

PinText 2: Attentive Bag of Annotations Embedding

Jinfeng Zhuang
jzhuang@pinterest.com
Pinterest
Seattle, WA

Jennifer Zhao
jzhao@pinterest.com
Pinterest
San Francisco, CA

Anant Srinivas Subramanian
asubramanian@pinterest.com
Pinterest
San Francisco, CA

Yun Lin
ylin@pinterest.com
Pinterest
San Francisco, CA

Balaji Krishnapuram
bkrishnapuram@pinterest.com
Pinterest
San Francisco, CA

Roelof van Zwol
rvanzwol@pinterest.com
Pinterest
San Francisco, CA

ABSTRACT

Word embedding is a basic building block in text processing tasks like classification, retrieval, and ranking. However, it is not a trivial task to aggregate a collection of word embeddings to a single vector representation. In particular, we have an annotation system at Pinterest where a set of concrete annotation terms are used to describe the content of a pin. Although we use a pre-trained PinText model [40] to derive the annotation embeddings, using the average of annotation embeddings as the pin's embedding is not always optimal for a particular application. This leads to a common choice that practitioners often have to make: pre-train a unified word embeddings and fine-tune it everywhere, or learn the embeddings end-to-end together with the task-dependent machine learning model parameters. In this paper, we focus on the best pin-level embeddings given pins' annotation set as the text input. We extend PinText to the second version by the following improvements applicable to scenarios where bag of text snippets are input: 1) extend the word embedding dictionary in the multitask learning setting to include phrase-level embeddings using a data-driven approach to identify the most important phrase dictionary; 2) propose a deep neural networks model architecture with attention mechanisms to best combine pin's annotation embeddings; 3) conduct thorough study on the performance gap between pre-trained and end-to-end embeddings.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations; Multi-task learning.**

KEYWORDS

Text Embedding, Multitask Learning, Attention Mechanism, Bag of Words;

ACM Reference Format:

Jinfeng Zhuang, Jennifer Zhao, Anant Srinivas Subramanian, Yun Lin, Balaji Krishnapuram, and Roelof van Zwol. 2020. PinText 2: Attentive Bag of Annotations Embedding. In *Proceedings of DLP-KDD 2020*. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

With more than 300 million people (pinner) come to Pinterest app looking for inspirations every month, we are able to convert such huge amount of users' engagements into supervised information, which leads to various stronger machine learning models that in turn help improve pinners' experience, than models learned in an unsupervised way. In particular, embedding models which map a concrete object to a real vector representation, including GraphSAGE embedding [12] on pins, Act-A-Like embedding [7] on users, Universal Visual Embedding [39] on images, and PinText embedding [40] on text snippets like search query or pin's title, are super useful due to the fact that they can be easily plugged into existing models or systems taking real vectors as input. They also produce off-the-shelf feature vectors for a cold start on a new application. Those embedding solutions have been playing important roles in the various machine learning-based backend retrieval and ranking systems.

In this paper, we focus on improving text embedding models. PinText [40] is a static pre-trained word-level embedding model at Pinterest aiming to capture word semantics. It has exhibited better performance in nearest neighbor retrieval than other open sourced text embedding models like word2vec [21] or fastText [11]. This makes it very useful for query expansion or embedding based pin retrieval. We conclude that the gain is most probably from two facts: 1) it is supervised training instead of unsupervised training as in word2vec; 2) it uses Pinterest user engagement data from multiple surfaces that fit the Pinterest application better. However, despite the gains, this PinText V1 still has the common limitations of pre-trained word embeddings algorithms. In particular, there are two aspects we hope to study further:

- It is not trivial to construct object-level representation from word-level embeddings. We use the average of word embeddings as default aggregation which is not necessarily optimal;
- The word semantics may depend on its context, but pre-trained word embeddings are static. They cannot change with different context;

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DLP-KDD 2020, August 24, 2020, San Diego, California, USA

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

(a) Pin Image	(b) Predicted top annotation terms derived from image and raw text	
	Annotation Terms	Score
	Balboa park san diego	0.97
	San diego california	0.95
	Balboa park	0.94
	Beautiful places	0.90
	Favourite places	0.90
	Places to go	0.89
	ferry building san francisco	0.89
	Places to travel	0.89
	Balboa	0.89
	Great places	0.88
	California dreaming	0.83

Figure 1: Example of pin’s annotation terms. The key observation: a standalone annotation system may contain many synonyms, e.g., “beautiful / favorite / travel / great” places, highlighted by blue rectangle. The one highlighted by a pink rectangle is a wrong annotation prediction. Using a simple average of annotation terms will make the final pin level embedding biased by the term “places”.

The solutions of these two limitations depend on the specific problems. For example, if the object is a sequence like a natural language sentence, then probably word order is important. If the object is a collection, the learned embedding should be independent of the text orders inside the collection. As a visual discovery platform, it is not uncommon that pins do not have direct text information associated with it. We rely on an annotation system¹ which predicts a set of annotation terms describing the content of the pin. Figure 1 gives a real example from our annotation model predictions. The annotation term dictionary is constructed in a data-driven approach and each term usually contains less than 5 words. In order to derive the pin level embedding, the unique “set of short phrases” characteristic makes existing natural language process (NLP) models like BERT [6] not necessarily optimal. Moreover, we hope to use the learned embedding for retrieval purpose in search or related pin surface on the fly. The heavy computation cost of BERT makes it prohibitive to deploy in a production environment.

For retrieval purposes, BERT still has the advantage of capturing semantics better while providing smooth control over the number of candidate documents compared with traditional keyword matching based information retrieval systems. However, it is not trivial to adopt BERT based models directly for large-scale retrieval systems as they are usually designed for NLP tasks that take sequential input data. For example, the fact that annotations of a pin is a set instead of a sequence implies there is no clear grammar dependency between different annotations. Useful tricks like position encoding will not apply here. Empirically, we also find that the representation generated by BERT may not work well for direct cosine similarity.

