BOOST THEN CONVOLVE: GRADIENT BOOSTING MEETS GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) are powerful models that have been successful in various graph representation learning tasks. Whereas gradient boosted decision trees (GBDT) often outperform other machine learning methods when faced with heterogeneous tabular data. But what approach should be used for graphs with tabular node features? Previous GNN models have mostly focused on networks with homogeneous sparse features and, as we show, are suboptimal in the heterogeneous setting. In this work, we propose a novel architecture that trains GBDT and GNN jointly to get the best of both worlds: the GBDT model deals with heterogeneous features, while GNN accounts for the graph structure. Our model benefits from end-to-end optimization by allowing new trees to fit the gradient updates of GNN. With an extensive experimental comparison to the leading GBDT and GNN models, we demonstrate a significant increase in performance on a variety of graphs with tabular features.

1 INTRODUCTION

Graph neural networks (GNNs) have shown great success in learning on graph-structured data with various applications in molecular design (Stokes et al., 2020), computer vision (Casas et al., 2019), combinatorial optimization (Mazyavkina et al., 2020), and recommender systems (Sun et al., 2020). The main driving force for progress is the existence of canonical GNN architecture that efficiently encodes the original input data into expressive representations, thereby achieving high-quality results on new datasets and tasks.

Recent research has mostly focused on GNNs with sparse data that represent either homogeneous node embeddings (e.g., one-hot encoded graph statistics) or bag-of-words representations. Yet tabular data with detailed information and rich semantics among nodes in the graph are more natural for many situations and abundant in real-world AI (Xiao et al., 2019). For example, in a social network, each person has socio-demographic characteristics (e.g., age, gender, date of graduation), which largely vary in data type, scale, and missing values. GNNs for graphs with tabular data remain unexplored, with gradient boosted decision trees (GBDTs) largely dominating in applications with such heterogeneous data (Bentéjac et al., 2020).

GBDTs are so successful for tabular data because they possess certain properties: (i) they efficiently learn decision space with hyperplane-like boundaries that are common in tabular data; (ii) they are well-suited for working with variables of high cardinality, features with missing values and of different scale; (iii) they provide qualitative interpretation for decision trees (e.g., by computing decrease in node impurity for every feature) or for ensembles via post-hoc analysis stage (Kaur et al., 2020); (iv) in practical applications they mostly converge faster even for large amounts of data.

In contrast, a crucial feature of GNNs is that they take into account both the neighborhood information of the nodes and the node features to make a prediction, unlike GBDTs that require additional preprocessing analysis to provide the algorithm with graph summary (e.g., through unsupervised graph embeddings (Hu et al., 2020a)). Moreover, it has been shown theoretically that message-passing GNNs can compute any function on its graph input that is computable by a Turing machine, i.e., GNN is known to be the only learning architecture that possesses universality properties on graphs (approximation (Keriven & Peyré, 2019; Maron et al., 2019) and computability (Loukas, 2020)). Furthermore, gradient-based learning of neural networks can have numerous advantages over the tree-based approach: (i) relational inductive bias imposed in GNNs alleviates the need to manually engineer

features that capture the topology of the network (Battaglia et al., 2018); (ii) the end-to-end nature of training neural networks allows multi-stage (Fey et al., 2020) or multi-component (Wang et al., 2020) integration of GNNs in application-dependent solutions; (iii) pretraining representations with graph networks enriches transfer learning for many valuable tasks such as unsupervised domain adaptation (Wu et al., 2020), self-supervised learning (Hu et al., 2020b), and active learning regimes (Satorras & Estrach, 2018).

Undoubtedly, there are major benefits in both GBDT and GNN methods. Is it possible to take advantages of both worlds? All previous approaches (Arik & Pfister, 2020; Popov et al., 2019; Badirli et al., 2020) that attempt to combine gradient boosting and neural networks are computationally heavy, do not consider graph-structured data, and suffer from the lack of relational bias imposed in GNN architectures. Other limitations of related work can be found in Appendix A. To the best of our knowledge, the current work is the first to explore using GBDT models for graph-structured data.

In this paper, we propose a novel learning architecture for graphs with tabular data, **BGNN**, that combines GBDT's learning on tabular node features with GNN that refines predictions utilizing the graph's topology. This allows **BGNN** to inherit the advantages of gradient boosting methods (heterogeneous learning and interpretability) and graph networks (representation learning and end-to-end training). Overall, our contributions are the following:

(1) We design a novel generic architecture that combines GBDT and GNN into a unique pipeline. To the best of our knowledge, this is the first work that systematically studies the application of GBDTs to graph-structured data.

(2) We overcome the challenge of end-to-end training of GBDTs by iteratively adding new trees that fit the gradient updates of GNN. This allows us to backpropagate the error signal from the topology of the network to GBDT.

(3) We perform an extensive evaluation of our approach against strong baselines in node prediction tasks. Our results consistently demonstrate significant performance improvements on heterogeneous node regression and node classification tasks over a variety of real-world graphs with tabular data.

(4) We show that our approach is also more efficient than the state-of-the-art GNN models due to much faster loss convergence during training. Furthermore, learned representations exhibit discernible structure in the latent space, which further demonstrates the expressivity of our approach.

2 BACKGROUND

Let $G = (V, E)$ be a graph with nodes having features and target labels. In node prediction tasks (classification or regression), some target labels are known and the goal is to predict the remaining ones. Throughout the text, by lowercase variables $\mathbf{x}_v$ $(v \in V)$ or $\mathbf{x}$ we denote features of individual nodes, and $\mathbf{X}$ denotes the matrix of all features for $v \in V$. Individual target labels are denoted by y_v, while Y_{all} and Y are the vectors of all and only training labels, respectively.

