# Improving Taxonomy-based Categorization with Categorical Graph Neural Networks

Tianchuan Du*
*Microsoft*
Sunnyvale, USA
Tianchuan.Du@microsoft.com

Keng-hao Chang
*Microsoft*
Sunnyvale, USA
kenchan@microsoft.com

Paul Liu*†
*Stanford University*
Stanford, USA
paul.liu@stanford.edu

Ruofei Zhang
*Microsoft*
Sunnyvale, USA
bzhang@microsoft.com

***Abstract*—In search and retrieval, a critical subtask is the classification of user search queries into predefined categories. Traditional supervised multi-class classification algorithms usually treat each category independently. In practical applications however, the categories have implicit relationships. Categories are organized as a tree-based taxonomy, which can be viewed as a graph. In this work, we explore a novel and systematic way of leveraging semantic information for improving taxonomy-based categorization. We propose a class of graph-based network structures, which we call Categorical Graph Neural Networks (CaGNN). CaGNNs leverage relationship information between neighbor categories and overlay the semantic information for each category, thus improving the performance of query categorization. The CaGNN framework can integrate a baseline categorizer with any Graph Neural Network, such as the commonly used Graph Attention Network and Graph Convolutional Network. Over a query categorization dataset of 2k categories and another ad title categorization dataset of 5k categories, CaGNN improves categorizers' performance significantly compared to a baseline Deep Neural Network model without the CaGNN structure. Notably top 3 prediction recall increases from 90.15% to 91.40% for the ad title categorization task, for which is quite significant at over 90% level for more than 5k categories. By inspecting the learned category embeddings and the flow of message passing, we show that CaGNN effectively encapsulates useful graph structural information. Online A/B testing result shows that an ad ranking model with CaGNN-based features has increased ad click-through rate by 1.81% and reduced defect rate by 2.64%. The model has been deployed to production.**

***Index Terms*—graph neural networks, deep learning, query classification, category embedding**

## I. Introduction

Search engine advertising has become a significant element of the web browsing experience. Search engines derive a large portion of their revenue from showing advertisements along with search results. A key task here is to assign a search query, an ad, or a web document to a predefined category based on its topics. This is essentially a multi-class classification (categorization) problem. If one can achieve accurate classification, one can improve the ad selection quality matching user intent, thus increasing the ad click probability and revenue. One can also achieve proper user targeting by summarizing user search history via the user's prior query category distribution.

* Corresponding authors.
† Part of this work was done during the author's internship at Microsoft.

Query classification has been widely studied with a variety of different methods [1]–[5]. These methods have built upon traditional supervised classifiers by leveraging semantic information, web results, and additional mined sources of information. However, traditional multi-class classification algorithms treat each category independently, without taking the relationship between categories into consideration. In applications, categories often have implicit or explicit relationships, and are organized by human editors through a taxonomy with root categories, inner categories and leaf categories. For example, a product ad search engine sometimes has to classify as many as five thousand product categories [6]. One example category is *Animals & Pet Supplies > Pet Supplies > Bird Supplies*, which represents a path from the root category, *Animals & Pet Supplies* to a leaf category, *Bird Supplies*. Intuitively, if one can model the relationship or correlation between categories, the performance of the categorizer should improve. Another constraint is that queries must be categorized into the strictest possible category. Although any *Bird Supply* product is also a *Pet Supply*, the most correct and most useful label is the one from the deepest category that the query fits in (relative to the taxonomy).

With the recent advances of Graph Neural Networks (GNN), several methods have been proposed for modeling category relationships. Perhaps the most relevant to our approach are the works of Lanchatin et al. [7] and Wang et al. [8]. Lanchatin et al. proposed Label Message Passing (LaMP) Neural Networks for multi-label classification, which treats categories as nodes on a category-interaction graph. Wang et al. [8] use semantic embeddings and knowledge graphs of categories to achieve zero-shot categorization with a Graph Convolutional Network (GCN). We discuss these works and related ones in depth in Section II. It is clear from prior work that leveraging both semantic and structural information is helpful for categorization.

### A. Our contribution

Motivated by these works, we propose a way to leverage the taxonomic and semantic information of ad category taxonomies via a class of graph-based network structures. We call these structures Categorical Graph Neural Networks (CaGNNs). CaGNNs leverage both semantic and structural relationship information between categories to improve classification accuracy. Our CaGNN algorithms can leverage any Graph Neural Network framework, such as the Graph Attention Network and

Graph Convolutional Network, to utilize structural information between target categories and integrate them with other baseline network structures. We summarize our contributions below:

- We apply our proposed CaGNN models to product ads query categorization. The result shows that CaGNNs improve the performance of a baseline model without GNN structure significantly.
- We test the CaGNN method across different production regimes, from query categorization to ad title categorization. Each regime has its own category taxonomy. The results show that our CaGNN model improves across all datasets.
- We provide a way to use the attention mechanism of a Graph Attention Network [9] within our CaGNN framework to inspect the learned embeddings and the flow of message passing.
- We show that CaGNNs produces meaningful standalone category embeddings that can be *leveraged in production*.

## II. RELATED WORKS

### A. Graph Neural Networks

In the past few years, graph-based learning algorithms have received much attention within the machine learning community. Much of this work revolves around so called Graph Neural Networks (GNNs), for which comprehensive surveys can be found in [9]–[11]. With various GNNs proposed, different papers have created varying taxonomies to categorize existing GNNs. Wu et al. [10] proposed a taxonomy to divide the state-of-the-art graph neural networks into four categories, namely recurrent graph neural networks, convolutional graph neural networks, graph autoencoders, and spatial-temporal graph neural networks. Zhou et al. [9], [12] propose a different taxonomy, instead categorizing by graph types, propagation types, and training methods. Most of the existing literature involving GNNs are about node classification or graph embeddings. In this work however, we use GNNs for the purpose of text classification. Contrary to traditional GNN applications, the texts are not fixed as nodes in our graph, but instead as arbitrary input. In our architecture, traditional deep neural network architectures are combined with GNNs to leverage both the taxonomic graph data of the categories as well as the semantic data from the queries. In this paper we mainly discuss two commonly used GNN types based on the propagation types proposed in [9]: the graph convolutional network (GCN), and the graph attention network (GAT).