¹<https://medium.com/pinterest-engineering/understanding-pins-through-keyword-extraction-40cf94214c18>

In this paper, we propose an attention-based algorithm to combine annotation level embeddings into pin level embedding, inspired by the key ideas in embedding language model (ELMo) [24] and hierarchical attention networks (HAN) [38]. Specifically, at the lower layer, we employ bi-directional recurrent neural networks or phrase-level PinText V1 to model the per annotation embedding. At the higher layer, we use attention score weighted average of annotation embeddings as the per pin embedding. We use related pin engagement to construct the learning objective function: the similarity between an engaged {subject pin, candidate pin} pair should be greater than a random negative pair. As shown in figure 2, a candidate pin is positive if it shares the inherently same concept as the subject pin to the user, i.e., Balboa park building. Pinners’ operations on candidate pins are essentially voting for such common concepts. Our motivation is that a good pin embedding is supposed to capture these concepts. We derive pin level embedding in a principled way by paying attention to annotations when optimizing pin to pin relationship and make the embedding depending on the context.

In addition to this related pin specific embeddings, we also improved PinText V1 by using more training data and covering phrases directly. As a result, we are able to evaluate them to answer two interesting questions:

- Is attentive combination better than a simple average when transferred in a different application?
- Is end-to-end embedding in a specific application better than a pre-trained static embedding?

We present the algorithms in section 3, the implementations in section 4, followed by experiments in section 5.

2 RELATED WORK

We refer readers to the related work section in [40] because they are still closely related to the solutions in this paper. In this section, we focus on analyzing the natural language processing models in recent years, which serves as the foundation of this paper and explain briefly the design decision of PinText V2.

The neural language model by Bengio et al. [2] is a seminal work about neural networks based NLP models, where each word can be represented in a vector space and feed-forward layers can be put on top for solving a particular NLP problem. After word2vec [21] is proposed, word embedding gradually becomes more of a standalone model rather than a module in a language model. Word2vec essentially learns the embedding of a word by reconstructing the words in an adjacent context window (either skip-gram or continuous bag of words). It achieves great success because it captures word co-occurrence information very well. Researchers have proposed a variety of word embedding algorithms to improve it in one way or another, including Glove [23], fastText [11], conceptNet [32], etc. These algorithms are usually unsupervised. StarSpace [36] takes a different approach to train embedding in a supervised way such that positive or negative pairwise relationships can be encoded by a simple similarity function in the embedded space. Because we have a huge amount of pinner engagement as supervised information, we train PinText V1 similarly in a multitask learning setting.

Pre-trained word embeddings are usually trained on a much larger training set than the training set available in a particular

application. It has the great advantages to transfer the knowledge encoded in the embeddings for reduced fine-tuning time and relatively good model performance. However, one drawback of these pre-trained word embeddings is that they are static and cannot change with context. For example, "banana" is a fruit, but when it appears with "banana republic", it becomes part of a fashion brand. Researchers proposed techniques to make each word embedding depending on the sequence it appears, e.g., CoVe [19], ULMFiT [27], ELMo [24], although these models are not designed for embedding only. In particular, the Transformer [33] model using the multi-head self-attention mechanism achieves very good performance in sequence to sequence NLP tasks. Based on the encoder part in the transformer architecture as the feature extractor, GPT [25, 26] / BERT [6] / MT-DNN [17] / XLNet [37] models pre-trained on large-scale data refresh all the NLP benchmarks. We do not use these models directly for two reasons: 1) our text input is short concrete phrases instead of long sentences; 2) there is no sequential information between phrases. Our text data is barely natural language. Another blocker for using transformer-based models after BERT in production is the heavy computation cost. It is probably feasible to run BERT offline. However, it is totally prohibitive to infer embedding online. There are some recent works focusing on making BERT more lightweight [10, 14, 15, 28, 31]. We will not elaborate on their details because we do not use Transformers.

However, the great success of the two-phase learning of these modern NLP models is very inspirational. That is one of the design motivations of PinText: we hope to train a text embedding model encoding most of the supervised information, then fine-tune it on a specific application. Therefore, transfer learning [18, 22] is a closely related branch. One important factor that has not been widely studied along embedding transfer is the maintenance cost [29] when multiple embeddings either on the same type or different types of entities are available. The critical point is whether the transferred embeddings are in the same space as the original embedding, such that entity similarities can be measured directly after the transfer.

To this end, we proposed to learn the attention scores [1] for a weighted combination of a bag of terms without changing the pre-trained term embeddings, inspired by the hierarchical attention networks [38]. In contrast, we also propose to learn the embeddings directly in a related pin classification task in the spirit of AutoML [8]. We examine their performance and cost in section 5.

3 ALGORITHMS

Inspired by the recent NLP models, in particular, ELMo [24], HAN [38], and Transformer [33], we propose the supervised pin-level embedding algorithms in this section. It is a simplified version of HAN tailored for our problem.