Graph Neural Networks (GNNs) use both the network's connectivity and the node features to learn latent representations for all nodes $v \in V$. Many popular GNNs use a neighborhood aggregation approach, also called the message-passing mechanism, where the representation of a node v is updated by applying a non-linear aggregation function of v's neighbors representation (Fey & Lenssen, 2019). Formally, GNN is a differentiable, permutation-invariant function $g_\theta : (G, \mathbf{X}) \mapsto Y_{all}$. Similar to traditional neural networks, GNNs are composed of multiple layers, each representing a non-linear message-passing function:

$$\mathbf{x}_v^t = \text{COMBINE}^t \left(\mathbf{x}_v^{t-1}, \text{AGGREGATE}^t \left(\left\{ (\mathbf{x}_w^{t-1}, \mathbf{x}_v^{t-1}) : (w, v) \in E \right\} \right) \right), \tag{1}$$

where $\mathbf{x}_v^t$ is the representation of node v at layer t, and COMBINE^t and AGGREGATE^t are (parametric) functions that aggregate representations from the local neighborhood of a node. Then, the GNN mapping g_θ includes multiple layers of aggregation (1). Parameters of GNN model θ are optimized with gradient descent by minimizing an empirical loss function $L(Y, g_\theta(G, \mathbf{X}))$.

Gradient Boosted Decision Trees (GBDT) is a well-known and widely used algorithm that is defined on non-graph tabular data (Friedman, 2001) and is particularly successful for tasks containing heterogeneous features and noisy data.

The core idea of gradient boosting is to construct a strong model by iteratively adding weak ones (usually decision trees). Formally, at each iteration t of the gradient boosting algorithm, the model $f(\mathbf{x})$ is updated in an additive manner:

$$f^t(\mathbf{x}) = f^{t-1}(\mathbf{x}) + \epsilon\, h^t(\mathbf{x}), \tag{2}$$

where f^{t-1} is a model constructed at the previous iteration, h^t is a weak learner that is chosen from some family of functions $\mathcal{H}$, and ϵ is a learning rate. The weak learner $h^t \in \mathcal{H}$ is chosen to approximate the negative gradient of the loss function L w.r.t. the current model's predictions:

$$h^t = \arg\min_{h \in \mathcal{H}} \sum_i \left(-\frac{\partial L(f^{t-1}(\mathbf{x}_i), y_i)}{\partial f^{t-1}(\mathbf{x}_i)} - h(\mathbf{x}_i) \right)^2. \tag{3}$$

The gradient w.r.t. the current predictions indicates how these predictions should be changed in order to improve the loss function. Informally, gradient boosting can be thought of as performing gradient descent in functional space.

The set of weak learners $\mathcal{H}$ is usually formed by shallow decision trees. Decision trees are built by a recursive partition of the feature space into disjoint regions called leaves. This partition is usually constructed in a greedy manner to minimize the loss function (3). Each leaf R_j of the tree is assigned to a value a_j, which is an estimate of the response y in the corresponding region. In our case a_j is equal to the average negative gradient value in the leaf R_j. To sum up, we can write $h(x) = \sum_j a_j 1_{\{x \in R_j\}}$.

3 GBDT MEETS GNN

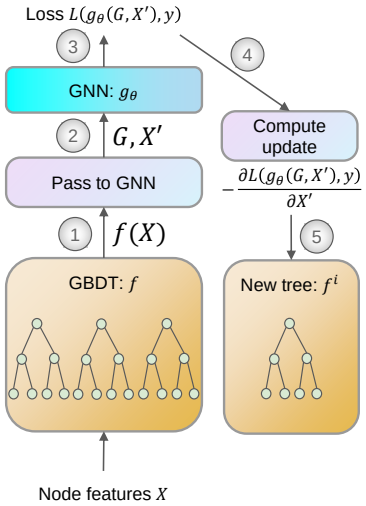


Figure 1: Training of **BGNN**, steps for one epoch are numbered.

Algorithm 1 Training of **BGNN**

Input: Graph G, node features $\mathbf{X}$, targets Y
Initialize GBDT targets $\mathcal{Y} = Y$
for epoch $i = 1$ to N **do**
  # Train k trees of GBDT model
  $f^i \underset{k}{\leftarrow} \arg\min_{f^i} L(f^i(\mathbf{X}), \mathcal{Y})$ with eq. (3)
  $f \leftarrow f + f^i$

  # Train l steps of GNN on new node features
  $\mathbf{X}' \leftarrow f(\mathbf{X})$
  $\theta, \mathbf{X}' \underset{l}{\leftarrow} \arg\min_{\theta, \mathbf{X}'} L(g_\theta(G, \mathbf{X}'), Y)$

  # Update targets for next iteration of GBDT
  $\mathcal{Y} \leftarrow \mathbf{X}' - f(\mathbf{X})$
end for
Output: Models GBDT f and GNN g_θ

Gradient boosting approach is successful for learning on tabular data; however, there are challenges of applying GBDT on graph-structured data: (i) how to propagate relational signal, in addition to node features, to otherwise inherently tabular model; and (ii) how to train it together with GNN in end-to-end fashion. Indeed, optimizations of GBDT and GNN follow different approaches: the parameters of GNN are optimized via gradient descent, while GBDT is constructed iteratively and the decision trees remain fixed after being constructed (decision trees are based on hard splits of the feature space which makes them non-differentiable).

A straightforward approach is to train the GBDT model only on the node features and then use the obtained predictions of GBDT, jointly with the original input, as new node features for GNN. In this

case, the graph-insensitive predictions of GBDT will further be refined by a graph neural network. This approach (which we call **Res-GNN**) can already boost the performance of GNN for some tasks. However, in this case, GBDT model completely ignores the graph structure and may miss descriptive features of the graph, providing inaccurate input data to GNN.

