Quantization Embedding Closes the Gap between Neural Networks and XGBoost

Anonymous submission

Abstract

Tabular data is one of the last remaining fields where neural networks do not outperform classical machine learning methods. We introduce Quantization Embedding, a technique that embeds every feature of the tabular data as a binary vector: every unique value of a feature is mapped to a one-hot encoding and a positional embedding. By using this technique, we show that a simple MLP has similar performance to XGBoost over all datasets in the hardest benchmark for neural networks available.

1 Introduction

While deep learning has made significant gains in vision or natural language, tabular data, where the rows represent observations and the columns are features, is the most common data format in the real world. Tabular datasets differ from other domains in that they contain heterogeneous features. Features can be numerical and categorical, and often need to be preprocessed individually, for example, through log-transformations, scaling, or one-hot encoding.

Tabular data has been dominated by tree-based methods such as XGBoost (Chen and Guestrin 2016) and Random Forests (Breiman 2001). The tree-based methods are preferred over their deep learning counterparts not only for their speed and interpretability but also because they still outperform neural networks across many tabular datasets (Grinsztajn, Oyallon, and Varoquaux 2022; Shwartz-Ziv and Armon 2022; Borisov et al. 2022).

However, why random forests are still outperforming neural networks is an open question. Grinsztajn, Oyallon, and Varoquaux (2022) suggest the irregularity of the true underlying function might be at play. They show the decision boundaries of random forests are better at capturing the shape of highly volatile functions, while neural networks create overly smooth decision boundaries.

We propose the Quantization Embedding (QE) that tackles this form of irregularity. In QE, we take a feature from the dataset and extract the unique values. Every unique value is given a one-hot encoding and a positional encoding. Together they can capture the irregularity of a feature: the one-hot encoding makes it possible to learn a highly non-linear function with only one neural network layer. The positional encoding ensures the neural network knows the order of the unique values.

In this work, we show that a standard MLP equipped with our QE has performance superior or comparable to XGBoost. In practice, QE has the most prominent performance improvement over the standard MLP on the datasets where the gap between the standard MLP and the XGBoost is the largest. The benchmark we used is composed of datasets specifically selected because of the big gap between XGBoost and MLP. Because we close this gap, we can confidently say the gap between XGBoost and MLP, in general, has been closed.

Not only does the QE improve performance, but the method itself is also very elegant. In QE, numerical features and categorical features are treated the same: the method only depends on the unique values present. Also, QE is invariant to one-to-one transformations of the data, such as the commonly used scaling and log-transformation. This simplifies the data preprocessing for practitioners and the field of AutoML alike.

2 Related Works

Tabular Datasets. In the current literature on tabular data, many papers propose architectures that claim to outperform XGBoost on their chosen datasets. Survey papers (Shwartz-Ziv and Armon 2022; Borisov et al. 2022) show the supposed superiority over XGBoost cannot be reproduced on other datasets. Grinsztajn, Oyallon, and Varoquaux (2022) provide a benchmark that has gathered datasets for which the tree-based methods clearly outperform the neural networks. This makes it a perfect benchmark to see progress.

Embeddings. Although most papers on neural networks are related to finding efficient architectures (Kossen et al. 2021; Arik and Pfister 2021; Somepalli et al. 2021), there is little literature about tabular data embeddings for neural networks yet. Huang et al. (2020) learn embeddings for categorical values by a transformer. Quantization embedding is most similar to the piecewise linear embedding as proposed by Gorishniy, Rubachev, and Babenko (2022). We implement their method and use it as the baseline.

Bayesian search. Kadra et al. (2021) show another promising method. They add regularizers such as dropout, weight decay, and data augmentation to a simple MLP and show that a Bayesian hyperparameter search improves the performance a lot. We take over this scheme in our work.

3 Method

We first define the standard embedding used in general and our own quantile embedding. Next, we discuss the design of the rest of the neural network.

3.1 Embedding

How do we embed feature x_{ij} from the full dataset $\mathcal{D}$? We provide three ways of transforming raw data x_{ij} to the embedded data z_{ij}.

Standard The standard way of embedding x_{ij} is to scale the numerical values, and one-hot encode the categorical values. There are numerous ways to scale, for example, through robust or quantile transformations. Let f_{scale} be the scaling function used based on information from the training dataset $\mathcal{D}_T$. The standard way to embed a numerical value is then:

$$z_{ij} = f_{\text{scale}}(x_{ij})$$

Categorical values are one-hot encoded. Let $\mathcal{U}_j$ be all the values x_{ij} can take and let u_{kj} be one of the categories $k = 1, ..., p_j$. Then one-hot encoding is creating a binary vector in $\mathbb{B}^{p_j}$:

$$z_{ijk} = \begin{cases} 1 & \text{if } x_{ij} = u_{kj} \\ 0 & \text{otherwise} \end{cases}$$

The final embedding vector z_i is the concatenation of all embedded vectors z_{ijk} from categorical features and all embedded scalars z_{ij}.