3.1 Overall Model Design

At a high level, we learn the text embeddings from related pin pairs. Please refer to figure 2 for the illustration. Each related pin pair involves a subject pin and a candidate pin. Based on the backend logging system, we are able to build a database of positive pairs according to users' engagement. Then we can sample a collection of training data $\{z_i, z_{ij}^+, z_{ijk}^-\}$, where j is the index of positive candidate pins and k is the index of negative candidate pins for a particular

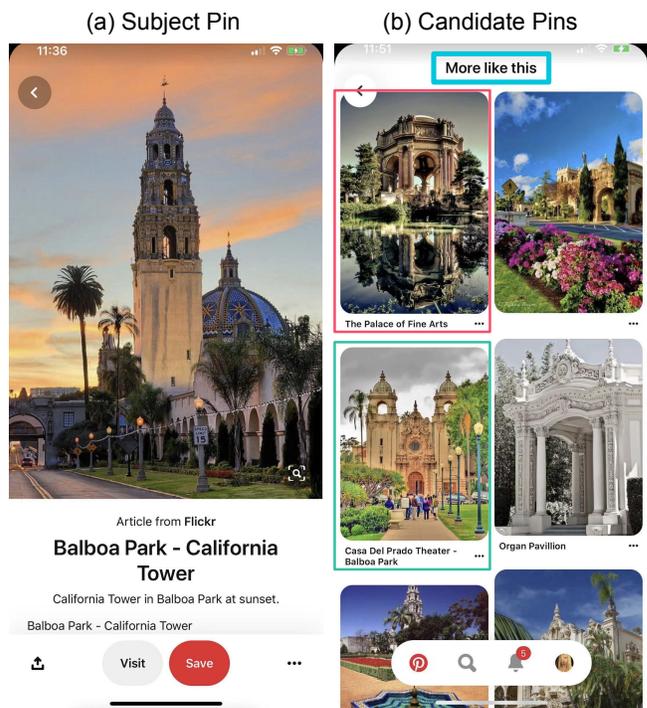


Figure 2: Illustration of related pins. When a pinner clicks a subject pin, a set of ranked candidate pins is presented to the pinner with this "more like this" surface. The highlighted pin by blue rectangle is a frequently engaged candidate pin (save / repin / click etc). The highlighted pin by pink rectangle is a pin from San Francisco instead of Balboa park. Thus these three pins form a triplet to help the best annotation combination.

subject pin index i . The learning goal is to make the similarity between positive pairs bigger than negative pairs in the embedding space. Specifically, we calculate the pin level embedding $z_i := H(P_i)$ as an attention weighted sum of the annotation level embedding $H(A)$.

In PinText V1, we use average of word embeddings as either annotation or pin embeddings. In this paper, we hope to study if 1) learning annotation weight like attention score is better than simple average, 2) learning end-to-end embedding is better than pre-trained embeddings. We refer audience to [40] for better understanding the Pinterest surfaces.

3.2 Pin Level Encoder

Annotation Encoder. In a transfer learning setting, we simply employ a pre-trained text embedding model to generate the annotation embedding. Section 4.1 introduces such a strong baseline that can cover annotation embeddings well. We focus on an end-to-end embedding model in this section. Given an annotation $A = [w_1, \dots, w_K]$, we use a bi-directional recurrent neural networks to encode an annotation. Specifically, we use Gated Recurrent Units (GRU) [4] instead of Long-Short Term Memory units (LSTM) [13] because we found it's simpler and sometimes results in better performance. For each word, we model it with a forward GRU \vec{h} reading the words from w_1 to w_K and a backward GRU \overleftarrow{h} reading from w_K to w_1 :

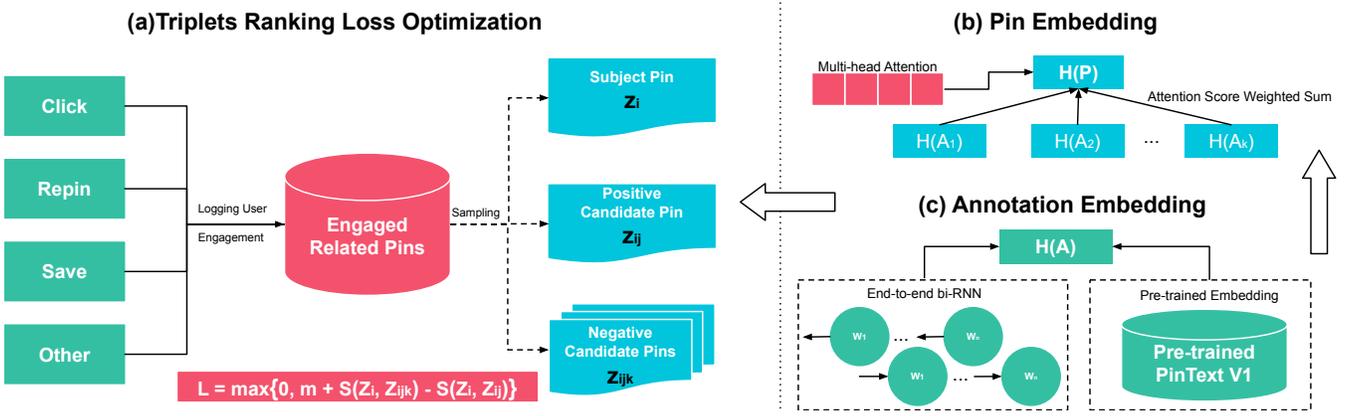


Figure 3: Overall design of end-to-end pin level text embedding. Figure a: we build related pin pair database from various user engagements. We sample training data this way: for each subject pin z_i and a positive (engaged) candidate pin z_{ij} , we sample a collection of negative pins z_{ijk} . Figure b: pin level text embedding $H(P)$ is the attention score weighted annotation embedding $H(A)$. Figure c: Annotation embedding $H(A)$ is from PinText V1, or RNN learned end-to-end.

$$\begin{aligned} \overrightarrow{h(w_i)} &= \overrightarrow{GRU}(x_i), i \in [1, K], \\ \overleftarrow{h(w_i)} &= \overleftarrow{GRU}(x_i), i \in [1, K]. \end{aligned} \quad (1)$$

Here x_i is the d -dimensional embedding vector of word w_i , which is going to be learned end-to-end with parameters at higher layers. The annotation level embedding vector would be the concatenation of the last word:

$$h(A) = [\overrightarrow{h(w_K)}, \overleftarrow{h(w_K)}]. \quad (2)$$

Pay attention to the fact that each annotation has three words on average. This short-length input significantly limits the gain of more complex models. We do care inference time complexity too. That's the major reason why we do not use Transformer / BERT [6] and why we do not add attention at the word level as in HAN [38].