In contrast, we propose end-to-end training of GBDT and GNN called **BGNN** (for Boost-GNN). As before, we first apply GBDT and then GNN, but now we optimize both of them taking into account the quality of final predictions. The training of **BGNN** is shown in Figure 1. Recall that already built decision trees cannot be properly tuned due to their discrete structure, so we iteratively update GBDT model by adding new trees such that they approximate GNN loss function.

In Algorithm 1 we present the training of **BGNN** model that combines GBDT and GNN for any node-level prediction problem such as semi-supervised node regression or classification. In the first iteration, we build a GBDT model $f^1(\mathbf{x})$ with k decision trees by minimizing the loss function $L(f^1(\mathbf{x}), y)$ averaged over the train nodes, following equation (2). Using all predictions $f^1(\mathbf{X})$, we update the node features to $\mathbf{X}'$ that we pass to GNN. Possible update functions that we experiment with include concatenation with the original node features and their replacement by $f^1(\mathbf{X})$.

Next, we train a graph neural network g_θ on a graph G with node features $\mathbf{X}'$ for all nodes by minimizing $L(g_\theta(G, \mathbf{X}'), Y)$ with l steps of gradient descent. Importantly, we optimize both the parameters θ of GNN and the node features $\mathbf{X}'$. Then, we use the difference between the optimized node features $\mathbf{X}'_{new}$ and the input node features $\mathbf{X}' = f^1(\mathbf{X})$ as the target for the next decision trees built by GBDT. If $l = 1$, the difference $\mathbf{X}'_{new} - \mathbf{X}'$ exactly equals the negative gradient of the loss function w.r.t. the input features $\mathbf{X}'$ multiplied by the learning rate η:

$$\mathbf{X}'_{new} = \mathbf{X}' - \eta \frac{\partial L(g_\theta(G, \mathbf{X}'), Y)}{\partial \mathbf{X}'} \, .$$

In the second iteration, we train a new GBDT model f^2, with the original input features $\mathbf{X}$ but new target labels: $\mathbf{X}'_{new} - \mathbf{X}'$. Intuitively, f^2 fits the direction that would improve GNN prediction based on the first predictions $f^1(\mathbf{X})$. In other words, GBDT approximates the gradient steps made by GNN for the node features $\mathbf{X}$.

After f^2 is trained, we combine the predictions $f(\mathbf{X}) = f^1(\mathbf{X}) + f^2(\mathbf{X})$ and pass the obtained values to GNN as node features. GNN model g_θ again does l steps of backpropagation and passes the new difference $\mathbf{X}'_{new} - \mathbf{X}'$ as a target to the next iteration of GBDT. In total, the model is trained for N epochs and outputs a GBDT model $f : \mathbf{X} \mapsto Y$ and GNN model $g_\theta : (G, \mathbf{X}) \mapsto Y$, which can be used for downstream tasks.

Intuitively, **BGNN** model consists of two consecutive blocks, GBDT and GNN, which are trained end-to-end, and therefore can be interpreted from two angles: GBDT as an embedding layer for GNN or GNN as a parametric loss function for GBDT. In the former case, GBDT transforms the original input features $\mathbf{X}$ to new node features $\mathbf{X}'$ which are then passed to GNN. In the latter case, one can see **BGNN** as a standard gradient boosted training where GNN acts as a complex loss function that depends on the graph topology.

4 EXPERIMENTS

We have performed a comparative evaluation of **BGNN** and **Res-GNN** against a wide variety of strong baselines and previous approaches on heterogeneous node prediction problems, achieving significant improvement in performance across all of them. This section outlines our experimental setting, the results on heterogeneous node regression problems, and extracted feature representations.

In our first experiments, we want to answer two questions:

Q1 *Does combination of GBDT and GNN lead to better qualitative results in heterogeneous node regression and classification problems?*

Q2 *Is end-to-end training proposed in Algorithm 1 better than a combination of pretrained GBDT with GNN?*

To answer these questions, we consider several strong baselines among GBDTs, GNNs, and pure neural networks. **CatBoost** is a recent GBDT implementation (Prokhorenkova et al., 2018) that uses oblivious trees as weak learners. **LightGBM** is another GBDT model (Ke et al., 2017) that is used extensively in ML competitions. Among GNNs, we tested four state-of-the-art recent models that showed superior performance in node prediction tasks: **GAT** (Veličković et al., 2018), **GCN** (Kipf & Welling, 2016), **AGNN** (Thekumparampil et al., 2018), **APPNP** (Klicpera et al., 2018). Additionally, we test the performance of fully-connected neural network **FCNN** and its end-to-end combination with GNNs, **FCNN-GNN**.

We compare these baselines against two proposed approaches: end-to-end model **BGNN** and not end-to-end model **Res-GNN**. **BGNN** model follows Algorithm 1 and builds each tree approximating the **GNN** error in the previous iteration. In contrast, **Res-GNN** first trains a GBDT model on the training set of nodes and then appends its predictions of all nodes in a graph to the original node features, after which GNN is trained on the updated features, and GNN's predictions are used to calculate metrics. Hence, **Res-GNN** is a two-stage approach where the training of GBDT is independent of GNN. On the other hand, **BGNN** trains GBDT and GNN simultaneously in an end-to-end fashion.

We ensure that the comparison is done fairly by training each model until the convergence with a reasonable set of hyperparameters evaluated on the validation set. We run each hyperparameter setting three times and take the average of the results. Furthermore, we have five random splits of the data, and the final number represents the average performance of the model for all five random seeds. More details about hyperparameters can be found in Appendix B.

4.1 Heterogeneous node regression

4.1.1 Datasets

We utilize five real-world node regression datasets that have different properties, outlined in Table 1. Three of these datasets, **House**, **County**, and **VK**, are balanced. The distributions of the other two, **Wiki** and **Avazu**, have a heavy tail of target variables and hence are imbalanced. All datasets have a heterogeneous setting with the features of different scale, nature, and missing values. Additional details about the datasets can be found in Appendix C.

Table 1: Summary of regression datasets.