### B. Graph Convolutional Networks

A graph convolutional layer encapsulates each node's hidden representation by aggregating feature information from its neighbors. After feature aggregation, a non-linear transformation is applied to the outputs. By stacking multiple layers, the final hidden representation of each node receives messages from a further neighborhood as a node embedding [10], [13]. This formulation was originally proposed by Kipf et al. [13], who used GCNs for graph-based semi-supervised learning. In a number of experiments on citation networks and on a knowledge graph dataset, GCNs outperform previous state-of-the-art methods by a significant margin. Over the past three years, many improvements for GCN-based architectures have been proposed [10], [14], [15]. It can be applied to the problem of classifying nodes in a graph, generating node embeddings, generating graph embeddings, as well as NLP-type applications [16]–[18]. However, as the GCN is based on a spectral approach, the learned filters depend on the eigenbasis of the underlying Laplacian, which depends on the whole graph structure. Thus, a model trained on a specific graph structure cannot be directly applied to a graph with a different structure [12]. Furthermore, the theory behind the original GCN is based on undirected graphs, whereas many graphs are directed in nature.

### C. Graph Attention Networks

The attention mechanism has been successfully used in many machine learning tasks such as machine translation, machine reading, and sequence classification [9]. In the context of GNNs, Veličković et al. [12] proposed a graph attention network (GAT) which incorporates the attention mechanism into the propagation step of a GNN. It computes the hidden states of each node by attending over its neighbors, following a self-attention strategy. Furthermore, they define a single graph attentional layer and construct arbitrary graph attention networks by stacking the GAT layer. By stacking layers, nodes are able to attend over their further neighborhood's features. Critically, GATs are able to specify different weights to different nodes in a neighborhood, without knowing the graph structure upfront. It addresses several key challenges of spectral-based graph neural networks and makes the model readily applicable to inductive as well as transductive problems. Further information can be found in Lee et al. [19], who reviewed different attention models applying to both homogeneous graphs and heterogeneous graphs.

### D. Using Category Structural Information

There is a wide variety of approaches that utilize the structural information between categories during classification. Some encode the graph information to the loss function by an explicit graph-based regularization, e.g. by using a graph Laplacian regularization term [20], [21]. Others learn hierarchical representations by embedding them into hyperbolic space [22]. However, those methods could not leverage other features from categories except graph information, like semantic features and hand-craft features in this paper. In the context of GNNs, Lanchantin et al. [7] proposed Label Message Passing (LaMP) Neural Networks for multi-label classification, which treats categories as nodes on a category-interaction graph. They then compute the hidden representation of each label node conditioned on the input using attention-based neural message passing. One limitation of this method is that it cannot generate category embeddings, because the category interaction is conditioned on the input. Wang et al. [8] use semantic embeddings and graphs of categories to achieve zero-shot categorization with GCNs in a visual classification problem.

This approach requires two separate steps of learning. First, they learn the classifiers of seen categories, then they use GCNs to learn the classifiers of unseen categories from the seen categories. Instead, our approach trains label embedding and query categorizer together in one shot, under a supervised learning setting.

## III. METHOD

In our setting, we are given a dataset with $n$ items:

$$(X, Y) = \{(x_i, y_i)\}_{i=1}^{n}, \tag{1}$$

where $x_i$ represents the textual string for item $i$ and $y_i \in \{1, \ldots, C\}$ represents its label, and $C$ is the total number of categories. The category taxonomy can be viewed as a collection of tree structures (i.e. a forest), starting with root categories, inner categories and then leaf categories. Note that the label $y_i$ can be any category, including the root categories. Given an input $x$, we want to learn a mapping function $f(x)$ to predict its target, $y$. We use different neural network models to approximate this mapping function and evaluate their performance. The architecture of our different experimental models is summarized and illustrated in Figure 1. In total we explore variations of five different models, represented by various paths through Figure 1 from left to right. We describe each path in detail below. All the models are trained with the same categorical cross-entropy loss function. Data preprocessing varied from dataset to dataset and is described in Section IV-A. We evaluated our method on two different internal Microsoft datasets over different category taxonomies. They are described in Sections IV-A1 and IV-A2.

### A. Baseline DNN Model

Our baseline model is a fully-connected deep neural network (DNN) with 3 hidden layers of size 2048 plus an output layer with size of output dimension (number of categories, $C$) and $softmax$ activation. The loss functions for all models in this paper are the same categorical cross entropy loss. The learning procedure is the same for all models with Adam optimization algorithm. The input to baseline network is the FastText embedding [23] features of input as mentioned in Section IV-A. It is Path ① in the network architecture in Figure 1. We add batch normalization and dropout layer after each fully-connected layer. The dropout rate is 0.3. For all hidden layer activation functions, we apply LeakyReLU [24], [25] with a negative slope of 0.3. The activation function can be replaced with different activation functions. The reason we picked the DNN model as baseline is because it corresponds to what is used in our production for Bing product ads query categorization. We use an in-house Convolutional Neural Network (CNN) model to generate query embedding first. The CNN model is similar to what is described in [26]. Then the query embedding is followed by a fully-connected DNN to make the final prediction (parameters are optimized for production). Due to the CNN model to generate the query embedding is not publicly available, we replace it with a publicly able query embedding model, FastText. Several constraints in production such as memory and latency requirements make certain state-of-the-art models infeasible, such as BERT model [27]. However, we note that in theory other baseline models can be swapped in as needed, like Recurrent Neural Networks (RNN) or BERT.