Annotation Attention. Given a pin $P = \{A_1, \dots, A_T\}$, after we get the vector representation $h(A)$ at the annotation level, we introduce the attention mechanism to help identify the annotations that are important for the overall meaning of the pin. For each annotation A_i , we map its embedding to a key vector k_i . Assuming there is a global pin-level query vector q , we have the attention score α_i and attentive annotation aggregation $h(P)$ defined as:

$$\begin{aligned} k_i &= \tanh(W h(A_i) + b), i \in [1, T], \\ \alpha_i &= \frac{\exp(k_i^T q / \sqrt{d})}{\sum_i \exp(k_i^T q / \sqrt{d})}, \\ h(P; W, b, q) &= \sum_i \alpha_i h(A_i), \end{aligned} \quad (3)$$

where d is the dimensionality of embedding vectors. We can compare the difference between this attentive aggregation and average over word embeddings: 1) each annotation embedding $h(A)$ is an RNN respecting word order information instead of a simple lookup in a pre-trained embedding table. This is particularly useful when unseen annotation terms emerge because they cannot be covered by the pre-trained phrase embeddings directly, and the averaging of word embeddings would lose the word order information; 2) each

pin embedding $h(P)$ is a weighted sum of annotation embeddings. The weight is essentially the attention score that helps capture which annotations are useful for the pin's semantics. They are supposed to show an advantage over the simple average.

Multi-head Attention. Each pin is polymorphic in nature, which means that the best representation may not be fixed in different contexts. Therefore, we employ the multi-head attention method as in the Transformer [33] model to learn multiple annotation combinations. Given H attention head, we have the concatenated multiple subspace representation as

$$h(P) = \left[\sum_i \alpha_i^n h(A_i) \right], \text{ where } n \in \{1, \dots, H\}, \quad (4)$$

where $[]$ is the vector concatenation operation. From the optimization perspective, multi-head attention mechanism uses more parameters to fit data than single-head attention, thus it has a chance to produce better empirical results.

3.3 Related Pin Classification

With the pin level's representation, we learn all the model parameters in a related pin classification task. Here is the tradeoff between classification precision and retrieval efficiency plays a critical role in the learning paradigm design. On the precision side, we can put multiple dense layers to squeeze out the best classification model when large amounts of labeled training data is available. On the retrieval efficiency side, in general, a retrieval system consists of multiple layers of rankers. It is totally feasible to apply complex models to the last re-ranking layers where the ranking candidate set usually contains hundreds of pins. However, there are billions of pins on the first layer to be retrieved and ranked. We must be able to build index mapping query vectors to candidate pins because we need to retrieve rank billions of pins in realtime.

Therefore, we simply use margin rank loss as in PinText V1 [40] to encode classification model into embedding vector similarity. Let $z_x := h(P_x)$ denote the pin level vector, we have the retrieval based

Table 1: Covered annotation terms by text embedding model across some top languages.

	EN	FR	DE	ES	PT	JA
CptNet [32]	98.3%	99.08%	91.09%	95.07%	96.91%	84.03%
PinText	100.0%	100.0%	99.98%	100.0%	100.0%	100.0%

objective function as:

$$J = \min \sum_{i=0}^L \sum_{j=0}^M \sum_{k=0}^N \max \left(0, \mu - (S(z_i, z_{ij}^+) - S(z_i, z_{ijk}^-)) \right), \quad (5)$$

Where we use cosine similarity as the similarity measure S . For each pin P_i , we have up to M positive related pins P_{ij}^+ . For each positive related pin pair $\{P_i, P_{ij}^+\}_{j=1}^M$, we have N randomly sampled background pair $\{P_i, P_{ijk}^-\}_{k=1}^N$. μ is the margin between positive and negative pairs.

4 IMPLEMENTATIONS

We present some implementation details in this section. First we describe some practical approaches that can improve PinText V1 significantly, then we describe how to handle large-scale embedding training.

4.1 Improved Pre-trained Baseline

The supervised multitask word embedding PinText V1 is a pretty strong baseline [40]. With the drawback of simple word embedding may lose useful word order information, it also has the advantage of ready-to-use for any cold start problem. This is essentially knowledge transfer from billion-level user voting for the semantic relationship between two entities. Therefore, we try to improve V1 as the baseline such that it still solves the problem if a downstream user does not have the data or resource to train their application-specific text embeddings.

As the major drawback of V1 is order-unawareness, we propose two major improvements:

- enumerate the word n-grams of input text instead of using word only;
- build a phrase dictionary by detecting valid n-grams in a data-driven approach.

In a nutshell, we learn pre-trained embeddings of not only words but also phrases at the training phase. We check all n-grams of input text against a phrase dictionary and use their mean embeddings as input text embedding at the inference phase. The learning and inference tech stack is the same as PinText V1.

The core of this improvement is to build the covered phrase embeddings and treat them as a single word. We respect the word orders in this way because different word orders form different phrases. We use a data-driven approach to identify phrases that have the best semantic meaning and have the most business impact. We end up with phrases in the following categories:

Search Queries. We rank search queries on Pinterest in the past two years in the descending order of search frequency. Then we merge the top query list to the phrase dictionary.

Ads Keywords. The advertisers on Pinterest platform are able to upload the keywords of their ads for targeting purposes. Those

keywords are often of high quality. They also have a significant impact on business goals.