	House	**County**	**VK**	**Wiki**	**Avazu**
Setting	Balanced	Balanced	Balanced	Imbalanced	Imbalanced
# Nodes	20640	3217	54028	5201	1297
# Edges	182146	12684	213644	198493	54364
# Features/Node	6	7	14	3148	9
Mean Target	2.06	5.44	35.47	27923.86	0.08
Min Target	0.14	1.7	13.48	16	0
Max Target	5.00	24.1	118.39	849131	1
Median Target	1.79	5	33.83	9225	0

4.1.2 Results

The results of our comparative evaluation for node regression are summarized in Table 2 and Table 3. We report the mean RMSE (with standard deviation) on the test set and the relative gap between RMSE of **GAT** model (Veličković et al., 2018) and other methods, i.e., $gap = (r_m - r_{gnn})/r_{gnn}$, where r_m and r_{gnn} are RMSE of that model and of **GAT**, respectively.

Our results successfully demonstrate significant improvement of **BGNN** model across all baselines and datasets. In particular, for the balanced case **BGNN** achieves 10%, 14%, and 5% reduction of the error for **House**, **County**, and **VK** datasets, respectively. **Res-GNN** model that uses pretrained **CatBoost** model for the input of **GNN** also decreases RMSE, although not as much as the end-to-end model **BGNN**. Likewise, **BGNN** model excels on imbalanced datasets having 26% and 6% decrease of RMSE for **Wiki** and **Avazu**, respectively. This shows that ***BGNN*** *leads to better qualitative results*

Table 2: Summary of results for heterogeneous node regression in balanced setting. The gap % is relative difference w.r.t. **GAT** RMSE (the smaller, the better). Top-2 results are highlighted in **bold**.

Balanced

	Method	House RMSE	House Gap %	County RMSE	County Gap %	VK RMSE	VK Gap %
GBDT	**LightGBM**	0.63 ± 0.01	15.55	1.41 ± 0.08	-3.57	7.19 ± 0.19	-2.01
	CatBoost	0.63 ± 0.01	14.82	1.39 ± 0.08	-4.89	7.19 ± 0.20	-1.97
GNN	**GAT**	0.55 ± 0.01	0	1.47 ± 0.07	0	7.34 ± 0.19	0
	GCN	0.63 ± 0.01	15.21	1.48 ± 0.08	1.04	7.32 ± 0.19	-0.25
	AGNN	0.58 ± 0.01	6.57	1.48 ± 0.10	1.17	7.34 ± 0.18	-0.04
	APPNP	0.71 ± 0.01	28.95	1.55 ± 0.13	5.49	13.2 ± 0.10	79.76
NN	**FCNN**	0.66 ± 0.01	21.02	1.46 ± 0.09	-0.44	7.44 ± 0.18	1.38
	FCNN-GNN	0.54 ± 0.01	-1.67	1.43 ± 0.05	-2.69	7.33 ± 0.18	-0.08
Ours	**Res-GNN**	$\mathbf{0.51 \pm 0.01}$	**-7.42**	$\mathbf{1.32 \pm 0.08}$	**-10.13**	$\mathbf{7.13 \pm 0.21}$	**-2.83**
	BGNN	$\mathbf{0.49 \pm 0.01}$	**-10.28**	$\mathbf{1.26 \pm 0.09}$	**-13.92**	$\mathbf{6.95 \pm 0.21}$	**-5.26**

and its end-to-end training outperforms other approaches in node prediction tasks for graphs with tabular data.

We can also conclude that the end-to-end combination **FCNN-GNN** leads to better performance than of using pure **GNN**; however, its improvement is smaller than for **BGNN** model that uses the advantages of GBDT models. Moreover, **CatBoost**, **LightGBM**, and **FCNN** can be effective on their own, but their performance is not stable across all datasets. Overall, these experiments demonstrate the superiority of **BGNN** model against other strong models. We additionally provide results for heterogeneous node classification in Appendix D.

Table 3: Summary of results for heterogeneous node regression in imbalanced setting.

Imbalanced

	Method	Wiki RMSE	Wiki Gap %	Avazu RMSE	Avazu Gap %
GBDT	**LightGBM**	48859 ± 2993	-9.8	0.1195 ± 0.02	1.16
	CatBoost	45464 ± 3916	-16.07	0.1202 ± 0.02	1.71
GNN	**GAT**	54169 ± 4797	0	0.1181 ± 0.01	0
	GCN	$\mathbf{43022 \pm 2939}$	**-20.58**	0.1236 ± 0.02	4.63
	AGNN	48293 ± 2836	-10.85	0.1206 ± 0.02	2.06
	APPNP	52512 ± 4230	-3.06	0.1236 ± 0.02	4.63
NN	**FCNN**	46196 ± 3766	-14.72	0.1229 ± 0.02	3.99
	FCNN-GNN	45433 ± 4801	-16.13	0.1149 ± 0.01	-2.79
Ours	**Res-GNN**	49500 ± 3250	-8.62	$\mathbf{0.1141 \pm 0.01}$	**-3.39**
	BGNN	$\mathbf{40341 \pm 1858}$	**-25.53**	$\mathbf{0.1115 \pm 0.01}$	**-5.62**

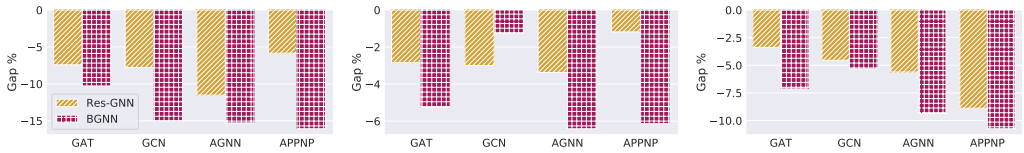
4.2 Consistency across different GNN models

Seeing that our models perform significantly better than strong baselines on various datasets, we want to test whether the improvement is consistent when different GNN models are used. In particular, we ask:

Q3 *Do different GNN models benefit from our approach of combination with GBDT model?*

To answer this question we compare GNN models that include **GAT** (Veličković et al., 2018), **GCN** (Kipf & Welling, 2016), **AGNN** (Thekumparampil et al., 2018), and **APPNP** (Klicpera et al., 2018).