### B. Categorical Graph Neural Networks with Graph Convolutional Networks

The motivation of the proposed class of graph-based network structures, Categorical Graph Neural Networks (CaGNN), is to improve a baseline categorization model's performance by integrating with Graph Neural Networks, which leverage relationship information between neighbor categories and overlay the semantic information for each category. Thus model architecture of CaGNN is built on top of a baseline categorization model by adding additional GNN path to add additional information from relationships and semantic information from categories as shown in Figure 1. Our Categorical Graph Neural Networks with Graph Convolutional Networks (CaGCN) have two graph convolutional layers. The inputs to the GCN network are the features of each category. The CaGCN model is depicted in Figure 1. Path ① is a fully-connected deep neural network with 3 hidden layers with the same size of 2048. This number is chosen to match with the baseline DNN model (see Section III-A). Path ② is a fully-connected deep neural network with 3 hidden layers with the same size of 2048 to output a query embedding. The setting is identical to the path ① to have a fair comparison so that they have the same expressive power. The networks in path ① and ② do not share their weights. Intuitively, a separate path is needed for the query transformation (path ②) to transform the query to the same space as path ③. Path ③ has two graph convolutional layers with size of 2048 to generate category embeddings. Following the embedding of the categories, the query embedding (path ②) interacts with each category embedding in path ③ to generate a score for each category (see Figure 1). Here the interaction we use is a dot product to generate a scalar score for each category. The interaction can be a predefined function or a network to get the interaction output. We have experimented with interaction functions (options Ⓐ, Ⓑ and Ⓒ) to join against path ①:

- Option Ⓐ is the weighted addition of the softmax layers of the two paths.
- Option Ⓑ is concatenation with the dense layer output from path ②.
- Option Ⓒ is the addition with the dense layer output.

For each dataset we experiment with different interaction functions for the best results.

A complete description of the GCN can be found in the seminal work of Kipf and Welling [13]. We give a brief summary below.

For each GCN layer, a hidden state $H^{(l)}$ is maintained and propagated via the equation:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right), \tag{2}$$

where $\tilde{A} = A + I_N$ is the adjacency matrix of an undirected graph $G$ with added self-connections. The connections between

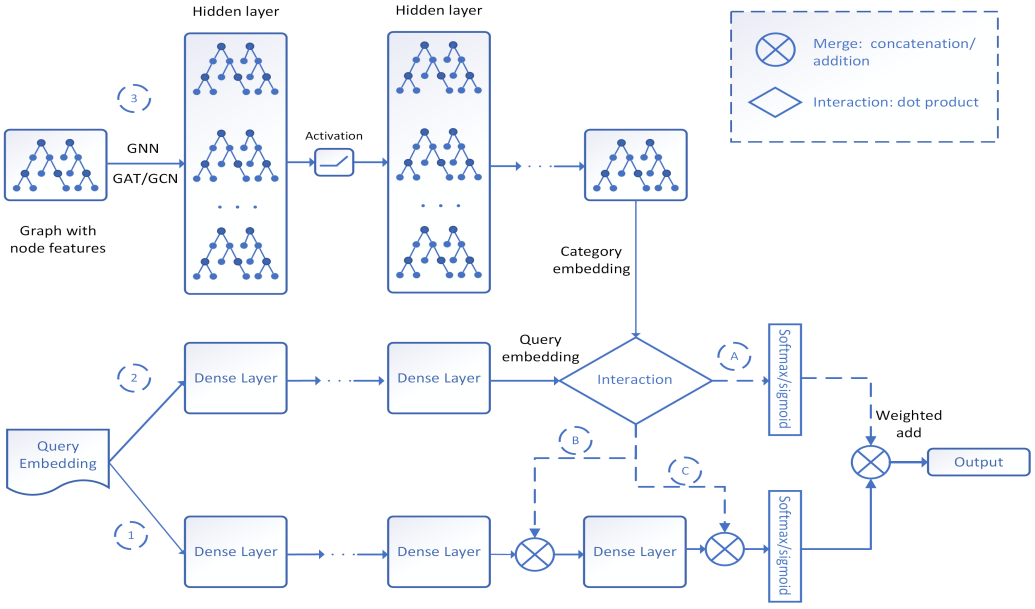


Fig. 1: Categorical Graph Neural Network Architectures

categories forms the graph (i.e. we are using the taxonomy trees). $I_N$ is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. $\tilde{D}_{ii}$ is the diagonal node degree matrix of $\tilde{A}$, and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ is an activation function (ReLU in our applications). $H^{(l)} \in \mathbb{R}^{C \times F}$ is the matrix of activations in the $l^{th}$ layer, which can be used to represent the category embeddings. $C$ is the number of categories. $F$ is the number of hidden features of each category. In the initial layer, $H^{(0)} = Z$, where $Z$ is the input category feature matrix. The last layer outputs the final category embedding (see Figure 1).

### C. Categorical Graph Neural Networks with Graph Attention Network

Our Categorical Graph Neural Networks with Graph Attention Network (CaGAT) have similar settings as the CaGCN. The CaGAT mode has all the ①, ②, and ③ paths in the network architecture diagram (see Figure 1), with path ③ using GAT layers instead of GCN layers. Path ③ has two GAT layers with size of 2048 to generate category embeddings. The GAT operations are described in [12] in detail. We give a brief summary below.

