GoBERT: Ordinal Text Classification using a Graph of Neural Networks

Anonymous EMNLP submission

Abstract

Text classification is an essential problem in NLP, for which various solutions were studied over the years, including a recent development of pre-trained language models, such as BERT. In this work we focus on *ordinal text classification*, a special case of text classification, in which there is an ordinal relationship among classes. Our suggested solution, GoBERT, takes advantage of class ordinality, making use of a primary BERT-based multi-class classifier aided by a set of helper BERT-based binary classifiers, whose ordinal-based predictions are fed into a graph neural network to provide an improved prediction. To this end, we propose a novel loss function to train GoBERT to account for the ordinal classification and show its effectiveness on multiple benchmarks.

1 Introduction

The *text classification* problem (Aggarwal and Zhai, 2012) draws attention in academia and industry due to its theoretical appeal and its wide range of applications including peer review analysis, sentiment analysis, spam detection, *etc.* Text classification assigns a text to a class out of a given set of possible classes. For example, in the case of multi-class paper review analysis, a text can be assigned with a class in {*reject (1), weak reject (2), neutral (3), weak accept (4), accept (5)*}.

Literature on NLP offers a variety of solutions addressing text classification. In particular, the use of deep neural networks to learn textual representation as means for classification was shown to be effective (Lai et al., 2015; Zhang et al., 2015; Liu et al., 2016; Yang et al., 2016a). Language models, pretrained on very large corpora in a self-supervised fashion and then fine-tuned towards specific target tasks (*e.g.*, ELMo (Peters et al., 2018), BERT (Devlin et al., 2019) and its succeeding versions including RoBERTa (Liu et al., 2019), DistilBERT (Sanh et al., 2019), and ALBERT (Lan et al., 2019)) were shown to be useful in various NLP tasks, including text classification.

In this work we focus on *ordinal text classification* (Frank and Hall, 2001), where class values have ordinal relationships that induce a total order over classes. For example, for a paper review analysis an *accept* recommendation is more positive than *weak accept* and significantly more positive than a *weak reject*. In addition, reviewers may tend to avoid extreme evaluations of *accept* and *reject* and we are more likely to find weak recommendations. Ordinal classification differs from multi-class classification due to these ordinal relationships. It also differs from ranking since we care about concrete class assignments. Finally, it differs from value prediction (regression) since the class is taken from a fixed and discrete set of categories.

Ordinal text classification is mostly handled in the literature using non-ordinal text classification methodologies, disregarding the order among predicted classes. This becomes especially counterproductive when computing the error of an erroneous classification, based on which models are tuned. For example, predicting a *weak accept* class when the actual label is *accept* should not be as bad as predicting *weak reject*.

We offer a novel modification to the commonly used cross entropy loss function, *ordinal cross entropy* (*OCE*), to fine-tune (BERT-based) models to account for class ordinality and provide improved predictions addressing ordinal classification. GoBERT, our proposed approach, further exploits classes ordinality by using a *primary* BERT-based multi-class classifier, aided by a set of *helper* BERT-based binary classifiers.[1] Each *helper* model is trained for a specific sub-task (*e.g.,* is the review text tends more to *accept* or *reject*?) that affects the overall classification and is impacted by the ordinal distance between classes. Once trained, GoBERT

[1]The wordplay “graph of neural network” in the title hints to the construction of a GCN model over BERT classifiers.

uses the predictions of *primary* and its *helpers* to construct multiple ordinal graphs that are fed into a set of *graph convolutional networks* (GCNs) (Kipf and Welling, 2017). We evaluate GoBERT on several known benchmarks and show its improvements over baseline methods. Specifically, the paper offers the following contributions:

1. A novel loss function for training ordinal classifiers (*OCE*), using ordinal weighting to modify the commonly used cross entropy loss.
2. A GCN ordinal architecture (GoBERT), combining multiple *OCE*-trained BERT-based classifiers, to enhance class prediction.
3. An empirical evaluation, with known benchmarks, showing the ability of GoBERT in providing improved results, even for small ordinal class-sets (3).

The rest of the paper is organized as follows. We discuss related work in Section 2 with respect to our solution to the ordinal text classification problem, which is introduced in Section 3. An empirical evaluation (Section 4) is followed by concluding remarks (Section 5).

2 Related work

Frank and Hall (2001) were the first to address the ordinal classification problem, by training a set of independent binary classifiers (one per class) and selecting the most probable class. Rennie and Srebro (2005) cast the problem as regression with discrete ordered labels and modified classification loss functions accordingly. Others, *e.g.,* (Vanbelle and Albert, 2009; Baccianella et al., 2009) extended nominal methods to address the ordinal classification (regression). Recently, Amigo et al. (2020) proposed an effectiveness metric for ordinal classifiers, which we use in our empirical evaluation. We propose an innovative use of pre-trained language models and graph neural networks, aided by a novel ordinal-aware loss to explicitly address ordinal text classification.