We substitute each of these models to **Res-GNN** and **BGNN** models and measure the relative change in performance with respect to the original GNN's performance.



(a) **House** (b) **VK** (c) **Avazu**

Figure 2: Gap % for different GNN architectures on node regression task. The gap % is relative difference w.r.t. GNN accuracy (the smaller, the better).

In Figure 2 we report the RMSE gap between **Res-GNN** and **BGNN** for each of the GNN architectures, i.e., we compute $gap = (r_m - r_{gnn})/r_{gnn}$, where r_m and r_{gnn} are RMSE of that model and of GNN respectively. This experiment positively answers Q3 showing that *all tested GNN architectures significantly benefit from our proposed approach*. For example, for **House** dataset the decrease in the mean squared error is 10%, 15%, 15%, and 16% for **GAT**, **GCN**, **AGNN**, and **APPNP** models respectively. Additionally, one can see that end-to-end training of **BGNN** (red, squared) leads to larger improvements than a naïve combination of **CatBoost** and **GNN** in **Res-GNN** model (yellow, diagonal). Exact metrics and running time are in Appendix E.

4.3 Running time

As previous experiments demonstrated superior quality across various datasets and GNN models, it is important to understand if the additional GBDT part can become a bottleneck in terms of efficiency for running this model on real-world datasets. Hence, we ask:

Q4 *Do **BGNN** and **Res-GNN** models incur a significant increase in running time?*

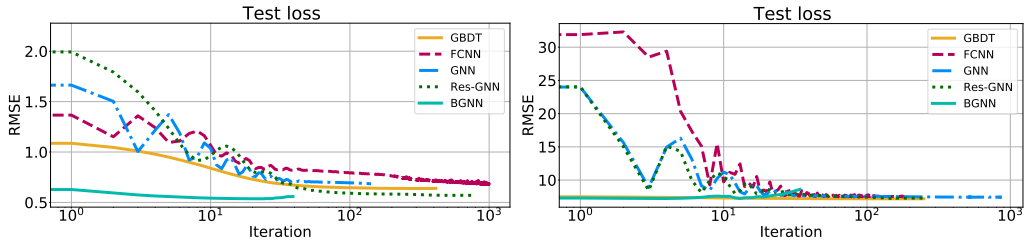
To answer this question, we measure the running time until obtaining the final results for each model and also plot RMSE on the test set during training for all models (with winning hyperparameters). Table 4 presents running time for each model. We can see that both **BGNN** and **Res-GNN** run faster than **GNN** in most cases. In other words, **BGNN** and **Res-GNN** models *do not incur an increase in running time* but actually are more efficient than **GNN**. For example, for **VK** dataset **BGNN** and **Res-GNN** run 3x and 2x faster than **GNN**, respectively. Moreover, **BGNN** model is consistently faster than another end-to-end implementation of **FCNN-GNN** model that uses **FCNN** instead of **CatBoost** to preprocess original input features.

Table 4: Summary of running time (s) in node regression task.

	Method	**House**	**County**	**VK**	**Wiki**	**Avazu**
GBDT	**LightGBM**	12.13	5.74	24.85	24.86	3.85
	CatBoost	6.39	2.2	7.50	19.96	3.30
GNN	**GAT**	52.75	76.14	39.01	22.4	13.18
	GCN	77.26	38.68	139.97	17.62	20.11
	AGNN	86.26	23.54	117.00	18.41	35.29
	APPNP	209.82	36.68	276.59	34.28	57.83
NN	**FCNN**	14.3	8.6	11.1	2.0	6.4
	FCNN-GNN	72.88	29.67	106.98	29.98	30.19
Ours	**Res-GNN**	28.13	10.49	15.11	50.2	9.61
	BGNN	50.12	4.69	13.86	23.4	7.79

The reason for improved efficiency is that **BGNN** and **Res-GNN** converge with a much fewer number of iterations as demonstrated in Figure 3. We can see that **BGNN** converges within the first

ten iterations (for $k = 20$), leading to fast training. In contrast, **Res-GNN** is similar in terms of convergence to **GNN** for the first 100 epochs, but then it continues decreasing loss function unlike **GNN** that requires much more epochs to converge. This behavior is similar in other datasets (see Appendix F).

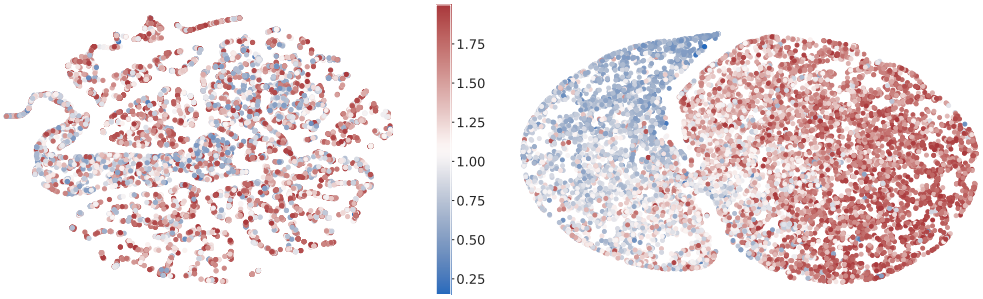


(a) House (b) VK

Figure 3: RMSE on the test set during training for two node regression datasets.