Phrases in Pre-trained Embeddings. There are valid phrases in some open-sourced text embedding models. In particular, we add all the phrases in ConceptNet [32] to our dictionary.

Annotation Terms. We have an internal annotation system [34] tagging the concepts in pins. They are also the text input of pins in this paper. All canonical annotation terms would be valid phrases. Note, this means all annotations are available in the learned embedding dictionary. A simple lookup operation generates annotation level embedding. That’s the major reason that we are confident the improved PinText V1 is a strong baseline to beat given the near 100% coverage of Annotation terms.

The above sources lead to about 8.5 million phrases. We get 18.7 million phrases and words, which has about 9 million non-English tokens (a token is a word or a phrase). It is very important to have tokens of different languages embedding in the same space such that we can handle all languages uniformly in downstream applications. The PinText algorithm is language independent. If the pair has both English and non-English tokens appear frequently enough, the rank margin loss minimization will learn the cross-language similarity automatically. This improved PinText baseline model is good enough to replace other open-sourced models. Table 1 presents the annotation coverage of some top languages. Empirically, we found that embeddings of Asian language words have worse than expected multilingual performance. But the other major languages are reasonably good. We have successfully deployed it into our ads retrieval/ranking systems.

4.2 Single Instance Training

In a typical setting, the learned embedding dictionary contains about 10+ million tokens and each token has a 64 dimensional real vector. Note that there are only a few thousands of English words used frequently. The coverage at this scale takes about 10 Gigabytes on disk. So such an embedding model can be fully hosted in memory. Suppose there are 10+ million positive training pairs, training can finish in hours with a modern workstation. In reality, dev’s time is way more expensive than the machine’s time or the machine’s hardware cost. We argue that single instance training with all training data should handle most of the cases. It should be the preferred way to a complex distributed setting.

When the number of training pairs goes up to billion+ level, it is not good to load them in one pass and let training go. Any interruptions would lead to either a corrupted model or an orphaned model having nowhere to continue. We find a practical way to handle billion+ level training: we first shuffle the training data and split it into 100 partitions of about the same size. Then we train the embedding sequentially. The trick is to use the output of the last partition as the warm start of the current partition. For the first partition, we need to collect all the tokens covered in training data and initialize their embeddings by random vectors. One can prove that the model in this way has the same quality as the model using the full training data in one run. For each positive pair, we sample the negative pins randomly within a batch of training pairs. This batch size is much smaller than the partition size (in the 10+ million scales). Therefore, gradient updates are the same because

the current batch for deriving gradients is essentially the same. It also provides the opportunity to evaluate model quality per batch, which is important for controlling model quality and stopping criterion.

4.3 Asynchronous Distributed Training

Another choice is to train the embedding in a distributed manner. Usually we categorize distributed training into data parallelism and model parallelism. In the former case, each training worker holds the whole model and uses a portion of training data to calculate gradients. In the latter case, the model is so huge that it cannot be handled by a single worker. One has to employ multiple nodes to hold the whole model architecture. As we mentioned in section 4.1, we have a model containing the embedding vectors of less than 20M tokens, which can be held in memory by a modern server. However, the number of training instances is easily above billions, or the physical size on disk easily goes up to a few terabytes. Therefore, data parallelism can be very helpful if there is a dedicated cluster available. We employ Kubeflow² as the framework for distributed training, where a parameter server (PS) holds the whole model and a couple of workers communicating with PS to exchange gradients. It is essentially an asynchronous stochastic gradient descent (SGD) algorithm [5]. There are many more recent works that make distributed SGD better (e.g., [20, 35]). But we argue that distribution itself is the most significant factor in this PinText context, instead of details on gradient calculation or communication, given our model is relatively simple.

5 APPLICATIONS AND EXPERIMENTS

We report some empirical results and discoveries in this section.

5.1 Experiments Setup

Aligning to section 3, we use cosine similarity to rank candidate pins w.r.t. a subject pin in the related pin ranking task. The solutions we have evaluated include:

- (i) **Raw Text:** rank based on traditional text-matching based retrieval. We use the number of matched annotation terms as the ranking score;
- (ii) **Concept Net:** among the open-sourced word embeddings, we found Concept Net usually has the best performance for our text similarity scenario [32];
- (iii) **PinText V1:** the average of annotation term embeddings. When annotation does not appear in the embedding dictionary, use the average of word embedding as annotation embedding;
- (iv) **PinText V2 Transfer:** fix the annotation embedding to be the results of improved V1 in section 4.1, but learn the multi-head attentions for a better pin-level annotation combination;
- (v) **PinText V2 End2End:** we learn both the annotation embedding and the pin-level combination in an end-to-end manner.

We evaluate three different types of candidate pins:

- **Organic Pins:** The most general cases where candidate pin could be any pin on the platform;

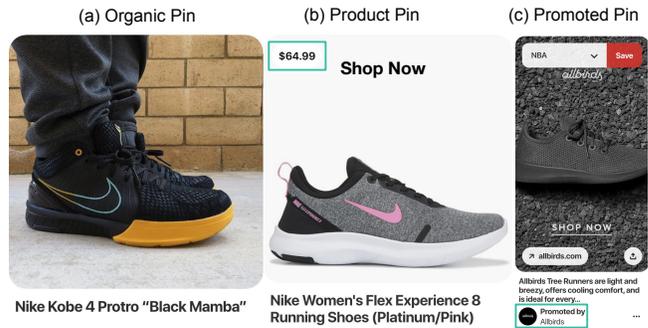


Figure 4: Examples of three different types of candidate pins. They have very different metadata and serving traffic, and different business impacts, although all three pins are about shoes.

Table 2: Three related pin dataset size in experiments labeled by vendors.

