




GBNet: Gradient Boosting packages integrated into PyTorch

Michael Horrell ¹

¹ Independent Researcher, USA

DOI: [10.21105/joss.08047](https://doi.org/10.21105/joss.08047)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: Øystein Sørensen  

Reviewers:

- [@animikhaich](#)
- [@bahung](#)
- [@IBCHgenomic](#)

Submitted: 17 March 2025

Published: 07 July 2025

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

GBNet is a Python software package that integrates the powerful Gradient Boosting Machines (GBMs) (Friedman, 2001) packages XGBoost (Chen & Guestrin, 2016) and LightGBM (Ke et al., 2017) with PyTorch (Paszke et al., 2019), a widely-used deep learning library. Gradient boosting is a popular machine learning technique known for its accuracy in predictive modeling. XGBoost and LightGBM are industry-standard implementations of GBMs recognized for their speed and strong performance across numerous applications (Kaggle, 2021). However, these libraries primarily handle standard machine learning tasks and present challenges when applied to complex or non-standard modeling scenarios. For example, using non-standard loss functions with either XGBoost or LightGBM requires manual computation of gradients and Hessians, a prohibitively difficult requirement for even moderately complex losses.

PyTorch is popular for its ease of defining and training neural networks. Its computational graph provides automatic differentiation capabilities. GBNet leverages these capabilities, linking gradient and Hessian calculations from PyTorch to XGBoost or LightGBM models. This integration allows users to construct and train complex hybrid models that combine gradient boosting with neural network architectures. GBNet significantly broadens the scope of problems that can be solved with the world-leading gradient boosting software packages.

Statement of need

While XGBoost and LightGBM are industry-standard solutions for tabular data machine learning problems, they offer limited flexibility in defining complex model architectures tailored to specific problem types. Users wishing to define custom loss functions, novel architectures, or other advanced modeling scenarios face substantial difficulty due to the complex gradient and Hessian calculations required by both XGBoost and LightGBM.

As a simple motivating example, consider a forecasting model that combines a linear trend with a periodic component. A natural specification of this model might be:

$$\text{Forecast}(t) = t\beta + \text{PeriodicFn}(t)$$

where β is a constant defining the trend and PeriodicFn is modeled using a GBM. Despite its relative simplicity, this model cannot be easily fit using XGBoost or LightGBM alone.

GBNet addresses this limitation by providing PyTorch Modules that wrap XGBoost and LightGBM. These Modules serve as model building blocks like any other PyTorch Module. Valid code defining a PyTorch module implementing the above forecast model is given in just a few lines:

```
import torch
from gbnet.xgbmodule import XGBModule
```

```
class ForecastModule(torch.nn.Module):
    def __init__(self, n, d):
        super().__init__()
        self.linear = torch.nn.Linear(d, 1)
        self.xgb = XGBModule(n, d, 1)

    def forward(self, t):
        return self.linear(t) + self.xgb(t)

    def gb_step(self):
        self.xgb.gb_step()
```

The key components of this code are `XGBModule`, the wrapper for XGBoost, and `gb_step()`, a method that updates the underlying XGBoost model. The `gb_step()` method is called after each forward pass to update the gradient boosting model, while PyTorch's autograd system handles the updates for the linear component.

As demonstrated in this example, once an instance of `XGBModule` is defined, it can be combined with any other model logic supported by PyTorch. This straightforward example illustrates GBNet's ease-of-use in defining complex models.

GBNet is the first software package to combine state-of-the-art gradient boosting software with neural network packages in a near-seamless and general way. Other packages either solve similar problems by providing Gradient Boosting packages with slightly more complex capabilities (Horrell, 2020; Ingas, 2024) or, when combining GBMs and Neural Networks, resort to different types of stacking or other more complex combinations (Balestrieri, 2017; Ke et al., 2019; Kontschieder et al., 2015; Popov et al., 2019). GBNet allows users of the world's best gradient boosting packages to explore many of the rich architectural possibilities available through PyTorch.

Research Applications

Several research areas stand to benefit from GBNet. The package includes a forecasting application (`gbnet.models.forecasting`) that demonstrates improved performance over Meta's Prophet algorithm (Taylor & Letham, 2018) on a set of benchmarks, as shown in the notebook linked [here](#). The package also provides an ordinal regression implementation (`gbnet.models.ordinal_regression`) featuring the ordinal loss, which is complex, has fittable parameters, and is not included in either XGBoost or LightGBM. A notebook [here](#) demonstrates the ordinal regression application.