The input node features to the GAT layer, $\mathbf{h} = \left\{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_C\right\}$, $\vec{h}_i \in \mathbb{R}^F$, where $C$ is the number of categories and $F$ is the number of features in each category. The layer produces a new set of node features, $\mathbf{h}' = \left\{\vec{h}'_1, \vec{h}'_2, \ldots, \vec{h}'_C\right\}$, where $\vec{h}'_i \in \mathbb{R}^{F'}$, as its output. A shared linear transformation, parametrized by a weight matrix, $\mathbf{W} \in \mathbb{R}^{F' \times F}$, is applied to every node. Following this, self-attention is applied to the nodes. The self-attention is governed by the attention mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ to compute attention coefficients:

$$e_{ij} = a\left(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j\right), \tag{3}$$

which indicates the importance of node $j$'s features to node $i$. The graph structure is injected in the attention step. It only computes $e_{ij}$ for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is the first-order/immediate neighbors of node $i$ in the category graph. Then we normalize them across all choices of $j$ using the $softmax$ function:

$$\alpha_{ij} = \text{softmax}_j\left(e_{ij}\right) = \frac{\exp\left(e_{ij}\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(e_{ik}\right)}. \tag{4}$$

In our experiments, the attention mechanism $a$ is a single-layer feedforward neural network, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying LeakyReLU nonlinearity. The coefficient computed by the attention mechanism can be fully expressed as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T\left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j\right]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T\left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k\right]\right)\right)}, \tag{5}$$

where $\|$ is the concatenation operation. The normalized attention coefficients are used to compute a linear combination of the features corresponding to each category, to serve as the final output features for every node (After applying a nonlinear function, $\sigma$):

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\vec{h}_j\right). \tag{6}$$

We can have multiple such attention head operations in each layer (given by a prespecified constant $K$). The $K$ heads are then combined by averaging the result of each head. Thus the final output feature of each category can be expressed as:

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right). \tag{7}$$

TABLE I: Summary Statistics of Query Categorization Data

| Name | Value |
|---|---|
| Number of Training Examples | 3,432,621 |
| Number of Validation Examples | 100k |
| Number of Test Examples | 100k |
| Number of Categories | 2048 |
| Number of Root Categories | 30 |
| Avg. Length of Query | 3.43 |

TABLE II: Summary Statistics of Ads Title Categorization Data

| Name | Value |
|---|---|
| Number of Training Examples | 15,637,643 |
| Number of Validation Examples | 100k |
| Number of Test Examples | 100k |
| Number of Categories | 5427 |
| Number of Root Categories | 21 |
| Avg. Length of Ads Title | 11.0 |

We used 1 attention head for all our CaGAT networks unless specified differently. The last GAT layer outputs the category embedding as shown in Figure 1.

### D. Implementation

All of our code is implemented with Tensorflow 1.14. Some of the evaluation metrics are reported with scikit-learn package [28].

## IV. RESULTS & DISCUSSION

### A. Data

*1) Query categorization data:* The query categorization data is collected from the Bing ads impression logs, of which each example $(x_i, y_i)$ represents a query $(x_i)$ and top clicked ad category $(y_i)$ as the label for that query $(x_i)$. Click-through rate and impression thresholds were used to ensure the quality of collected data. The choice is the threshold is not in the discussion of this paper. Statistics of the dataset are summarized in Table I. The queries were normalized to lower case and special symbols removed. Then each query was converted to a 300-dimensional embedding by averaging each word's FastText embedding [23] (pretrained on the common-crawl dataset). The FastText embedding was implemented using the gensim package [29]. Each FastText word embedding was L2 normalized before use. The query categorization data was randomly split to a training dataset, a validation dataset and a test dataset to ensure they have same data distribution and fairness on reported results. In practice, we also evaluate on human labeled data. There are 3.4M examples in the training dataset. There are 100k examples in the validation dataset, which is used for tuning hyper-parameters and model selection. There are 100k examples in the test dataset, which is used for reporting the model performance. There are 2048 target categories in total. The 2048 categories are organized as a taxonomy, which can be viewed as a directed forest. It has 30 root categories. Each category has a text label and a path from a root category to that category. For example, the category, *Projector Accessories*, belongs to a category path: *Electronics > TV & Video > Projectors > Projector Accessories*. In the category path, *Electronics* is a root category and *Projector Accessories* is a leaf category. A query's target label can be any category from the 2048 categories, meaning a query's label can be a root category, a intermediate category, or a leaf category. We featurize each category as the input features to GNNs with two parts. The first part is the semantic embedding of each category's text calculated by weighted sum of FastText embedding from root category (less weight) to target category (more weight). Here we use the same FastText embedding method as the one used for embedding queries. The second part of category features are one-hot encoding of node types (root category, leaf category or inner category).

*2) Ads title categorization data:* The ads categorization data is sampled from a database where the product ad titles and ad categories are provided by Bing advertisers. Each example $(x_i, y_i)$ represents an ad title text $(x_i)$ and category $(y_i)$ for that ad $(x_i)$. Some statistics of the data is summarized in Table II. The ad title text were processed in the same way as query categorization dataset in Section IV-A1. There are 15.6M examples in the training dataset, which is about 5 times bigger the query categorization dataset. There are 100K examples in the validation dataset and 100K examples in the test dataset. There are 5427 target categories in total, which is different from the categories used for the query categorization. It is 2 times bigger than the query categorization dataset. The 5427 categories are organized as a product category taxonomy. It has 21 root categories. As in the case of the query categorization data, each category has a text label and a path from a root category to that category.