Quantization We now introduce our new embedding, called quantization embedding. Let $\mathcal{U}_j$ be the set of all p_j unique values of feature j in training set $\mathcal{D}_T$. We order the unique values $u_{kj} \in \mathcal{U}_j$ such that $u_{kj} < u_{k+1,j}$ for all $k = 1, ..., p_j - 1$. For every unique value, we make a bucket that contains the unique value and all around it. The boundary of the bucket is given by h_{kj} the midpoint between two consecutive unique values:

$$h_{kj} = \frac{u_{k+1,j} + u_{kj}}{2}$$

Given these boundaries h_{kj}, we define buckets b_{kj} as follows:

$$b_{kj} = \begin{cases} (-\infty, h_{kj}] & \text{if } k = 1 \\ [h_{k-1,j}, h_{kj}], & \text{if } 2 \leq k \leq p_j - 1 \\ [h_{k-1,j}, \infty), & \text{if } k = p_j \end{cases}$$

This way, the bucket b_{kj} always contains unique value u_{kj}.

Given these buckets, we define two embeddings that we concatenate. First, the one-hot encoding:

$$z_{ijk,1} = \begin{cases} 1 & \text{if } x_{ij} \in b_{kj} \\ 0 & \text{otherwise} \end{cases}$$

and second, the positional encoding:

$$z_{ijk,2} = \begin{cases} 1 & \text{if } x_{ij} \in b_{kj} \text{ or } x_{ij} > u_{kj} \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 1: Progressive rounding of unique values

```
Input: features $x_{ij}$ from training set $\mathcal{D}_T$; Maximum
  number of unique values $M$;
Output: unique values $\mathcal{U}_j$;
$\mathcal{U}_j \leftarrow$ extract unique values$(x_{ij})$;
$d \leftarrow 20$;
while $|\mathcal{U}_j| > M$ do
    $\hat{x}_{ij} \leftarrow$ round $x_{ij}$ on $d$ digits;
    $\mathcal{U}_j \leftarrow$ extract unique values$(\hat{x}_{ij})$;
    $d \leftarrow d - 1$;
end
```

This embedding scheme is used for both the numerical and the categorical values: quantization embedding makes no distinction between the two.

In the case where the number of unique values is large, quantization embedding creates a huge embedding vector. To prevent quantization embedding from making an overly large vector, we use progressive rounding, given in Algorithm 1. Throughout this paper, we set $M = 10000$.

To further reduce the size of the network, we constrain the number of connections of the first layer. For every feature j, we create a linear layer that only sees feature j. Let the dimension of the quantization embedding be given by $d_Q = 2\sum_j p_j$. A batch $\boldsymbol{z}_j \in \mathbb{R}^{n \times 2p_j}$ with batch size n goes through a feed-forward layer individually:

$$\boldsymbol{x}_j = \text{ReLU}(\boldsymbol{W}_j \boldsymbol{z}_j)$$

The batch $\boldsymbol{x} \in \mathbb{R}^{n \times d_Q}$ is the concatenation of all individual $\boldsymbol{x}_j$ over all features. From there, it goes through a standard MLP. This setup reduces the total number of parameters needed in the first layer by setting d_Q a lot smaller than the model dimension size.

3.2 Design

We use a simple MLP with a variety of regularization techniques as done by Kadra et al. (2021). All are shown in the hyperparameter Table 2. The MLP can be of different sizes and use different activation and normalization functions. Scaler refers to the name of the Scikit-learn (Pedregosa et al. 2011) scaler. Intensity is the intensity of the data augmentation. We use the Adam optimizer with coefficients β_1 and β_2, linear warmup epochs and cosine annealing with period T_{max}. We do global unstructured pruning after training and apply Stochastic Weight Averaging starting after a certain percentage of the total number of epochs. Finally, we have the dimension of the quantization embedding. The hyperparameters are tuned by SMAC3 (Lindauer et al. 2022), a Bayesian hyperparameter search technique using random forests.

id	name	#obs	#cat	#num	XGBoost -	MLP -	MLP + PE	MLP **+ QE**
44089	credit	11699	10	0	0.7727	**0.7831**	0.7790	0.7734
44090	california	14443	8	0	**0.9007**	0.8830	0.8824	0.8725
44091	wine	1787	11	0	**0.8102**	0.6967	0.7710	0.7926
44120	electricity	26931	7	0	0.9212	0.8382	0.8768	**0.9357**
44122	pol	7056	26	0	0.9767	0.9806	**0.9855**	**0.9855**
44123	house_16H	9441	16	0	**0.8851**	0.8777	0.8832	0.8829
44124	kdd_ipums_la_97-small	3631	20	0	0.8805	0.8684	0.8498	**0.8955**
44125	MagicTelescope	9362	10	0	**0.8692**	0.8551	0.8620	0.8609
44126	bank-marketing	7404	7	0	**0.7996**	0.7859	0.7789	0.7791
44127	phoneme	2219	5	0	0.8551	**0.8866**	0.8698	0.8550
44130	eye_movements	5325	20	0	0.6505	0.5888	0.6189	**0.7631**
44156	electricity	26931	7	1	0.9327	0.8640	0.8710	**0.9414**
44157	eye_movements	5325	20	3	0.6551	0.5955	0.6380	**0.6848**
44158	KDDCup09_upselling	3521	34	11	0.8083	0.7986	0.8076	**0.8087**
44160	rl	3479	5	6	0.8058	0.6598	0.7362	**0.8152**
44162	compass	11650	8	8	0.8065	0.7942	0.7345	**0.8271**

Table 1: Main results. Comparison between accuracies of XGBoost and a standard MLP with standard embedding, with piecewise embedding (PE) and quantization embedding (QE). For every dataset, we give the OpenML id, the name, the number of observations and the number of categorical and numerical variables.