Over the years, numerous methods were suggested for text classification (Aggarwal and Zhai, 2012) and the use of deep learning for the task (Aggarwal and Zhai, 2012; Lai et al., 2015; Zhang et al., 2015; Yang et al., 2016b; Joulin et al., 2017). Recently, pre-trained language models, *e.g.,* ELMo (Peters et al., 2018), led the conjoint addressing of NLP tasks. Using bidirectional transformers pre-trained on multiple tasks, BERT (Devlin et al., 2019) provided state-of-the-art performance for several NLP tasks, including (multi-class) text classification (Munikar et al., 2019; Xie et al., 2020). BERT was modified/extended for several purposes, *e.g.,* RoBERTa (Liu et al., 2019) aims to provide a more robust BERT model and ALBERT (Lan et al., 2019) aims to generate smaller BERT models, sharing the same methodology as presented by Devlin et al. (2019). Different from these works, we do not aim to modify the BERT(-based) model but rather extend its usage for the task of ordinal text classification by connecting several BERT-based models using GCNs to provide an improved prediction.

Most studies that use graph neural networks for text classification construct input text as a graph where nodes correspond to words and edges to context (*e.g.,* (Defferrard et al., 2016)) and inter-text relations (*e.g.,* (Kipf and Welling, 2017)). More recently, Yao et al. (2019) and Huang et al. (2019) used a GCN to cojoin words and documents in a heterogeneous graph. Our approach is different. We use the predictions of multiple BERT-based models, each offering a different facet of the problem, to represent an input text as a graph over which several GCNs are used for learning.

3 Ordinal Text Classification using a Graph of Neural Networks

In this work we address the *Ordinal Text Classification* (OTC) task. Similar to other text classification tasks, the input for OTC is a sequence of words $s = \langle w_1, w_2, \dots \rangle$ and the output is a predicted class $\hat{k}^s \in \{1 \dots, K\}$, where $K \in \mathbb{N}$ (usually $K > 2$). With ordinal text classification, a total order is induced over the classes $\{1, \dots, K\}$ and the outcome should prefer classes that are as close as possible to the true class.

Our proposed solution follows a common practice of using *pre-trained language models*, and specifically, BERT-based models (Devlin et al., 2019) (Section 3.1) to address the text classification (and basically any NLP) task. Therefore, we propose the use of a set of pre-trained BERT-based language models to solve the OTC problem.

Training of language models mainly uses regression-based or classification-based loss functions, which are inadequate to capture the ordinal nature of the classification. Therefore, in Section 3.2, we offer *OCE*, a loss function targeted at training ordinal classifiers.

To best capture the role of ordinality in ordinal classification tasks, a network of classifiers is

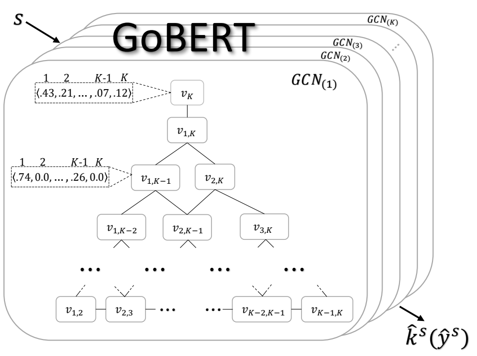

Figure 1: Illustrating GoBERT, given an input text s, GoBERT utilizes K trained GCNs ($\{\text{GCN}_{(1)}, \ldots, \text{GCN}_{(K)}\}$) each specializes in a different class prediction. Each GCN views the input text as a graph, constructed by the prediction vectors of a multi-class BERT-based classifier (v_K) and a set of binary BERT-based classifiers ($v_{k,k'}$). GoBERT 's output is a prediction vector $\hat{y}^s$, from which a class prediction $\hat{k}^s$ is produced.

constructed, connected according to their ordinal relationships, over which a *graph convolutional network* (GCN), a type of graph neural network (GNN), is trained (Section 3.3). We use a multi-class *primary* classifier, and a set of binary *helper* classifiers. Each helper is designed for a different subset of classes, *e.g.*, $m_{1,5}$ aims to distinguish between the classification of *reject* (1) and *accept* (5), and is fine-tuned accordingly (Section 3.3.1). The *primary* model prediction (v_K) and its *helpers* predictions ($v_{i,j}$) are connected via a GCN, which in our design accounts for class ordinality, as illustrated in Figure 1. The nodes in a representative graph correspond to the predictions of the aforementioned BERT-based models regarding a given text input, based on which the network aims to learn the proper classification (Section 3.3.2).

Our framework, GoBERT (**G**raph **of BERT**-based models), illustrated in Figure 1, uses K trained GCNs, $\{\text{GCN}_{(1)}, \text{GCN}_{(2)}, \ldots, \text{GCN}_{(K)}\}$, each specializing in a class $\{1, 2, \ldots, K\}$, to provide a prediction (Section 3.4).

3.1 Preliminaries: Pre-trained Language Models

We start with a brief background on BERT for completeness. See (Devlin et al., 2019) for more details.