More broadly, GBNet may benefit any researcher looking to leverage non-parametric methods while maintaining structural control over their model. In particular, researchers using PyTorch primarily for its ability to produce outputs suited for their application may prefer GBNet at times because XGBoost and LightGBM are robust estimators. Neural networks can be finicky, requiring many small adjustments and normalizations, while GBMs often work reliably with minimal tuning.

Research into network architectures specifically tailored for GBMs may also hold intrinsic value. Several classic architectures previously explored exclusively with pure neural network methods are now accessible for GBMs through GBNet. Important concepts and methods such as embeddings (Mikolov et al., 2013), autoencoders (Hinton & Zemel, 1993), variational methods (Kingma et al., 2013), and contrastive learning (Hadsell et al., 2006) may exhibit novel and interesting properties when integrated with GBMs.

Software Description and Examples

GBNet comprises two primary sets of submodules:

- `gbnet.xgbmodule`, `gbnet.lgbmodule`, `gbnet.gblinear`: Contain PyTorch Module classes (XGBModule, LGBModule and GBLinear) that integrate XGBoost, LightGBM and a linear booster respectively.
- `gbnet.models`: Includes practical implementations of models using either XGBModule or LGBModule. Currently there are two implementations. `gbnet.models.forecasting` provides a scikit-learn interface (Pedregosa et al., 2011) for an optimized version of the forecast model shown above. `gbnet.models.ordinal_regression` provides a scikit-learn interface for ordinal regression.

Forecasting Example

`gbnet.models.forecasting.Forecast` is compared to the Meta Prophet algorithm over 500 independent trials as reported in the following table. Each trial consists of selecting a dataset uniformly at random, selecting a training cutoff uniformly at random, selecting a test period cutoff uniformly at random, and finally training a model and testing performance. The default `gbnet.models.forecasting.Forecast` beats Prophet in 74% of trials and has a higher than 50% win rate on 8 out of 9 datasets when comparing RMSE values. In addition, `gbnet.models.forecasting.Forecast`, when it has the losing RMSE, tends to lose by less in comparison to Prophet.

Dataset	N trials	GBNet win Rate (%)	Avg. GBNet Losing RMSE Ratio	Avg. Prophet Losing RMSE Ratio
Air Passengers	50	74%	1.42	1.64
Pedestrians Covid	56	66%	1.21	1.73
Pedestrians	54	70%	1.34	1.35
Multivariate				
Retail Sales	75	81%	1.26	1.97
WP Log R	59	90%	2.19	2.60
WP Log R	60	77%	1.40	2.56
Outliers1				
WP Log R	49	71%	1.85	2.47
Outliers2				
WP Log Peyton Manning	45	44%	1.36	2.22
Yosemite Temps	52	85%	2.16	2.93

Code for these results is [here](#).

Ordinal Regression Example

Ordinal regression fits a model with a 1-dimensional output, $F(X) \in \mathbb{R}$, that is thresholded at different points to achieve an ordinal classification. McCullagh (1980) introduces a cumulative logit model with thresholds to define a consistent statistical model for ordinal regression. `gbnet.models.ordinal_regression.GBOrd` implements the cumulative logit model. Specifically GBOrd fits threshold parameters $\theta_i \in \mathbb{R}$ and a GBM, $F(X)$, to optimize the likelihood defined by

$$P(y \leq i | X) = \sigma(\theta_i - F(X)).$$

Fitting this ordinal regression model using GBMs without a tool like GBNet is complex: (1) neither XGBoost nor LightGBM offer this objective; (2) calculating the negative log-likelihood (that is, its loss) has multiple steps—a cumulative distribution function is calculated and then differenced to find the likelihood; (3) the objective has parameters, θ_i , that need to be fit along with $F(X)$.

GB0rd leverages XGBModule and LGBModule and native PyTorch functionality to make fitting an ordinal regression model straightforward using either XGBoost or LightGBM back-ends. As an illustration, Figure 1 shows the fitted probabilities on the Ailerons dataset from Gagolewski (2022). The breakpoint parameters are fit via gradient descent simultaneously with the GBM. The uneven spacing of the breakpoints in the figure demonstrates that the model has learned a more optimal separation between classes rather than using evenly spaced breakpoints.