### B. Result for Query Categorization

*1) Summary of query categorization results:* The experimental result is summarized in Table III. We report the accuracy/top 1 recall, top 2 recall, top 3 recall, top 5 recall and weighted average F1 score. The top-$k$ recall means the percentage of test examples for which the correct label is in the top-$k$ predictions over all the test examples. The CaGCN-ADD model, which is the CaGCN with the option © to add the interaction output to path 1's dense layer in Figure 1, increases the accuracy from 70.23% to 71.34% and the top 3 recall from 84.59% to 85.39%. The CaGAT-ADD model, which is the CaGAT with the option © to add the interaction output to path 1's dense layer, increased the accuracy from 70.23% to 71.35% and top 3 recall from 84.59% to 85.37%. We also calculate per category level performance and macro average of all categories' performance also improved. Both the CaGAT models and the CaGCN models increased the performance significantly compared to the baseline DNN model. Although the absolute accuracy increase from 70.23% to 71.35% seems not too big, we note that it is challenging to achieve since the model needs to predict the top 1 from over 2k categories. In addition, it's significant to observe almost 1% increase of the top 5 recall, while its

TABLE III: Summary of Query Categorization Results

| Model | Accuracy (%) | Top2 Recall (%) | Top3 Recall (%) | Top5 Recall (%) | Weighted Avg. F1 (%) |
|---|---|---|---|---|---|
| Baseline DNN | 70.23 | 80.65 | 84.59 | 88.15 | 69.68 |
| **CaGCN-ADD** | **71.34** | **81.52** | **85.39** | **88.92** | **70.87** |
| **CaGAT-ADD** | **71.35** | **81.52** | **85.37** | **88.98** | **70.85** |
| CaGAT-Identity-ADD | 71.11 | 81.20 | 85.14 | 88.75 | 70.57 |
| CaGAT-ADD-Directed-Graph | 71.16 | 81.38 | 85.28 | 88.81 | 70.64 |

absolute value is already close to 90%. The improvement of CaGNN models can be explained in the following aspects. First, it reserves information comes from baseline model. Second, it adds semantic information and meta data form individual category. Third, it adds relationship information between categories learned by GNN. In this experiment, the performance of the two models (CaGCN-ADD and CaGAT-ADD) are similar. However, each model has its pros and cons in practice. The CaGAT model is directly applicable to inductive learning problems to generalize to a unseen graphs, which makes it possible to generate category embedding for unseen category or to work for modified category taxonomies. While for CaGCN, a model trained on a specific structure cannot be directly applied to a graph with a different structure since the learned filters depend on the Laplacian eigenbasis, which depend on the whole graph structure. As opposed to CaGCN, CaGAT allows for assigning different importance to neighbor nodes, and the learned attentional weights may lead to benefits in interpretability of which neighbours are the most important. Another benefit of CaGAT is that the graph is not required to be undirected like the CaGCN. Thus, several variants of the CaGAT allowing directed graph inputs were tested. However, CaGAT is more compute intensive than the CaGCN because of the additional attention operations. Due to this reason, we use CaGCN for the ads title categorization dataset (Section IV-A2), which has a much larger graph and many more training examples. For option Ⓐ, we saw decreased model performance due to the constraint on weighting of outputs of $softmax$ layers. For option Ⓑ, the model performance is similarly to option Ⓒ. However, it loses the constraint that interaction output of each category is only used for that category's scoring due to the concatenation, which makes the interaction output hard to interpret. Thus, we mainly focused on experiments with option Ⓒ.

*2) Visualization of category embedding of CaGAT:* To validate the learned embedding of each category, we visualize the t-SNE plot of each category embedding of CaGAT (GAT-ADD) model in Figure 2. Each category was colored by its root category. The root categories are plotted with a large X and the leaf categories are plotted with dots. The assumption is if the categories belong to the same category tree (start from the same root) their category embedding should be closer than categories from other trees. Good category embedding algorithms should also embed the relationship between categories into the category embeddings. Figure 2.a is t-SNE visualization of the original category features of each category, which is the concatenation of the FastText embedding of category text and one-hot encoding of node type. In Figure 2.a, we can see clearly three major clusters. The cluster on the top consists of the root nodes. The bottom right cluster consists mainly of inner nodes. From the visualization, we see that the original feature of each category are mixed together except their node types. There is little separation between the different category trees. Figure 2.b shows the intermediate category embedding output from the first GAT layer. We find the category embeddings start to cluster around the root categories. Figure 2.c shows the GAT final category embedding output by the last GAT layer. The category's clustering pattern is clearly more visible than preceding layers.

*3) Ablation study of graph:* To evaluate how the graph edges between categories affect the model performance and the category embedding, we also built a model, CaGAT-Identity-ADD, by removing all the edges of CaGAT-ADD model. That means the information cannot be passed through the connection between category nodes. The matrix used for attention operation becomes an identity matrix. The t-SNE plot in Figure 2.d shows the t-SNE plot of the final category embedding of CaGAT-Identity-ADD model. We can see that the categories are scattered around and mixed. The clustering pattern around each category tree is gone. We can see three main clusters of categories, which represents the node type encoding of root nodes, leaf nodes, and inner nodes, similar as the original category input features' visualization in Figure 2.a. The root categories plotted with a large X are clustered at the top right in Figure 2.d. The accuracy dropped from 71.35% to 71.11%. As we expect, this indicates connection between categories does make a difference for the category embedding and performance. While the performance is still better then baseline because we still have semantic information of category names, which is helpful as well.

*4) Different number of GAT layers:* The number of the CaGNN layers is a hyper parameter, which can be optimized based on the dataset. We experimented CaGAT with number of GAT layers ranges from 1 to 5. The top 1 recalls are 71.22%, 71.35%, 71.16%, 71.00%, and 71.03%, respectively. In our experiments, two GAT layers give us the best result. The deterimental effect of adding too many GNN layers is a well noted phenomenon [13]. Combining signals from too many hops away in a graph will make it noisier.