BERT is a pre-trained language model that contains multiple layers (commonly 12) of bidirectional transformer blocks with multi-head attention. BERT tokenizes the input text sequence and adds a special classification token ($[cls]$) to its beginning. Each BERT layer generates a hidden state vector (typically of size 768). For classification tasks, the final hidden state of the token $[cls]$ is used to represent the input sequence. A single fully connected layer is then applied, followed by a $softmax$ operation to obtain an output K-dimensional vector $\hat{y}^s$. An element in the vector, $\hat{y}^s_k$ ($k \in \{1, \ldots, K\}$) represents the probability for the k'th class to be the correct one as assigned by the model. The class prediction of such model for a word sequence s can be given, for example, by $\hat{k}^s = \underset{k \in \{1,\ldots,K\}}{\text{argmax}} \ (\hat{y}^s_k)$.

In a supervised learning setting, a loss function is used to fine-tune a model. A common loss function for BERT models is Cross Entropy (CE) (Devlin et al., 2019). Given a text sequence s, let y^s be a K-dimensional correct classification vector such that $y^s_k = 1$ if k is the correct class and 0 otherwise. The cross entropy loss for s is given by:

$$CE(s, \hat{y}^s, y^s) = -\sum_{k=1}^{K} y^s_k \cdot \log(\hat{y}^s_k) \qquad (1)$$

Recall that y^s is a binary vector with a single 1 for the correct class entry, and therefore CE accounts for the probability assigned to the correct class alone, ignoring other entries in the prediction vector and the ordinality of classes. Next, we provide our alternative take on training ordinal classifiers.

3.2 Training an Ordinal Classifier

Generally, the output of a multi-class classification task is a binary vector with only a single correct class, *i.e.,* the output vector assigns a value of 1 to one class and others are set to 0. We argue that although the task outcome is binary, the model's confidence regarding other (incorrect) classes is essential for learning. In addition, in ordinal classification (and different from non-ordinal classification tasks), not every mis-prediction equally errs.

We illustrate these observations using two common loss functions, namely CE (Eq. 1) and MSE. CE disregards ordinality when computing loss, contributing the same amount of error to a classification loss regardless of its ordinal distance. For example, given a correct class of *weak accept* (4), predicting either *reject* (1) or *neutral* (3) yields the same loss. One way to account for different error weights is to use a regression-based loss (*e.g.,* MSE) for a classification task (Rennie and Srebro, 2005). This way, for example, given a correct class of *weak accept* (4), predicting the class *neutral* (3) contributes less error ($(4-3)^2 = 1$) than predicting the *reject* (1) class ($(4-1)^2 = 9$). Using a regression-based loss suffers from drawbacks since its output is viewed as scalar rather then a distribution vector and classes are assumed to be equidistant (Amigo et al., 2020). Referring again to the example, predicting *neutral* (3) and predicting *accept* (5) might not be symmetric with respect to the class *weak accept* (4) as in an MSE computation.

In what follows, we introduce a novel loss function that caters well to ordinal domains and assists learning by considering the probability distribution of predicted classification. We do so with the assistance of a proximity function ($prox$), inspired by Amigo et al.(2020), which measures similarity between classes with respect to class distribution in a given dataset (Section 3.2.1). We use $prox$ in defining a weighted ordinal cross entropy (OCE) loss in Section 3.2.2.

3.2.1 Measuring Proximity

Let n be the dataset size and n_k and $n_{k'}$ be the number of data points of classes $k, k' \in \{1, \ldots, K\}$ in the dataset, respectively. The proximity between classes k and k' is computed as follows.

$$prox\,(k,k') = \begin{cases} -\log\left(\frac{1}{n}\left(\frac{n_k}{2} + \sum_{j=k+1}^{k'} n_j\right)\right), & k < k' \\ -\log\left(\frac{1}{n}\left(\frac{n_k}{2}\right)\right), & k = k' \\ -\log\left(\frac{1}{n}\left(\frac{n_k}{2} + \sum_{j=k'}^{k-1} n_j\right)\right), & k > k' \end{cases} \tag{2}$$

For a text sequence s, given a prediction $\hat{k}^s$ and a true class k^s, the larger $prox(\hat{k}^s, k^s)$ is, the smaller the error of predicting $\hat{k}^s$ becomes. Dealing with mis-predictions, we are interested in *information distance* between classes, given by an inverse version of $prox$. For scalability, we normalize it using the maximal $prox$ value over the pairs of classes in the domain. The normalized inversed proximity (NI-$prox$) between $\hat{k}^s$ and k^s is given by

$$NI\text{-}prox\left(\hat{k}^s, k^s\right) = \frac{\max\limits_{\{k,k' \in \{1,\ldots,K\}\}} prox\,(k,k')}{prox\left(\hat{k}^s, k^s\right)} \tag{3}$$

Example 1. *To illustrate the value of using prox and NI-prox, we now compare it to the common measure MSE. Recall the paper review analysis of Section 1. Let the number of reviews be $n = 300$, $n_1 = 20$, $n_2 = 80$, $n_3 = 140$, $n_4 = 50$, and $n_5 = 10$, such that n_1 is the number of* reject *recommendations in the dataset, n_2 is the number of* weak reject, etc. *The squared distance between the classes* weak reject *(2) and* neutral *(3) is the same as that between* weak accept *(4) and* accept *(5). Yet, reviewers tend to avoid extreme evaluations, as reflected in the classes distribution. Using prox, we obtain $prox(2,3) \approx .51$ and $prox(4,5) \approx 2.14$, and accordingly, NI-$prox(2,3) \approx 8.02$ and NI-$prox(4,5) \approx 1.91$, indicating that the former error is greater than the latter.*

3.2.2 Weighted Ordinal Cross Entropy Loss

Using NI-$prox$, we can now formulate a weighted ordinal version of the cross entropy loss (*OCE*), also taking into account the model's confidence regarding incorrect classes. Recall (Section 3.1) $\hat{y}^s$ as a prediction vector for the text sequence s, which is associated with a correct classification vector y^s from which k^s is inferred using $k^s = \underset{k \in K}{\operatorname{argmax}}\,(y_k^s)$.