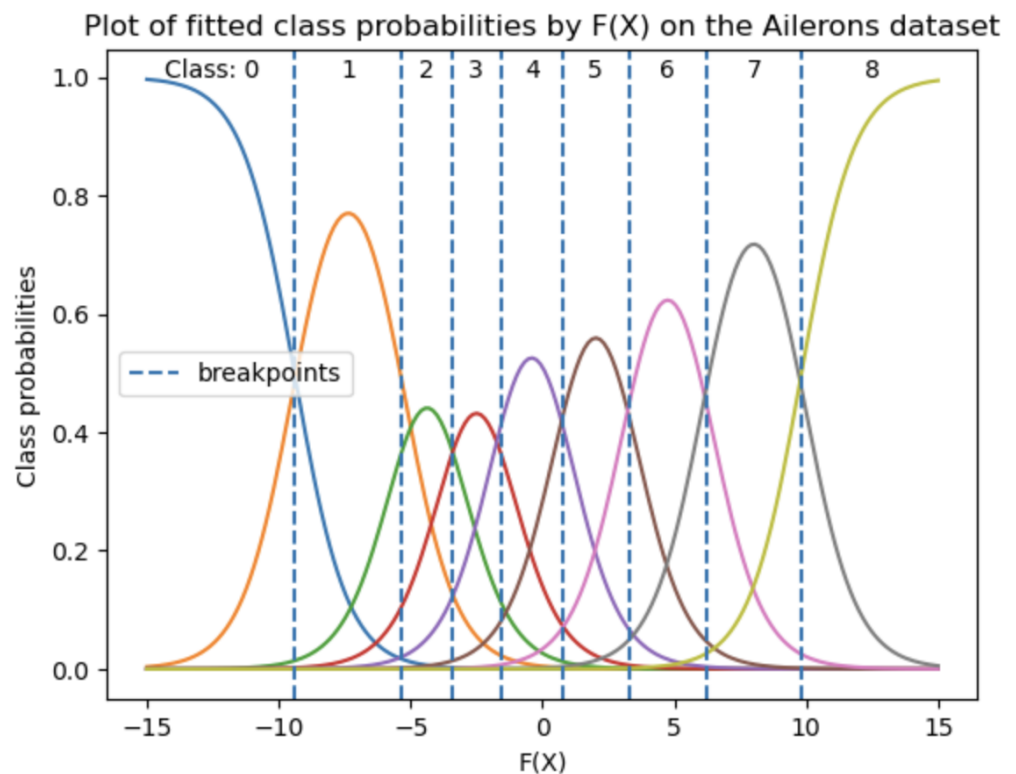


Figure 1: Fitted ordinal regression probabilities for the Ailerons dataset

A reproducible benchmark comparing GB0rd to alternative approaches across several data sets is provided [here](#).

Acknowledgements

The author gratefully acknowledges insightful feedback from Joe Guinness.

References

- Balestrieri, R. (2017). Neural decision trees. *arXiv Preprint arXiv:1702.07360*.
- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, 785–794. <https://doi.org/10.1145/2939672.2939785>

- Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), 1189–1232. <https://doi.org/10.1214/aos/1013203451>
- Gagolewski, M. (2022). *Ordinal regression*. https://github.com/gagolews/teaching-data/tree/master/ordinal_regression.
- Hadsell, R., Chopra, S., & LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2, 1735–1742. <https://doi.org/10.1109/cvpr.2006.100>
- Hinton, G. E., & Zemel, R. (1993). Autoencoders, minimum description length and Helmholtz free energy. *Advances in Neural Information Processing Systems*, 6.
- Horrell, M. (2020). Wide boosting. *CoRR*, abs/2007.09855. <https://arxiv.org/abs/2007.09855>
- Ingas, A. (2024). *OrdinalGBT - gradient boosting for ordinal regression*. <https://github.com/adamingas/ordinalgbt>
- Kaggle. (2021). *2021 Kaggle machine learning & data science survey*. Kaggle, <https://www.kaggle.com/c/kaggle-survey-2021>.
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS '17)*, 3149–3157.
- Ke, G., Xu, Z., Zhang, J., Bian, J., & Liu, T.-Y. (2019). DeepGBM: A deep learning framework distilled by GBDT for online prediction tasks. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 384–394.
- Kingma, D. P., Welling, M., & others. (2013). *Auto-encoding variational Bayes*. Banff, Canada.
- Kontschieder, P., Fiterau, M., Criminisi, A., & Rota Bulò, S. (2015). Deep neural decision forests. *Proceedings of the IEEE International Conference on Computer Vision*, 1467–1475. <https://doi.org/10.1109/ICCV.2015.172>
- McCullagh, P. (1980). Regression models for ordinal data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 42(2), 109–142. <http://www.jstor.org/stable/2984952>
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS 2019)*, 8024–8035. <https://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Popov, S., Morozov, S., & Babenko, A. (2019). Neural oblivious decision ensembles for deep learning on tabular data. *arXiv Preprint arXiv:1909.06312*.
- Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, 72(1), 37–45. <https://doi.org/10.1080/00031305.2017.1380080>