhishek Gupta, and Shakti Sinha. 2016. Search by Ideal Candidates: Next Generation of Talent Search at LinkedIn. In *Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 195–198. https://doi.org/10.1145/2872518.2890549

[8] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*. 2042–2050.

[9] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2333–2338.

[10] Michael Mohler and Rada Mihalcea. 2009. Text-to-text semantic similarity for automatic short answer grading. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 567–575.

[11] M Andrea Rodríguez and Max J. Egenhofer. 2003. Determining semantic similarity among entity classes from different ontologies. *IEEE transactions on knowledge and data engineering* 15, 2 (2003), 442–456.

[12] Sriram Sankar and Asif Makhani. 2014. Did you mean "Galene"? (June 2014). Retrieved December 15, 2017 from https://engineering.linkedin.com/search/did-you-mean-galene

[13] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep Crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 255–262.

[14] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 373–374.