The ordinal cross entropy loss (*OCE*) with respect to a text sequence s, is defined as follows:

$$OCE\,(s, \hat{y}^s, y^s) = \sum_{k=1}^{K} \frac{\Phi \cdot y_k^s \cdot \log(\hat{y}_k^s)}{(1 - y_k^s) \cdot \log(\hat{y}_k^s) \cdot g(NI\text{-}prox\,(k, k^s))} \tag{4}$$

where Φ is a hyperparameter and $g(\cdot)$ is a smoothing function (*e.g.*, $\sqrt[K]{NI\text{-}prox\,(k,k^s)}$).

OCE is a normalized weighted version of CE (Eq. 1), which weighs in the confidence of the model regarding incorrect classes ($(1 - y_k^s) \cdot \log(\hat{y}_k^s)$) as well as the information distance from the correct class (NI-$prox\left(\hat{k}^s, k^s\right)$). We also introduce a balancing factor, Φ, to set a balance between the correct class (nominator) and incorrect classes (denominator).

Example 1 (cont.). *Let s_1, s_2 be two input texts associated with the predictions $\hat{y}^{s_1} = \langle 0.1, 0.5, 0.2, 0.1, 0.1\rangle$ ($y^{s_1} = \langle 0,0,1,0,0\rangle$, $k^{s_1} = 3$) and $\hat{y}^{s_2} = \langle 0.1, 0.1, 0.1, 0.5, 0.2\rangle$ ($y^{s_2} = \langle 0,0,0,0,1\rangle$, $k^{s_2} = 5$), respectively. Using CE (Eq. 1) or MSE we obtain the same error for both predictions (of $\sim$1.61 for CE and 1 for MSE). However, since the classes 2 and 3 have a larger mass in the dataset (class 2 and 3 have 27% and 47% of the reviews, respectively), a confusion between the two (as for s_1) should result in a larger penalty than a misclassification between the classes 4 and 5 (as for s_2) which constitute only 17% and 3% of the dataset, respectively. Using prox (Eq. 2) and NI-prox (Eq. 3), we account for the difference between the misclassification. $prox(\hat{k}^{s_1}, k^{s_1}) = prox(2,3) \approx .51$ (s_1) and $prox(\hat{k}^{s_2}, k^{s_2}) = prox(4,5) \approx 2.14$ (s_2), and NI-$prox(\hat{k}^{s_1}, k^{s_1}) = NI$-$prox(2,3) \approx 8.02$*

(s_1) while $NI\text{-}prox(\hat{k}^{s_2}, k^{s_2}) = NI\text{-}prox(4, 5) \approx 1.91$ (s_2). Assuming $\Phi = 5$, our loss (Eq. 4) yields $OCE(s_1) \approx 0.67$ for s_1 and $OCE(s_2) \approx 0.57$ for s_2, demonstrating the expected difference.

3.3 Ordinality-in-a-Graph

Equipped with the necessary loss function, we now describe our approach to using the ordinal interplay between classes to better train an ordinal classifier. We start by introducing primary and helper BERT-based models (Section 3.3.1) and then show how the primary and helper models can be connected using a GCN to create a class predictor with the assistance of other helpers (Section 3.3.2).

3.3.1 Primary and Helper Models

We present two BERT-based (see Section 3.1) model types, namely, *primary* and *helper* models, differing in their fine-tuning. *Primary* is fine-tuned to distinguish among the full set of classes while the *helper* models are fine-tuned to distinguish between two specific classes ($k, k' \in \{1, \ldots, K\}$).

Let $D = \{(s_i, y_i)\}_{i=1}^n$ be a training dataset where s_i is the i-th text sequence and its respective ground truth class y_i, given as a K-dimensional vector (see Section 3.1). *Primary* is fine-tuned over D using OCE, the ordinal cross entropy loss (Eq. 4). Given (an unseen) text sequence s, *primary* returns a K-dimensional prediction vector $\hat{y}^s$ such that $\hat{y}^s_k$ represents the probability for the k-th class to be the correct class (see Section 3.1).

$\binom{K}{2}$ *helper* models assist *primary* to form a final prediction. Each *helper* $m_{k,k'}$ is trained to solve a binary classification problem, where the possible classes are $k, k' \in \{1, \ldots, K\}$ ($k < k'$). Thus, for a *helper* $m_{k,k'}$ we create a sub-dataset $D_{k,k'} = \{(s_i, y_i) \,|\, (s_i, y_i) \in D \wedge \big(y_{i_k} = 1 \vee y_{i_{k'}} = 1\big)\}$, over which it is fine-tuned.