4.4 Learned representations

We also demonstrate the effectiveness of the learned representations by plotting the first layer of pretrained **GNN** and **BGNN** models. Both are pretrained on the train set of **House** dataset with the **GAT** architecture (Veličković et al., 2018). For both we extract the first hidden layer of GNN and apply the t-SNE transformation (Maaten & Hinton, 2008) with identical parameters to obtain 2D coordinates. We color each point based on the target value. This representation exhibits perceptible clustering in the projected latent space for **BGNN** model, unlike **GNN**'s embeddings which are rather scattered haphazardly. In addition to other experiments it further illustrates the discriminative power of our approach.



(a) **GNN** (b) **BGNN**

Figure 4: A t-SNE plot of the computed feature representations of **GNN** and **BGNN** model's first hidden layer on the **House** dataset. Colors correspond to target labels.

5 Conclusion

We have presented **BGNN**, a novel architecture for learning on graphs with heterogeneous tabular data. **BGNN** first takes advantages of the GBDT model to build hyperplane decision boundaries that are common for heterogeneous data, and then utilizes GNN to refine predictions using relational information. Our approach is end-to-end and can be incorporated with any message-passing neural network and gradient boosting method. Extensive experiments demonstrate that the proposed architecture is superior to strong existing competitors in terms of accuracy of predictions and running time. A promising research direction is to extend this approach to graph-level predictions such as graph classification or subgraph detection.

REFERENCES

Sercan O. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning, 2020. URL https://openreview.net/forum?id=BylRkAEKDH.

Sarkhan Badirli, Xuanqing Liu, Zhengming Xing, Avradeep Bhowmik, and Sathiya S Keerthi. Gradient boosting neural networks: Grownet. *arXiv preprint arXiv:2002.07971*, 2020.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, pp. 1–31, 2020.

Sergio Casas, Cole Gulino, Renjie Liao, and Raquel Urtasun. Spatially-aware graph neural networks for relational behavior forecasting from sensor data. *arXiv preprint arXiv:1910.08233*, 2019.

Djork-Arne Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *4th International Conference on Learning Representations*, 2016.

Ji Feng, Yang Yu, and Zhi-Hua Zhou. Multi-layered gradient boosting decision trees. In *Advances in neural information processing systems*, pp. 3551–3561, 2018.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Matthias Fey, Jan E. Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. Deep graph matching consensus. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HyeJf1HKvS.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In *37th International Conference on Machine Learning (ICML 2020)*, 2020.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. 2020a.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=HJlWWJSFDH.

Junteng Jia and Austin Benson. Outcome correlation in graph neural network regression. *arXiv preprint arXiv:2002.08274*, 2020.

Harmanpreet Kaur, Harsha Nori, Samuel Jenkins, Rich Caruana, Hanna Wallach, and Jennifer Wortman Vaughan. Interpreting interpretability: Understanding data scientists' use of interpretability tools for machine learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30*. 2017.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 7092–7101, 2019.

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

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Pan Li, Zhen Qin, Xuanhui Wang, and Donald Metzler. Combining decision trees and neural networks for learning-to-rank in personal search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2032–2040, 2019.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=B1l2bp4YwS`.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. *arXiv preprint arXiv:1901.09342*, 2019.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *arXiv preprint arXiv:2003.03600*, 2020.

R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33 (3):291–297, 1997.

Ben Peters, Vlad Niculae, and André FT Martins. Sparse sequence-to-sequence models. *arXiv preprint arXiv:1905.05702*, 2019.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In *Advances in neural information processing systems*, pp. 6638–6648, 2018.

Yuxiang Ren, Bo Liu, Chao Huang, Peng Dai, Liefeng Bo, and Jiawei Zhang. Heterogeneous deep graph infomax. *arXiv preprint arXiv:1911.08538*, 2019.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2019.

Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJj6qGbRW`.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1161–1170, 2019.

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackerman, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

Jianing Sun, Wei Guo, Dengcheng Zhang, Yingxue Zhang, Florence Regol, Yaochen Hu, Huifeng Guo, Ruiming Tang, Han Yuan, Xiuqiang He, and Mark Coates. A framework for recommending accurate and diverse items using bayesian graph convolutional neural networks. KDD '20, 2020.

Ke Sun, Zhouchen Lin, and Zhanxing Zhu. Adagcn: Adaboosting graph convolutional networks into deep models. *arXiv preprint arXiv:1908.05081*, 2019.

Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*, pp. 539–548, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Duo Wang, Mateja Jamnik, and Pietro Lio. Abstract diagrammatic reasoning with multiplex graph networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxQB1BKwH.

Man Wu, Shirui Pan, Chuan Zhou, Xiaojun Chang, and Xingquan Zhu. Unsupervised domain adaptive graph convolutional networks. In *Proceedings of The Web Conference 2020*, WWW '20, 2020.

Yuxin Xiao, Zecheng Zhang, Carl Yang, and Chengxiang Zhai. Non-local attention learning on large heterogeneous information networks. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019.

Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

Zhi-Hua Zhou and Ji Feng. Deep forest. *National Science Review*, 6(1):74–86, 2019.

A FURTHER RELATED WORK

To the best of our knowledge, there are no approaches combining the benefits of GBDT and GNN models for representation learning on graphs with tabular data. However, there are many attempts to adapt non-graph neural networks for tabular data or to combine them with gradient boosting in different ways.

Several works (Popov et al., 2019; Yang et al., 2018; Zhou & Feng, 2019; Feng et al., 2018; Hazimeh et al., 2020) attempt to mitigate the non-differentiable nature of decision trees. For example, Popov et al. (2019) proposed to replace hard choices for tree splitting features and splitting thresholds with their continuous counterparts, using α-entmax transformation (Peters et al., 2019). While such an approach becomes suitable for a union of decision trees with GNN, the computational burden of training both end-to-end becomes a bottleneck for large graphs.