Dataset	#Pairs	#TotalUniquePin	#AvgCandidatePin
Organic	22,396	24,519	9.95
Product	7,157	3,045	8.79
Promoted	21,244	3,392	14.75

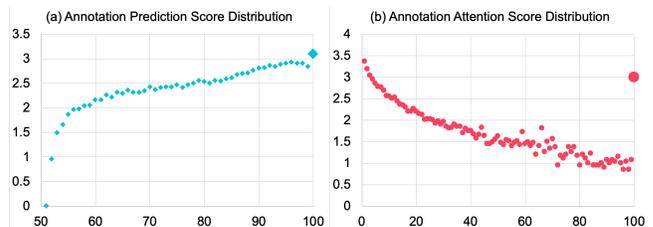


Figure 5: Annotation score distribution. X-Axis: normalized score s by $\text{int}(s/\max(s)*100)$ per pin. Y-Axis: log-scale normalized score frequency in 1K randomly sampled pins. Figure a: annotation prediction score from a GBDT model. Figure b: learned attention score.

- **Product Pins:** The candidate pin is about a product provided by partners that is likely to be purchased offline;
- **Promoted Pins:** The candidate pin is essentially an ads generated by advertisers.

For a given subject pin, we rank its collection of candidate pins and evaluate the average precision@K. Those three types of pins have very different metadata. For example, a promoted pin (a.k.a. ads) usually have a clear text description to attract pinners click on it. But general organic pins do not necessarily have text descriptions. Product pins and promoted pins also have different triggering mechanisms to be distributed from organic pins. Therefore, it is better to evaluate different types of pins to eliminate algorithm bias as much as possible. Table 2 presents the labeled dataset size of three types of pins.

We employ vendors to label the relevance degree for each related pin pair. We always have an odd number of labels and use majority voting to decide whether it is positive or negative. We use the idea of active learning [30] to sample the candidate pins, because we hope to avoid too easy cases where embedding algorithm itself may not even be needed. Given a related pin ranking model and the final utility boosting logics in our system, we select candidates

²<https://www.kubeflow.org/>

Table 3: P@K for organic pin ranking based on embedding similarity.

Model	K=1	K=2	K=3	K=4	K=5
Raw Text	48.99%	45.93%	44.63%	43.07%	41.13%
Concept Net	50.63%	48.44%	46.06%	44.01%	41.48%
PinText V1	51.83%	48.86%	46.31%	44.45%	42.08%
V2 Transfer	78.52%	70.93%	65.02%	58.98%	53.29%
V2 End2End	51.65%	48.22%	45.61%	43.35%	41.56%

Table 4: P@K for product pin ranking based on embedding similarity.

Model	K=1	K=2	K=3	K=4	K=5
Raw Text	67.90%	66.91%	65.43%	61.60%	58.76%
Concept Net	68.88%	67.28%	64.77%	62.46%	60.00%
PinText V1	71.36%	68.52%	66.50%	63.40%	60.40%
V2 Transfer	74.07%	70.49%	68.64%	65.74%	62.91%
V2 End2End	74.07%	69.87%	66.83%	63.46%	60.99%

Table 5: P@K for promoted pin ranking based on embedding similarity.

Model	K=1	K=2	K=3	K=4	K=5
Raw Text	53.15%	50.27%	47.34%	44.91%	42.72%
Concept Net	52.74%	51.37%	48.85%	46.29%	43.84%
PinText V1	53.98%	52.40%	50.14%	47.28%	44.81%
V2 Transfer	56.60%	53.43%	51.05%	48.04%	45.71%
V2 End2End	55.91%	52.74%	50.27%	47.42%	44.97%

with model predicted scores below than a threshold and its final ranked position bigger than a threshold. The relevance of those lower ranking pins is essentially harder to be defined.

The hyperparameters we used in these experiments: embedding dimension $d = 64$, attention head number $H = 4$, max sampled positive candidate number $M = 10$, max sampled negative candidate number $N = 50$, the rank loss margin $\mu = 0.5$, starting learning rate is 0.01. We use Adam optimizer and Tensorflow implementation.

5.2 Attention is Better than Simple Average

The top-k ranking precision is presented in table 3, 4, and 5. First of all, there is a clear trend across all types of pins about the performance of the evaluated methods: (iv) > (iii) > (ii) > (i). We draw the following conclusions about this performance order:

The attentive bag-of-annotation combination is better than simple average of word embeddings. It shows 3+% to 10+% P@1 gains. This verifies our design motivation exactly. It is not uncommon that a higher level entity is composed of a collection of lower level entities. In this particular case, the higher-level entity is a pin and the lower level entity is an annotation term. As shown in figure 1, the annotation have two limitations: a single annotation can have multiple synonyms that may dominate the pins text information; some annotation is simply not related to the pin’s content. Learning the attention scores as weights gives the chance to decide the correctness or relevance of the annotation terms. As a result, it always produces better precision than non-attention solutions.

One straightforward way to combine annotation embeddings is to use the annotation prediction score as weights. However, as shown in the example in figure 1, all the annotation scores have a similar distribution. It cannot really distinguish the truth concept

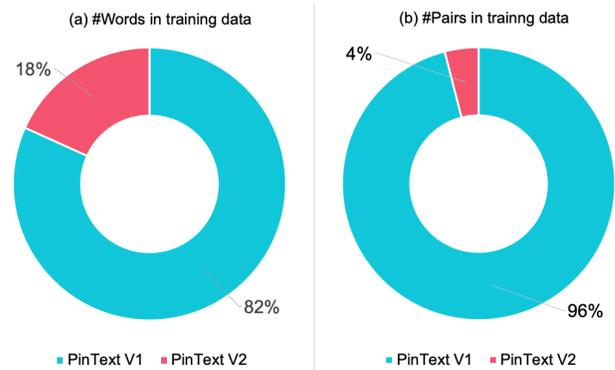


Figure 6: Training data comparison between V1 and V2. For V1, we use large volume of training data from all the surfaces we can collect, which covers all the free text. For V2, we focus on the related pin classification / ranking application, which covers top annotation terms.