Given a text sequence s, $m_{k,k'}$ returns a 2-dimensional prediction vector corresponding to k and k'. To comply with the output of *primary*, we transform the output vector into a K-dimensional prediction vector $\hat{y}^s$ such that the entry $\hat{y}^s_k$ ($\hat{y}^s_{k'}$) represents the probability of the k-th (k'-th) class to be the correct class. All other entries are set to zero. For example, if $K = 5$ and the binary output of $m_{1,3}$ is $\langle 0.6, 0.4\rangle$, then $\hat{y}^s = \langle 0.6, 0, 0.4, 0, 0\rangle$.

3.3.2 A Graph of Neural Networks

Employing *primary* and its *helpers*, we now describe a graph convolutional network (GCN) structure to improve upon their predictions. The GCN is applied over an input graph, constructed for an input text, which is composed of *primary* and its *helpers* predictions. Given an input text sequence s, an undirected graph $G^s = (V^s, E^s)$ is created. The nodes of G^s (V^s) correspond to the predictions of *primary* and the *helper* models, *i.e.,* the feature vector representing a node $v \in V^s$ is a K-dimensional prediction vector (see Section 3.3.1). The GCN inference is performed using a message passing algorithm following Kipf and Welling (2017), with the following three main phases.

Graph Creation and Initialization: Let v_K denote the node representing the *primary* model and $v_{k,k'}$ denote the node representing the *helper* $m_{k,k'}$. Each *helper* node is associated with a distance $d(v_{k,k'}) = |k' - k|$ measuring the absolute distance between k and k'. As illustrated in Figure 1 (GCN$_{(1)}$), a graph G is constructed in a triangular shape. The *primary* node v_K (which is a multi-class classifier) is linked to $v_{1,K}$, a binary classifier that determines between the class extremes of 1 and K. Then, each layer further down is composed of nodes sharing the same associated distance. Two nodes of subsequent layers are linked by an edge if they share a common class, *e.g.*, $v_{2,K}$ is linked to $v_{1,K}$, $v_{2,K-1}$ and $v_{3,K}$. Leaf nodes ($d(v) = 1$) are also linked to each other in an ordinal form, *e.g.*, a node $v_{2,3}$ is first connected to $v_{2,4}$ and $v_{3,5}$ who are in the preceding layer and to $v_{1,2}$ and $v_{3,4}$ within the same layer.

Nodes and edges of the GCN are initialized with a feature vector, where in our case, a node $v_{k,k'}$ (v_K) is assigned with the output $\hat{y}^s$ of the pre-trained model (Section 3.3.1) $m_{k,k'}$ (*primary*). An edge (v, v') is fixed with a K-dimensional feature vector in which an element is set to 1 if it is in the interaction of the nodes it connects and 0 otherwise.

The graph structure aims to capture the multiple inter-relationships of ordinal classification. Specifically, the suggested structure allows models to share predictions with other models that have similar expertise regarding the classes. *Primary* and its *helpers* serve the network by offering predictions, and the proposed GCN framework can be used with any multi-class classifier and binary classifiers predictions, not limited to BERT-based (or any specific) model.

Messages Propagation: The GCN is composed of q layers, each produces a hidden state vector, which is generated by aggregating the vectors of the adjacent nodes. Using the multi-layer GCN, each node iteratively transmits its current information to its neighboring nodes, and to itself. The first hidden layer for the node v, $h_v^{(1)}$, is initialized to be the prediction vector of the respective model. The updates to the hidden state vectors at the layer $t+1$ ($t \in \{1, 2, \ldots, q-1\}$) of the node v are given by:

$$h_v^{(t+1)} = \sigma\left(\sum_{v' \in N(v) \cup v} \frac{1}{c_{v,v'}} \cdot h_v^{(t)} \cdot W^{(t)}\right) \quad (5)$$

where σ is an activation function (*e.g., ReLU*), $W^{(t)}$ is the weights matrix of the t-th convolution layer, and $N(v)$ is the set of neighbors of the node v in G. $c_{v,v'}$ denotes a normalization of the edge from source node v' to target node v ($e_{v,v'}$), incorporating the corresponding feature vector. We use the notation $c_{v,v'}$ for readability. See (Kipf and Welling, 2017) for additional details.

Graph Prediction: The final phase of training, after messages are propagated through the GNN, is obtaining a final prediction by performing a weighted average pooling over all final hidden layers. The hidden representation of G is given by:

$$h_G = \frac{1}{|V|} \cdot \sum_{v \in V} \beta_v \cdot h_v^{(q)} \quad (6)$$

where $h_v^{(q)}$ is the final hidden state vector of node v and $\beta_v \in [0, 1]$ is the weight of v ($\sum_{v \in V} \beta_v = 1$).

Applying a fully connected layer followed by a $softmax$ operation over h_G yields the following:

$$\hat{y}^s = \left(softmax\left(W^{(fn)} \cdot h_G\right)\right) \quad (7)$$

where $W^{(fn)}$ is the weights matrix of the fully connected layer. $\hat{y}^s$ is used, given a text sequence s, as the prediction. The respective class prediction is given by $\hat{k}^s = \underset{k \in \{1,\ldots,K\}}{\text{argmax}}\ \hat{y}^s$.