Another method (Badirli et al., 2020) uses neural networks as weak learners for the GBDT model. For graph representation problems such as node regression, one can replace standard neural networks with graph neural networks. However, training different GNN as weak classifiers at once would be exhaustive. Additionally, such a combination lacks some advantages of GBDT, like handling heterogeneous and categorical features and missing values. An approach called AdaGCN (Sun et al., 2019) incorporates AdaBoost ideas into the design of GNNs in order to construct deep models. Again, this method does not exploit the advantages of GBDT methods.

Finally, Li et al. (2019) investigated different ways of combining decision-tree-based models and neural networks. While the motivation is similar to ours — get the benefits of both types of models — the paper focuses specifically on learning-to-rank problems. Additionally, while some of their methods are similar in spirit to **Res-GNN**, they do not update GBDT in an end-to-end manner, which is a substantial contribution of the current research.

B HYPERPARAMETERS

For GBDT models we have the following hyperparemters. **LightGBM** has 31 leaves, 0.9 feature fraction, 0.8 bagging fraction, 5 bagging frequency, and learning 0.03. **CatBoost** has depth 8 and learning rate 0.03. Our experiments didn't show significant variation among different hyperparameters, except learning rate, which we selected using hyperparameter tuning.

For GNN models we use the following hyperparameters. All neural models are initialized using Glorot initialization (Glorot & Bengio, 2010) and trained to minimize RMSE (regression) and cross-entropy on the training nodes using the Adam SGD optimizer (Kingma & Ba, 2014). In all cases we use an early stopping strategy on RMSE (regression) or Accuracy (classification) on the validation nodes, with a patience of 100 epochs (except for **BGNN** where we use 20 epochs due to larger number of updates in one epoch). **GAT**, **GCN**, and **AGNN** models have two convolutional layers with dropout and ELU activation function (Clevert et al., 2016). **APPNP** has a two-layer fully-connected neural network with dropout and ELU activation followed by a convolutional layer with $k = 10$ and $alpha = 0.1$. We use eight heads with eight hidden neurons for **GAT** model and 64 hidden neurons for other GNN models. For **Res-GNN** and **BGNN** models we use **CatBoost**'s predictions together with **GAT** model.

For fully-connected neural networks **FCNN** and **FCNN-GNN** we experiment with three layers, with dropout and ReLU activation function. We have a hidden dimension of 128. End-to-end combination of FCNN and GNN appends the predictions of **FCNN** to the original input data, which is then passed to **GNN** model.

All models are trained until convergence with the early stopping of 300 epochs. For neural networks, we perform a hyperparameter search on learning rate in the range [0.1, 0.01, 0.005]. Additionally, we run hyperparameters on the number of updates k and l for **BGNN** in the range [1,5,10,15,20] and whether to concatenate predictions of GBDT with the original input features. Every hyperparameter setting is evaluated three times and an average is taken. We use five random splits for train/validation/test with 0.6/0.2/0.2 ratio (for regression) and 800/400/remaining (for classification). The average across five seeds is reported in the tables.

C REGRESSION DATASETS.

In **House** dataset (Pace & Barry, 1997) nodes are the properties, edges connect the proximal nodes and the target is the price of the property. We use publicly available dataset (Pace & Barry, 1997) of all the block groups in California collected from the 1990 Census. We connect each block with at most five of its nearest neighbors if they lie within a ball of a certain radius, as measured by latitude and longitude. We keep the following node features: *MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup*.

County dataset (Jia & Benson, 2020) is a county-level election map network. Each node is a county and two nodes are connected if they share a border; we predict the unemployment rate for a county. We use a node regression dataset with the node features, coming from 2016 year. These features include *DEM, GOP, MedianIncome, MigraRate, BirthRate, DeathRate, BachelorRate, UnemploymentRate*. We follow the setup of the original paper and select UnemploymentRate as the target label. We filter out all nodes in the original data if they do not have features.

VK dataset (Tsitsulin et al., 2018) is a popular social network where people are mutually connected based on friendships and the regression problem is to predict the age of a node. We use an open-access subsample of the VK social network of the first 1M users[1]. Then, the dataset has been preprocessed to keep only the users who opt in to share their demographic information and preferences: *country, city, has_mobile, last_seen_platform, political, religion_id, alcohol, smoking, relation, sex, university*.

Wiki dataset (Rozemberczki et al., 2019) represents a page-page network on a specific topic (squirrels) with the task of predicting average monthly traffic. The features are bag-of-words for informative nouns (3148 in total) that appeared in the main text of the Wikipedia article. The target is the average monthly traffic between October 2017 and November 2018 for each article.

Avazu dataset (Song et al., 2019) represents a device-device network, with two devices being connected if they appear on the same site within the same application. The goal in this dataset is to predict CTR for each device. We take the first 10M rows from the publicly available train log of user clicks[2]. We compute click-through-rate (CTR) for each device id and filter those ids that do not have at least 10 impressions. We connect two devices if they had impressions on the same site id from the same application id. The node features are anonymized categories: *C1, C14, C15, C16, C17, C18, C19, C20, C21*.

D HETEROGENEOUS NODE CLASSIFICATION

D.1 DATASETS

A common way to represent HIN is through meta-paths, i.e., a collection of all possible paths between nodes of a particular type. For example, for citation network one may specify paths of the type paper-author-paper (PAP) and of the type paper-subject-paper (PSP). Then the original graph is approximated as several adjacency matrices for different types. To obtain a single graph in this work we use adjacency matrices for relations GG for **SLAP** and APA for **DBLP**, which closely reflect the original relationships between genes for **SLAP** and authors for **DBLP**. Each author in DBLP has a bag-of-words representation (300 words) of all the abstracts published by an author. Each gene in **SLAP** has 3000 features that correspond to the extracted gene ontology terms (GO terms). Furthermore, for every node, we compute the degrees for all types of relations and append them as additional node features. For example, for **DBLP** we have two additional node features corresponding to degrees for paper nodes in APA and APCPA adjacency matrices.