*5) Attention analysis of GAT layers:* The attention mechanism of CaGAT gives us the opportunity to analyze the message passing flow in the graph. The analysis of the attention coefficients of different layers of a 4-layer CaGAT network and 2-layer CaGAT are summarized in Table IV. The results

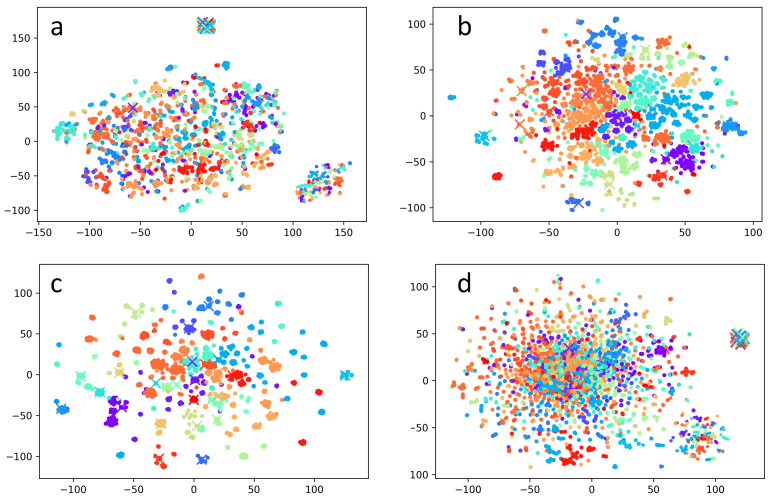


Fig. 2: t-SNE Visualization of category embeddings using CaGAT for the Query Categorization dataset. For visualization purposes, each of the 2048 categories is coloured by its root category in the taxonomy tree. a. Original input category features. b. Intermediate category embedding output from the first GAT layer. c. Final category embedding output by the last GAT layer. d. Final category embedding of CaGAT-Identity-ADD model.

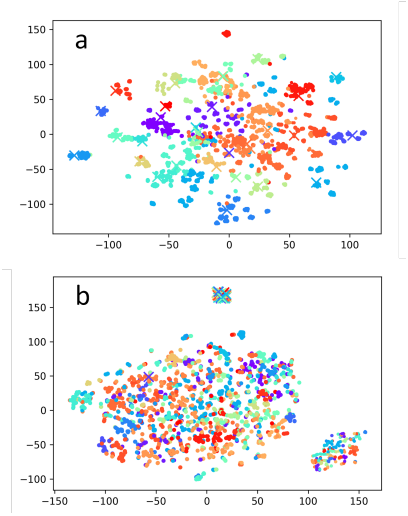


Fig. 3: t-SNE Visualization of category embeddings using CaGCN for the Query Categorization dataset. Each category is coloured by its root category in the taxonomy tree. a. Final category embedding output by the last GCN layer. b. Final category embedding of CaGCN-Identity-ADD.

TABLE IV: Summary of CaGAT attentions

| Attention values | 1st/4 layer | 2nd/4 layer | 3rd/4 layer | 4th/4 layer | 1st/2 layer | 2nd/2 layer |
|---|---|---|---|---|---|---|
| Avg. self-attention of root nodes | 0.093 | 0.133 | 0.126 | 0.135 | 0.112 | 0.127 |
| Avg. self-attention of inner nodes | 0.076 | 0.163 | 0.110 | 0.141 | 0.094 | 0.143 |
| Avg. self-attention of leaf nodes | 0.881 | 0.393 | 0.655 | 0.466 | 0.871 | 0.426 |
| Avg. attention of inner nodes on parent node | 0.071 | 0.162 | 0.124 | 0.164 | 0.090 | 0.150 |
| Avg. attention of inner nodes on child nodes | 0.852 | 0.674 | 0.764 | 0.694 | 0.814 | 0.705 |

here are particularly insightful. One finding is that for the root nodes and inner nodes more attention is put on their children and less attention on themselves. The average self-attention of those nodes are consistently less than 20%. For the inner nodes, they put much more attention on their child nodes than their parent nodes consistently. This implies the message passing flows mainly from leaf nodes to root nodes. Another interesting finding is that the first layer of leaf nodes put more attention on themselves. The average self-attention values are 0.881 for 1st layer in the 4 layer CaGAT and 0.871 for 1st layer in the 2 layer CaGAT. While for the second layer, they starts to put less attention on themselves (0.393 for 2nd layer in the 4 layer CaGAT and 0.426 for 2nd layer in the 2 layer CaGAT) and more attention on their parents. This is because the parent node of a leaf node has not yet aggregated the information of the leaf node's siblings and grandparent with the first layer. While for the second layer the parent node of a leaf node has aggregated the information of the leaf node's siblings and grandparent, the leaf node can learn more from them and put more attention on its parent node.

*6) Categories as directed graph:* Since Section IV-B5 showed the majority of message passing is from the leaves to the root, we ask if we can treat the graph as a directed graph by keeping only the directed edges from child to parent (thus dropping half of the connections in the adjacency matrix). The CaGAT-ADD-Directed-Graph is the directed version of CaGAT-ADD, and the result is shown in Table III. In our experiments, this dropped the weighted average F1 score to 0.7064 and the clustering pattern of category embedding is gone. That means the edges from both directions are important despite the fact that taxonomies have a natural parent to child relation.

*7) Visualization of CaGCN category embedding:* We visualize the t-SNE plot of each category embedding of CaGCN (CaGCN-ADD) model in Figure 3.a. Each category was colored by its root category. We can see a clustering pattern around each category tree as well, although the separation looks slightly less strong from the category embedding of CaGAT (CaGAT-ADD) model in Figure 2.c. Figure 3.b shows category embedding of CaGCN-Identity-ADD, by dropping all the connections of

CaGCN-ADD model. As expected, this breaks up the clustering patterns of the category embedding.

### C. Result for Ads Title Categorization

The result of the ads title categorization model is summarized in Table V. The baseline DNN model is the fully-connected DNN model with the same architecture as the query categorization model with the same parameter settings. Here we picked the CaGCN for the comparison because the CaGAT model is more compute-intensive due to the attention mechanism. Since the ads title category graph and number of training examples are larger than the query categorization dataset, it takes much longer for the CaGAT to train. The CaGCN model improved the accuracy from 79.14% to 80.48%. It improves the top 3 recall from 90.15% to 91.40%. Please note the top 3 recall is already over 90% and the model still increased over 1% in absolute value and over 10% in relative value, which is significant to achieve. The t-SNE plot of the category embedding of different models are illustrated in Figure 4. Overall the patterns are similar to the patterns in Figure 2 for query categorization dataset. Each category was colored by its root category as well. The root categories are plotted with bigger dots. Figure 4.a is the t-SNE of original category features of each category. We can clearly see three clusters there. They are for the root categories, inner categories and leaf categories, respectively. It's showing the same that the original feature of each category are mixed together. Figure 4.b shows the intermediate category embedding output from the first GCN layer. Figure 4.c shows the final category embedding output by GCN, which not surprisingly has stronger clustering pattern, and the boundary between different category trees are more distinguishable.