3.4 GoBERT Inference using GCNs

GoBERT's full inference is composed of K GCNs, $\{\text{GCN}_{(1)}, \text{GCN}_{(2)}, \ldots, \text{GCN}_{(K)}\}$, each specializes in a respective class $\{1, 2, \ldots, K\}$.

Each GCN is constructed as described in Section 3.3.2 and uses the same set of trained BERT-based models (Section 3.3.1). Given a dataset $D = \{(s_i, y_i)\}_{i=1}^n$, we use the respective predictions of *primary* ($\hat{y}^{s_i}$) to generate a sub-dataset $D_{(k)} = \{(s_i, y_i) \,|\, (s_i, y_i) \in D \wedge (\hat{y}_k^{s_i} = 1)\}$ over which we use a set of fine-tuned *primary* and *helpers* to construct a graph G^{s_i}. Each $\text{GCN}_{(k)}$ is trained using OCE (Eq. 4) over $D_{(k)}$. By doing so, we supervise $\text{GCN}_{(k)}$ to specialize in data samples of the k-th class.

During inference (testing), we use *primary*'s prediction $\hat{k}^s$ of an input text s to navigate a sample to $\text{GCN}_{(\hat{k}^s)}$. For example, a text that is classified as *accept* (5) by *primary* is redirected by GoBERT to $\text{GNN}_{(5)}$, whose expertise is in *accept* (5) texts.

4 Empirical Evaluation

We present empirical evidence for GoBERT effectiveness in training ordinal classifiers using three datasets and comparing to state-of-the-art baseline methods. We describe the experimental setup (Section 4.1), followed by experiment analysis (Section 4.2) and an ablation study (Section 4.3). Our empirical evaluation reveals that:

1. GoBERT obtains *state-of-the-art ordinal classification results.*
2. Replacing the standard CE loss with OCE loss, results in better ordinal classification.
3. Constructing a graph of (standard) BERT-based models and applying GCN, boosts ordinal classification results.

4.1 Setup

We now detail the datasets, evaluation measures, implementation details, and baselines.

Datasets: We use the following datasets.[2]

SST-5 (Socher et al., 2013):[3] The Stanford Sentiment Treebank is composed of single-sentence movie reviews, each associated with an ordinal sentiment class in $\{1, 2, 3, 4, 5\}$ such that 1 means very negative and 5 very positive.

SEMEVAL (Rosenthal et al., 2017):[4] SemEval-2017 is composed of tweets, from a variety of topics. To show the effectiveness of GoBERT on a small class-set, we focus on Task 4A (English), where tweets are annotated on a three level scale, namely, negative (1), neutral (2) and positive (3).

AMAZON (Ni et al., 2019):[5] A dataset of product reviews from Amazon, each associated with a rating level expressing user satisfaction on a 1-5 scale such that 1 means extremely unsatisfied and 5 means very satisfied.

[2]Train/Valid/Test splits are in an anonymous git repository: https://github.com/authorAnonymousGit/GoBERT

[3]https://nlp.stanford.edu/sentiment/code.html

[4]https://alt.qcri.org/semeval2017/task4

[5]https://nijianmo.github.io/amazon/index.html

Evaluation Measures: To evaluate ordinal classification we use the closeness evaluation measure (CEM) (Amigo et al., 2020), performed over classes prediction. Given a dataset D for which each text s is associated with a correct class k^s and a predicted class $\hat{k}^s$. The CEM measures the relative proximity ($prox$, see Eq. 2) between the prediction and the correct prediction as follows.

$$CEM(D) = \frac{\sum_{s \in D} prox(\hat{k}^s, k^s)}{\sum_{s \in D} prox(k^s, k^s)} \qquad (8)$$

We also report on standard classification quality using accuracy (ACC) and use the common regression measures of mean average error (MAE):

$$\begin{aligned} ACC(D) &= \frac{1}{D} \cdot \sum_{s \in D} \mathbb{I}_{\{k^s = \hat{k}^s\}}, \\ MAE(D) &= \frac{1}{D} \cdot \sum_{s \in D} \left| k^s - \hat{k}^s \right| \end{aligned} \qquad (9)$$

where $\mathbb{I}$ is the indicator function. MAE is used, rather than MSE, since it is known to be less sensitive to outliers. We report mesaures in percentage to comply with related works, *e.g.,* (Ke et al., 2020).

Baselines: We compare the performance of GoBERT with three baselines for all benchmarks, namely, BERT$_{LARGE}$ (BERT) (Devlin et al., 2019), ALBERT (Lan et al., 2019), and RoBERTa$_{BASE}$ (RoBERTa) (Liu et al., 2019). BERT$_{LARGE}$ provides state-of-the-art results for AMAZON (Xie et al., 2020) and all BERT-based models outperform the top results reported in (Rosenthal et al., 2017). For SST-5, a widely researched sentiment analysis benchmark, we also compare against SentiLARE (Ke et al., 2020), which to the best of our knowledge performs best, outperforming SentiX (Zhou et al., 2020) and SentiBERT (with RoBERTa) (Yin et al., 2020).[6]