Balboa park from general vague concept *Beautiful places*. In order to get deeper insights, we plotted the relative prediction score histogram in figure 5 (a). One can see that the frequency of $s > 80$ takes a big portion. The overall distribution is flat.

Figure 5 (b) plots the relative attention score distribution on the same data setup. It has a clear big gap between the largest score and the second-largest score, which probably means that it captures the most important concept of the pin. It also has a high frequency for the 0 score, which probably means it can single out irrelevant information. Together with the multi-head mechanism, it provides a principled way of combining entities in a bag-of-entity context.

In-house multi-task supervised embedding is better than open source embeddings for internal applications. This has been previously proved in the former version [40] in a text classification setting. We hope to highlight some basics of the PinText advantages: 1) it is supervised; 2) it uses large-scale multiple surfaces’ data; 3) it is multilingual in nature. On the promoted pin dataset, it has only +1.2% gain compared to ConceptNet. Note the fact that we only used English annotation terms in these experiments because our focus is to evaluate attention mechanism and end-to-end embedding. The gain of PinText would be bigger on international text due to its significantly better coverage. The other factor to consider is that we purposely sacrifice some embedding model performance by using a smaller dimension 64, compared to 300 in Concept Net. We found 64 is the minimum number we can take empirically. Reducing it further to 32 dimensions would hurt performance significantly.

Embedding based ranking is better than simple raw text match based ranking. The best embedding based method boosts P@1 by 49% \rightarrow 78%, 68% \rightarrow 74%, and 53% \rightarrow 57%, respectively. This is not surprising because embedding based ranking provides a smooth comparison between text semantics. It leads to the opportunity to match pins when there are no exact text match between them.

5.3 End2End is Not Better than Pre-trained

The second important dimension we measure is that if end-to-end embedding is better than pre-trained embedding. It has tied P@1 74% on the product pin dataset, slightly worse 56.6% \rightarrow 55.9% on the promoted pin dataset, and much worse 78% \rightarrow 51% on the organic dataset.

Before we draw conclusions, we need to carefully study the training setup of Pre-trained and End2end embedding. The truth is that we can never have fair comparison between them on the same training data. Or put the other way: the gain from pre-trained embedding transfer is because that it has the opportunity to use huge volume of possible data, while End2end learning has to be within the scope of the problem it tries to solve. If we restrict pre-trained embedding to the same training data, then probably end2end learning will win, on the condition that 1) the training dataset size is big enough to derive a good model; 2) the training data have identical or very similarity distribution as the inference stage.

In an academic setup, we leave out a portion of the training data to tune hyperparameters (or use cross-validation) and another portion of the data for evaluation purposes. In industry, the evaluation is usually an online service that takes multiple features as inputs. Depending on the input data for the embedding, it may have a distribution that is totally different from the offline collected training data. This is a true dilemma:

It requires a multiple million numbers of training instances, considering the fact that we need to learn the embedding vectors together with the layers in the deep neural network model. Suppose there are 20,000 popular words, then the number of embedding parameters goes up to $20K * 64D > 1M$. We simply cannot afford the labeling cost of multiple million levels of data. Instead, we try to "mock" the relevance between related pins by pinners' engagements. This is a good way to unlock the training data size. However, when we evaluate the model, we care about its performance on the challenging cases, which are labeled by human beings. That is exactly the training/evaluation discrepancy checks in.

Figure 6 compares the training data of PinText V1 and V2. As stated in [40], we use all available engagements from all three surfaces including homefeed, search, and related pin to train V1. Even after reasonably strict filtering, we end up with multiple billion levels of training instances. We have to stay with the particular related pin ranking problem in V2 end2end training. As a result, it only takes less than $< 5\%$ of training dataset size. It also covers only $< 20\%$ of V1 English words because it only cares about canonical annotation tokens. We conclude that the performance gap of end2end and pre-trained embeddings are from training data difference and the discrepancy between training data and evaluation data. We also must stress that there is no golden rule about which one is the best strategy. It is totally dependent on the problems and the resources to build the model.

5.4 Some Applications

Given the techniques we presented so far, we are able to derive the pin level embedding from the text snippets it has. We have deployed PinText to several important productions. We would like to select a few here in the hope of inspiring other practitioners and researchers. One direct impact is text classification tasks. We have many text classification problems at Pinterest, for example, Query2Interest and Pin2Interest [16], which maps an input text snippet to a node in an interest taxonomy [9]. We use FastText a lot as a pure text classifier [3] which learns word embeddings internally. But there are also many cases where we have other crafted non-text features,

where embeddings can be plugged in as inputs. We are able to make those scenarios better by the V2 techniques here.

A second important application is pin level ranking. The V2 approach provides a principled way to generate pin-level embedding given text as inputs. By building indexes for pins, we are able to retrieve the embedding vector in real-time, and use it as input layer of deep neural network-based ranking models. Due to the limited space and the fact that they are not replicable to non-Pinterest engineers, we omit the online A / B experiments in this paper. It shows that such text-based pin embedding provides extremely useful complementary results in addition to other types of embeddings, like visual embedding [39] or GraphSAGE embedding [12].

The V2 transfer setting brings an important feature: the attention score function is nonlinear, but the combination of annotation embeddings is indeed linear. This means the pin embedding lives in the same space as the text embedding model. It avoids embedding projection or other complex techniques to make pin to text comparison possible.