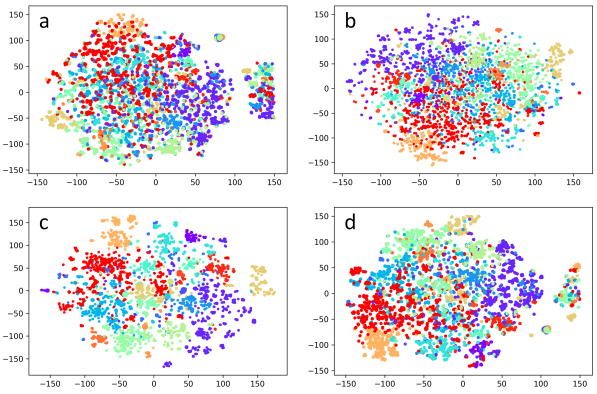


Fig. 4: t-SNE Visualization of category embeddings using CaGCN for the Ads Title dataset.

To test the effectiveness of the graph information, the CaGCN-Identity-ADD model removes all the edges in the graph. The t-SNE plot in Figure 4.d shows the t-SNE plot of the categories embedding of CaGCN-Identity-ADD model. The clustering pattern around each category tree is gone. Both the accuracy drops from 80.48% to 79.91%. As was the case for the query categorization, this indicates that the connection between categories does make a difference for the category embedding and performance.

### D. Comparison with Other Methods

When comparing with other GNN based algorithms utilizing category graph, like the Label Message Passing (LaMP) Neural Networks proposed by Lanchantin et al. [7], one advantage of our CaGNN algorithms is that they can be integrated with other baseline network structures while utilizing structure information between categorizes. Another advantage of our CaGNN algorithms is that it can generate meaningful category embeddings, which can be used in other applications, like how we use it in section IV-E1. While some other methods which encode the graph information to the loss function by an explicit graph-based regularization [20], [21] can not generate such category embeddings. In addition, those methods could not easily leverage other features from categories, like semantic features or other human designed features. However, there are also some limitations of the CaGNN algorithms. One is that they are computationally more expensive because they add two additional neural network paths to the baseline model. Another limitation is that the quality of the generated category embedding depends on the quality of training data.

### E. Applications in Production

*1) Application in ad relevance model:* Query categorization and ad categorization are important signals in Bing ads ranking models. One application of the CaGNN model is in ad relevance model. For an Bing product ad relevance task, we use a boosted decision tree type model, for which details can be found in [30], to measure the query-ad relevance for ad ranking. The ad relevance model is a binary classification model trained with human labeled data. Once our CaGNN model is trained, the category embeddings are generated. It is possible to apply this graph structure-aware category embedding to downstream tasks that use the same categories. Thus, we designed some category-to-category distance features based on category embeddings trained with our CaGNN model and apply them in our ad-ranking model. Binary category matching features (whether the query-ad categories matched exactly), along with many other features, is used in our internal ad ranking model previously. In this task, we used the trained category embeddings to measure category-to-category distance. To be precise, we can use the category-to-category Euclidean distance as features between queries and ads for ad ranking. For example, given a query with a corresponding category, *Books & Magazines* > *Magazines* > *Hobby & Special Interest Magazines* and an ad with a corresponding category, *Books & Magazines* > *Magazines* > *Sports & Leisure Magazines*. We can calculate the category embedding distance between those two categories as a feature in our model. These distances can be pre-computed for all possible category pairs, and adds little overhead when serving the model in production. In Table VI, we show the results of applying our learned category embeddings from CaGAT-ADD model (Table III) to the ad relevance model. Retraining this model with category-to-category distances features (measured via the

TABLE V: Summary of Ads Title Categorization

| Model | Accuracy (%) | Top2 Recall (%) | Top3 Recall (%) | Top5 Recall (%) | Weighted Avg. F1 (%) |
|---|---|---|---|---|---|
| Baseline DNN | 79.14 | 87.15 | 90.15 | 92.73 | 78.28 |
| **CaGCN-ADD** | **80.48** | **88.65** | **91.40** | **93.68** | **80.29** |
| CaGCN-Identity-ADD | 79.91 | 88.24 | 91.00 | 93.45 | 79.66 |

TABLE VI: AUC performance on ad-ranking tasks with CaGNN category embeddings

| Model | ROC AUC | PR AUC |
|---|---|---|
| Baseline Model | 87.43 | 62.44 |
| Model with CaGNN features | **87.97** | **63.38** |

TABLE VII: Online A/B Test Results for Ad Relevance Model

| Treatment | ΔRPM | ΔCTR | ΔDefect |
|---|---|---|---|
| Model with CaGNN features | +0.3% | **+1.81%** | **-2.64%** |

$\ell_2$ distance of the category embeddings) produces significant AUC (area under the curve) gains in Table VI. The Baseline Model, using the binary category matching features along some other existing features, gives a ROC (receiver operating characteristics) curve AUC of 87.43% and PR (precision-recall) curve AUC of 62.44%. The Model with CaGNN features model replacing the binary category matching feature with category-to-category distance features, improves the ROC AUC to 87.97% and PR AUC to 63.38%. This indicates the CaGNN based category-to-category distance feature is better than the binary category matching feature for ad relevance model and the learned category embedding is helpful.