Implementation Details: Experiments were performed on a server with 2 Nvidia Quadro RTX 6000 and a CentOS 6.4 operating system. Networks, implemented using huggingface transformers (Wolf et al., 2020) in PyTorch,[7] are available in an anonymous git repository.[2]

We performed preprocessing following Devlin et al. (2019), for which we set the maximum input length to 256. For the primary model (see Section 3.3.1), we experimented with the three different BERT-based models. For SST-5, we also use SentiLARE as a primary model. GoBERT fine-tunes the primary model using OCE (Eq. 4). We report on the best performing model (in terms of CEM) on the validation set. For the loss function (Eq. 4), following preliminary experiments, we fixed $g(\cdot) = \sqrt[\delta]{NI\text{-}prox\,(k, k^s)}$. After validation, we report $\Phi = K, \delta = K - 1$ for SST-5 (K=5), $\Phi = K^2, \delta = K - 1$ for SEMEVAL (K=3) and $\Phi = K^2, \delta = K + 2$ for AMAZON (K=5). We trained the models over 4 epochs (3 for the helper models) with a batch size of 16 using *Adam* optimizer (Kingma and Ba, 2014). We report average results over three different seeds.

GCNs were implemented according to Section 3.3.2 with varying graph convolution layers number ($q \in \{4, 5, 6\}$) with *Adam* optimizer (Kingma and Ba, 2014) and a batch size of 32. The hidden state sizes of the layers were chosen in $\{8, 16, 32, 64\}$ in a decreasing order, *i.e.,* $|h_t| \geq |h_{t+1}|$. For β (see Eq. 6), a set of hyperparameters controlling the weights of nodes in the final decision of GoBERT, we evaluated two alternatives during validation, namely, setting the weight of v_K to 1 and all others to 0 and equal weights of all nodes and report on the former.

4.2 Results: GoBERT as a Whole

We first evaluate the configuration of GoBERT as a whole, including a primary model trained using OCE (Eq. 4), whose results, together with the results of its helpers (binary classifiers), are fed into a graph, over which a GCN is applied (see Section 3). Section 4.3 analyzes the components of GoBERT.

The results of GoBERT and baselines (see Section 4.1) in terms of CEM (Eq. 8), MAE (Eq. 9), and ACC (Eq. 9) are reported in Table 1. We also provide relative improvements (in percentage)[8] over the respective primary model.

Our experiments confirm that GoBERT improves all examined measures over a baseline state-of-the-art BERT-based model. Specifically, GoBERT achieves an average improvement of 0.5%, 3.73%, and 1.2% in terms of CEM, MAE, and ACC, respectively, over the most competitive baseline (denoted in parenthesis). For SST-5, for which we also report current state-of-the-art results, GoBERT provides an improvement of 0.62%, 6.3% and 1.93% in terms of CEM, MAE, and ACC,

[6]Unfortunately, we were unable to run SentiLARE on other benchmarks.

[7]https://pytorch.org/

[8]recall that lower MAE is better.

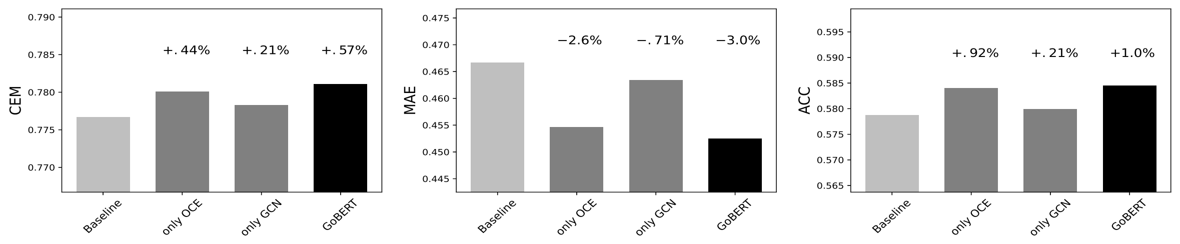

Figure 2: CEM (left), MAE (middle), and ACC (right) of a baseline (SentiLARE), after only applying the OCE loss (only OCE), after only applying the GCN (only GCN) and the full GoBERT (GoBERT).

	Models	CEM	MAE	Acc
baselines	SentiLARE	77.67	46.67	57.87
	BERT	77.01	48.69	56.52
	ALBERT	74.04	53.94	52.22
	RoBERTa	75.74	51.83	54.32
	GoBERT (SentiLARE)	**78.11**(↑ .57%)	**45.25**(↓ 3.0%)	**58.45**(↑ 1.0%)

(a) SST-5

	Models	CEM	MAE	ACC
baselines	BERT	81.05	28.66	72.81
	ALBERT	79.75	28.16	73.27
	RoBERTa	81.50	28.50	72.86
	GoBERT (RoBERTa)	**82.00**(↑ .62%)	**26.71**(↓ 6.3%)	**74.27**(↑ 1.93%)

(b) SemEval

	Models	CEM	MAE	Acc
baselines	BERT	82.96	35.02	69.21
	ALBERT	82.70	34.90	70.22
	RoBERTa	83.12	34.87	70.16
	GoBERT (RoBERTa)	**83.38**(↑ .31%)	**34.20**(↓ 1.9%)	**70.62**(↑ .66%)

(c) Amazon

Table 1: Results in terms of CEM, MAE, and ACC by dataset. Improvements (in percentage) over the respective primary models are denoted in parenthesis (lower MAE is better).

respectively, over SentiLARE (Ke et al., 2020), achieving, to the best of our knowledge, best reported results over this benchmark.