6 CONCLUSIONS

In this paper, we improved PinText V1 by making it include phrases in a pure data-driven approach. Then we proposed PinText V2 that can combine lower entity level embeddings to a single higher-level entity embedding based on attention mechanism. We studied a very interesting and important question about if end2end embedding with limited data can beat pre-trained embedding with a huge volume of training data.

The answer is clear in a related pin ranking problem: in-house text embedding is better than open sourced embedding due to the fact that the former fits our data better; embedding based ranking is better than plain text-matching based ranking due to the fact that the former provides smooth and semantic level similarity calculation; pre-trained embedding can be better than end-2-end embedding when the latter has limited training data and when there is concept drift between evaluation and training data. We have to mention that the conclusions above are context-dependent. It is possible to have end2end embedding better, with application-specific data sampling and model architecture design. Our motivation here is to bring up a general embedding learning strategy.

In future, we plan to work in the following space: first, when there are multiple embeddings available in the system, how to make them work together to squeeze out the best performance. One pain point is that different embeddings are not comparable directly. There is no easy way to project them into the same space without losing quality. Second, technical debt checks in when we try to upgrade embedding versions that are widely used in production. Making sure of auto-regular updates is an essential requirement to keep downstream models up-to-date. Third, avoid end2end embedding learning for a particular application as much as possible, to reduce tech debt and transfer existing knowledge.

ACKNOWLEDGMENTS

We thank ads quality team, visual computation team, and cloud management platform team at Pinterest for the fruitful collaboration and discussion. We thank Chen Chen for the help of collecting related pin data.

REFERENCES

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- [2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* 3 (March 2003), 1137–1155.
- [3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.
- [4] Kyunghyun Cho, Bart van Merriënboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 1724–1734.
- [5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc' Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 1232–1240.
- [6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805
- [7] Stephanie deWet and Jiafan Ou. 2019. Finding Users Who Act Alike: Transfer Learning for Expanding Advertiser Audiences. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. 2251–2259. <https://doi.org/10.1145/3292500.3330714>
- [8] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An Open Source AutoML Benchmark. *CoRR* abs/1907.00909 (2019). arXiv:1907.00909
- [9] Rafael S. Gonçalves, Matthew Horridge, Rui Li, Yu Liu, Mark A. Musen, Csongor I. Nyulas, Evelyn Obamas, Dhananjay Shroutry, and David Temple. 2019. Use of OWL and Semantic Web Technologies at Pinterest. In *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*. 418–435. https://doi.org/10.1007/978-3-030-30796-7_26
- [10] Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing {BERT}: Studying the Effects of Weight Pruning on Transfer Learning. <https://openreview.net/forum?id=SJlPOCEkVH>
- [11] Edouard Grave, Tomas Mikolov, Armand Joulin, and Piotr Bojanowski. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*. 427–431.
- [12] William L. Hamilton, Zitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 1024–1034.
- [13] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.
- [14] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling BERT for Natural Language Understanding. arXiv:cs.CL/1909.10351
- [15] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *CoRR* abs/1909.11942 (2019). arXiv:1909.11942
- [16] Eileen Li. 2019. Understanding pins through keyword extraction. <https://medium.com/pinterest-engineering/pin2interest-a-scalable-system-for-content-classification-41a586675ee7>.
- [17] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-Task Deep Neural Networks for Natural Language Understanding. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. 4487–4496.
- [18] Jie Lu, Vahid Behbood, Peng Hao, Hua Zuo, Shan Xue, and Guangquan Zhang. 2015. Transfer learning using computational intelligence: A survey. *Knowl.-Based Syst.* 80 (2015), 14–23. <https://doi.org/10.1016/j.knsys.2015.01.010>
- [19] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in Translation: Contextualized Word Vectors. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 6294–6305.
- [20] Qi Meng, Wei Chen, Jingcheng Yu, Taifeng Wang, Zhiming Ma, and Tie-Yan Liu. 2016. Asynchronous Accelerated Stochastic Gradient Descent. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. 1853–1859.
- [21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). arXiv:1301.3781 <http://arxiv.org/abs/1301.3781>
- [22] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2010), 1345–1359.
- [23] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar*. 1532–1543.
- [24] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. 2227–2237.
- [25] Alec Radford and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. In *arxiv*.
- [26] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018. Language Models are Unsupervised Multitask Learners. (2018).
- [27] Sebastian Ruder and Jeremy Howard. 2018. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. 328–339.
- [28] Victor Sanh, Lysandre Debut, and Thomas Wolf. 2019. Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT. <https://medium.com/huggingface/distilbert-8cf3380435b5>.
- [29] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Densnison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. 2503–2511.
- [30] Burr Settles. 2012. *Active Learning*. Morgan & Claypool Publishers. <https://doi.org/10.2200/S00429ED1V01Y201207AIM018>
- [31] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. arXiv:cs.CL/1909.05840
- [32] Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. 4444–4451.
- [33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 6000–6010.
- [34] Heath Vinicombe. 2019. Understanding Pins through keyword extraction. <https://medium.com/pinterest-engineering/understanding-pins-through-keyword-extraction-40cf94214c18>.
- [35] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 1509–1519.
- [36] Ledell Yu Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. 2018. StarSpace: Embed All The Things!. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. 5569–5577.
- [37] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *CoRR* abs/1906.08237 (2019).
- [38] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. 1480–1489.
- [39] Andrew Zhai, Hao-Yu Wu, Eric Tzeng, Dong Huk Park, and Charles Rosenberg. 2019. Learning a Unified Embedding for Visual Search at Pinterest. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. 2412–2420. <https://doi.org/10.1145/3292500.3330739>
- [40] Jinfeng Zhuang and Yu Liu. 2019. PinText: A Multitask Text Embedding System in Pinterest. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. 2653–2661. <https://doi.org/10.1145/3292500.3330671>