Table 5: Summary of classification datasets.

	SLAP	**DBLP**
# Nodes	20419	14475
# Edges	172248	40269
# Features/Node	2701	5002
Classes	15	4
Min Class	103	745
Max Class	534	1197

[1] https://github.com/xgfs/vk-userinfo

[2] https://www.kaggle.com/c/avazu-ctr-prediction

We utilize two heterogeneous node classification datasets adopted from heterogeneous information networks (HIN), where nodes have a few different types. A summary of statistics is outlined in Table 5.

DBLP dataset (Ren et al., 2019) is a network with three node types (authors, papers, conferences) and four target classes of the authors (database, data mining, information retrieval, and machine learning).

SLAP dataset (Xiao et al., 2019) is a multiple-hub network in bioinformatics that contains node types such as chemical compound, gene, disease, and pathway, etc. The goal is to predict one of 15 gene types.

D.2 RESULTS

Table 6: Summary of results for *heterogeneous node classification*. The gap % is relative difference w.r.t. GNN accuracy (the higher, the better). Top-2 results are highlighted in **bold**.

		Slap		**DBLP**	
	Method	Accuracy	Gap %	Accuracy	Gap %
	CatBoost	**0.9554 ± 0.01**	**127.59**	**0.8595 ± 0.01**	**13.57**
	FCNN	0.9043 ± 0.03	115.40	0.7717 ± 0.02	1.97
	GNN	0.4198 ± 0.01	0.00	0.7568 ± 0.01	0.00
Ours	**Res-GNN**	0.5010 ± 0.04	19.33	0.3865 ± 0.01	-48.94
Ours	**BGNN**	**0.9355 ± 0.00**	**122.83**	**0.8396 ± 0.01**	**10.94**

The experimental results for node regression are summarized in Tables 6. We report the mean accuracy (with standard deviation) on the test nodes and the relative gap between accuracy of GNN model (**GAT** in this experiment) and other methods, i.e., $gap = (a_m - a_{gnn})/a_{gnn}$, where a_m and a_{gnn} are accuracies of that model and of **GNN** respectively. As can be seen, this setting is particularly challenging for **GNN** due to the original sparse vector representation of input features. However, notably **BGNN** model significantly outperforms both **GNN** and **FCNN** and matches the results of **CatBoost**. This demonstrates that graph networks can benefit from end-to-end training with **CatBoost** approach presented in Algorithm 1.

E COMPARISON OF GNN MODELS

In this section we include exact RMSE and time (s) for all tested GNN models on all datasets. We consider several state-of-the-art GNN models that include **GAT** (Veličković et al., 2018), **GCN** (Kipf & Welling, 2016), **AGNN** (Thekumparampil et al., 2018), and **APPNP** (Klicpera et al., 2018).

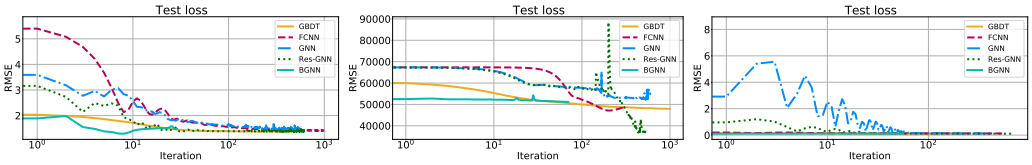
Table 7 demonstrates that for all considered models **BGNN** and **Res-GNN** achieve significant increase in performance compared to vanilla **GNN**. Additionally, end-to-end training of **BGNN** achieves typically better results than a straightforward implementation of **Res-GNN**.

Table 7: Summary of results for different GNN architectures in heterogeneous node regression. The gap % is relative difference w.r.t. GNN RMSE (the smaller, the better).

	Method	House		County		VK		Wiki		Avazu	
		RMSE	Time (s)	RMSE	Time (s)	RMSE	Time (s)	RMSE	Time (s)	RMSE	Time (s)
GAT	**GNN**	0.55	52.75	1.47	76.14	7.34	39.02	54169	22.4	0.1181	13.1
	Res-GNN	0.51	28.13	1.32	10.49	7.14	15.12	49500	50.2	0.1141	9.61
	BGNN	0.49	50.12	1.26	4.69	6.96	13.87	40341	23.4	0.1115	7.8
GCN	**GNN**	0.63	77.26	1.48	38.68	7.33	139.98	43022	17.62	0.1236	20.12
	Res-GNN	0.58	41.11	1.33	13.6	7.11	81.39	42699	16.9	0.1149	35.65
	BGNN	0.54	61.98	1.33	33.85	7.23	76.76	46426	38.33	0.112	8.25
AGNN	**GNN**	0.58	86.26	1.48	23.54	7.34	117.01	48293	18.41	0.1206	35.29
	Res-GNN	0.52	56.95	1.26	22.4	7.09	59.33	47774	17.63	0.1151	42.4
	BGNN	0.49	60.48	1.27	4.99	6.87	32.83	49508	46.58	0.1099	5.31
APPNP	**GNN**	0.71	209.82	1.55	36.68	13.2	276.59	52512	34.28	0.1236	57.83
	Res-GNN	0.66	153.09	1.41	25.55	13.05	246.93	52641	37.16	0.117	70.6
	BGNN	0.59	109.39	1.34	19.64	12.39	46.28	53171	56.03	0.1103	13.06

F LOSS CONVERGENCE

We present RMSE of test set during training for remaining datasets, **County**, **Wiki**, and **Avazu**. These results confirm that **BGNN** model converges to its optimal value within the first ten iterations (for $k = 20$), while **Res-GNN** converges similar to **GNN** for the first 100 iterations and then it continues to decrease its loss function faster than **GNN**.



(a) **County** (b) **Wiki** (c) **Avazu**

Figure 5: Summary of RMSE of test set during training for node regression datasets.