One limitation of GoBERT is in training more models, which requires longer training period. Note that for inference this overhead is negligible, adding only ∼2.4ms per batch. In the following section, we show that using a "costless" (in terms of training time) adjustment of replacing a common loss function with our suggested loss function, we can achieve better ordinal classification results.

4.3 Results: GoBERT Ablation Study

We now turn our efforts to analyze the components of GoBERT. To do so, we compare a baseline BERT-based model (*e.g.,* in this case SST-5, SentiLARE) to 1) using an ordinal loss function instead of a regular loss (only OCE), 2) using GoBERT with a primary model trained with CE (only GCN), and 3) full GoBERT model (GoBERT).

Figure 2 presents a comparison, in terms of CEM (left), MAE (middle), and ACC (right), of the aforementioned variations over SST-5.[9] As demonstrated, solely replacing the standard CE loss with our new OCE (only OCE) achieves improvements over a baseline model, improving CEM by .44%, MAE by 2.6%, and ACC by .92%. This observation is especially beneficial from an efficiency perspective as OCE by itself already boosts the results without increasing training/inference time. In addition, encoding ordinality into a graph structure, aided by a GCN learning, also boosts the results of a baseline model. Finally, we observe that the full GoBERT achieves the best results by utilizing ordinality both in the loss function and in a order-aware graph learning.

5 Conclusions and Future Work

In this work we introduce GoBERT, an architecture for ordinal text classification that uses a set of BERT-based models to construct a graph. Our proposed model uses a novel order-aware loss function to modify BERT-based models. Using an ordinal trained classifier, aided by a set of helper binary classifiers, we construct a graph of model predictions, further exploiting the ordinality of classes, over which we apply GCNs. An empirical evaluation, using well-known benchmarks and state-of-the-art baselines highlights the effectiveness of the proposed solution. Our experiments serve as a proof-of-concept that given a state-of-the-art text classifier (BERT-based or other), we can fine-tune it as an ordinal classier using our newly suggested loss and further improve its results using GoBERT.

In future work, we intend to investigate the use of our GCN-based architecture for additional tasks.

[9]Due to space limitation we only show SST-5. Other benchmarks present similar trends, given in https://github.com/authorAnonymousGit/GoBERT/tree/main/Ablation

References

Charu C Aggarwal and ChengXiang Zhai. 2012. A survey of text classification algorithms. In *Mining text data*, pages 163–222. Springer.

Enrique Amigo, Julio Gonzalo, Stefano Mizzaro, and Jorge Carrillo-de Albornoz. 2020. An effectiveness metric for ordinal classification: Formal properties and experimental results. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3938–3949.

Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. 2009. Evaluation measures for ordinal regression. In *Ninth international conference on intelligent systems design and applications.*, pages 283–287. IEEE.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems*, 29:3844–3852.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Eibe Frank and Mark Hall. 2001. A simple approach to ordinal classification. In *European Conference on Machine Learning*, pages 145–156. Springer.

Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang, and WANG Houfeng. 2019. Text level graph neural network for text classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3435–3441.

Armand Joulin, Édouard Grave, Piotr Bojanowski, and Tomáš Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers.*, pages 427–431.

Pei Ke, Haozhe Ji, Siyang Liu, Xiaoyan Zhu, and Minlie Huang. 2020. SentiLARE: Sentiment-aware language representation learning with linguistic knowledge. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6975–6988, Online. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR).*

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

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. In *International Conference on Learning Representations.*

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2873–2879.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692.*

Manish Munikar, Sushil Shakya, and Aakash Shrestha. 2019. Fine-grained sentiment classification using bert. In *Artificial Intelligence for Transforming Business and Society (AITB)*, pages 1–5.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197.

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, Volume 1 (Long Papers)*, pages 2227–2237.

Jason DM Rennie and Nathan Srebro. 2005. Loss functions for preference levels: Regression with discrete ordered labels. In *M-PREF@IJCAI*, volume 1. Kluwer Norwell, MA.

Sara Rosenthal, Noura Farra, and Preslav Nakov. 2017. Semeval-2017 task 4: Sentiment analysis in twitter. In *SemEval@ACL*, pages 502–518.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108.*

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Sophie Vanbelle and Adelin Albert. 2009. A note on the linearly weighted kappa coefficient for ordinal scales. *Statistical Methodology*, 6(2):157–163.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems*, 33.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016a. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016b. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489.

Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7370–7377.

Da Yin, Tao Meng, and Kai-Wei Chang. 2020. Sentibert: A transferable transformer-based architecture for compositional sentiment semantics. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3695–3706.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657.

Jie Zhou, Junfeng Tian, Rui Wang, Yuanbin Wu, Wenming Xiao, and Liang He. 2020. SentiX: A sentiment-aware pre-trained model for cross-domain sentiment analysis. In *Proceedings of the 28th International Conference on Computational Linguistics.*, pages 568–579.