We further evaluate the online performance of the ad relevance model, trained with added CaGNN-based category-to-category features, with online A/B experiment. Over the time from 2020/09/26 to 2020/10/17, we conducted online A/B testing in Bing Ads search system. Our traffic allocation is base on ad impression across control group and treatment group. Our control group is the existing ad relevance model in production. Our treatment group is an ad relevance model trained with added CaGNN-based category-to-category features. Table VII shows the changes in page revenue per thousand impressions (RPM), ad click-through rate (CTR), and defect rate for the treatment group. We found that the model with CaGNN features significantly increased CTR by 1.81% (p-value<0.05), significantly reduced ad defect rate by 2.64% (p-value<0.05) with +0.3% RPM (p-value>0.05). The CTR and RPM are calculated with user click data. The defect rate is calculated based on sampled ads impressions and followed by human judgement. Ads not related to search queries are labeled as defects. The increased CTR and reduced defect rate indicates the ad relevance model with CaGNN features improves the ads quality without sacrificing RPM. Given the improved performance of the treatment model, we have deployed it into production since 10/28/2020.

*2) Potential applications:* In the ad relevance model, we didn't update the query categorization model with CaGNN models directly is due to time constraint and system constraint. Instead, we used the category embeddings from CaGNN model. However, we can update the query categorization model with CaGNN models in the future since they have improved accuracy. We also note that since we only need the category embedding of the GNN output of during inference time, we can trim the GNN path of CaGNN model and replace it with the learned category embedding to avoid computationally intensive GNN operations. We can potential apply the learned category embedding to other categorization tasks, for example product image categorization. Finally, although we evaluated our Categorical Graph Neural Networks in the ad domain, it can be potentially applied to other categorization tasks where graph structure can be formulated between categories.

## V. Conclusions and Future Work

We have introduced a novel class of graph-based network structures, which we call Categorical Graph Neural Networks (CaGNN). CaGNNs aim to leverage semantic information and relationship information between categories in a taxonomy. As a by-product, category embeddings of the categories are produced during the training phase. These embeddings can be integrated to improve performance in downstream applications. Our results show that the proposed CaGNNs can improve categorization models' performance significantly over a query categorization dataset of 2k categories and another ad title categorization dataset of 5k categories. The performance improvement is consistent across the two large datasets. In addition, we show that an attention mechanism (CaGAT) can improve the overall interpretability of the model. We further applied the CaGNN model to Bing's product ads relevance model. The online A/B testing result shows the model with CaGNN-based features has increased ad click-through rate and reduced defect rate. With the improved online performance, we have deployed the model to production.

There are several potential improvements and extensions to CaGNN that could be addressed as future work, such as using GraphSAGE like GNN model to handle larger graphs [31], taking advantage of the attention mechanism to perform a thorough analysis on the model interpretability, and integrating the CaGNN on top of state-of-the-art Transformer-based NLP models, such as BERT [27]. We should see similar improvement on top of BERT-like models as well. We also plan to research on heterogeneous graphs, e.g. categories with auxiliary information such as product names.

## ACKNOWLEDGEMENT

We'd like to thank Karim Filali for providing data for ad title categorization and Changbo Hu for suggestions on writing.

## REFERENCES

[1] H. Yang, Q. Hu, and L. He, "Learning topic-oriented word embedding for query classification," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2015, pp. 188–198.

[2] J. Hu, G. Wang, F. Lochovsky, J.-t. Sun, and Z. Chen, "Understanding user's query intent with wikipedia," in *Proceedings of the 18th international conference on World wide web*, 2009, pp. 471–480.

[3] F. Wang, Z. Yang, Z. Li, and J. Zhou, "Query classification by leveraging explicit concept information," in *International Conference on Advanced Data Mining and Applications*. Springer, 2016, pp. 636–650.

[4] D. J. Brenes, D. Gayo-Avello, and K. Pérez-González, "Survey and evaluation of query intent detection methods," in *Proceedings of the 2009 workshop on Web Search Click Data*, 2009, pp. 1–7.

[5] S. Zhou, K. Cheng, and L. Men, "The survey of large-scale query classification," in *AIP Conference Proceedings*, vol. 1834, no. 1. AIP Publishing LLC, 2017, p. 040045.

[6] "Google product categories," accessed: 2020-01-30. [Online]. Available: https://support.google.com/merchants/answer/6324436?hl=en

[7] J. Lanchantin, A. Sekhon, and Y. Qi, "Neural message passing for multi-label classification," *arXiv preprint arXiv:1904.08049*, 2019.

[8] X. Wang, Y. Ye, and A. Gupta, "Zero-shot recognition via semantic embeddings and knowledge graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6857–6866.

[9] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[11] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *arXiv preprint arXiv:1812.04202*, 2018.

[12] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[14] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[15] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 499–508.

[16] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," *arXiv preprint arXiv:1703.04826*, 2017.

[17] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an, "Graph convolutional encoders for syntax-aware neural machine translation," *arXiv preprint arXiv:1704.04675*, 2017.

[18] D. Marcheggiani, J. Bastings, and I. Titov, "Exploiting semantics in neural machine translation with graph convolutional networks," *arXiv preprint arXiv:1804.08313*, 2018.

[19] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 13, no. 6, pp. 1–25, 2019.

[20] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 639–655.

[21] H. Peng, J. Li, S. Wang, L. Wang, Q. Gong, R. Yang, B. Li, P. Yu, and L. He, "Hierarchical taxonomy-aware and attentional graph capsule rcnns for large-scale multi-label text classification," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[22] M. Nickel and D. Kiela, "Poincaré embeddings for learning hierarchical representations," in *Advances in neural information processing systems*, 2017, pp. 6338–6347.

[23] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[24] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[25] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[26] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 2333–2338.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[30] C. J. Burges, "From RankNet to LambdaRank to LambdaMART: An Overview."

[31] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.