4 Experiment

We let the MLP run on 16 different datasets from the benchmark provided by Grinsztajn, Oyallon, and Varoquaux (2022). For saving time, we only consider the classification datasets and remove the datasets that run the slowest. On every dataset, we perform 100 random search runs and then 400 Bayesian search runs, each run having 100 epochs. We do the same tuning of XGBoost, using the hyperparameter space as defined by Grinsztajn, Oyallon, and Varoquaux (2022). The datasets are randomly split in a 70/10/20 validation test split. We don't do any form of K-fold validation for time reasons.

The results are given in Table 1. First, we see that XGBoost does clearly outperform a standard MLP. Although the implementation by Kadra et al. (2021) slightly differs from us, we don't think the big gap between MLP and XGBoost can be explained by that. Specifically, the wine (-11.4%), electricity (-8.3%, -6.8%), eye movement (-6.2%, -6.0%), and rl (-14.6%) datasets seem to be difficult.

The Quantization Embedding is particularly strong on these hard datasets: it changes the gap to wine (-1.7%), electricity (+1.5%, +0.9%), eye movement (+11.3%, 3.0%), and rl (-2.0%). Interestingly, the same performance increase is not apparent on the datasets where XGBoost and the standard MLP are close. This suggests our QE is the missing puzzle piece: exactly the thing that is needed to close the gap on the hardest datasets for MLP, but not a free performance boost on all datasets.

We also see that QE outperforms Gorishniy, Rubachev, and Babenko (2022)'s embedding. The accuracies of the piecewise linear + periodic embedding on the hardest datasets lie somewhere between the standard MLP and the QE. Overall, there is no dataset where they significantly outperform the QE. We conclude that we outperform the baseline.

Hyperparameters	Type	Range	Log
layers	Integer	[0, 4]	-
dimension	Integer	[1, 2048]	✓
activation	Categorical	{relu, silu, gelu, sigmoid}	-
normalization	Categorical	{none, layer, batch}	-
scaler	Categorical	{standard, robust, minmax, quantile, power}	-
			-
augmentation	Categorical	{identity, zero, swap, mixup, cutmix}	-
			-
intensity	Continous	[0, 0.5]	-
learning rate	Continous	[1e-7, 0.1]	✓
Adam β_1	Continous	[0, 0.999]	✓
Adam β_2	Continous	[0, 0.99999]	✓
decay	Continous	[1e-8, 10]	✓
warmup epochs	Integer	[0, 10]	-
cosine annealing	Categorical	{True, False}	-
cosine T_{max}	Integer	[5, 100]	✓
dropout	Continous	[0, 0.99]	-
pruning	Continous	[0, 0.99]	-
SWA	Categorical	{True, False}	-
SWA start	Continous	[0.5, 1.0]	-
Quantization	Integer	[1, 256]	✓

Table 2: The search space of Hyperparameters. Log column denotes that the hyperparameters are searched on log uniform distribution.

5 Conclusion

The Quantization Embedding is the missing puzzle piece that closes the gap between the standard MLP and XGBoost on the hardest datasets. On datasets where the gap between MLP and XGBoost is small, we see no benefit in using QE.

The next steps in our research is to understand why QE works. We plan to show the individual contributions of the one-hot encoding and the positional encoding.

References

Arik, S. O.; and Pfister, T. 2021. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 6679–6687. Issue: 8.

Borisov, V.; Leemann, T.; Seßler, K.; Haug, J.; Pawelczyk, M.; and Kasneci, G. 2022. Deep Neural Networks and Tabular Data: A Survey. ArXiv:2110.01889 [cs].

Breiman, L. 2001. Random forests. *Machine learning*, 45(1): 5–32.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. ArXiv:1603.02754 [cs].

Gorishniy, Y.; Rubachev, I.; and Babenko, A. 2022. On Embeddings for Numerical Features in Tabular Deep Learning. ArXiv:2203.05556 [cs].

Grinsztajn, L.; Oyallon, E.; and Varoquaux, G. 2022. Why do tree-based models still outperform deep learning on tabular data? ArXiv:2207.08815 [cs, stat].

Huang, X.; Khetan, A.; Cvitkovic, M.; and Karnin, Z. 2020. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*.

Kadra, A.; Lindauer, M.; Hutter, F.; and Grabocka, J. 2021. Well-tuned Simple Nets Excel on Tabular Datasets. ArXiv:2106.11189 [cs].

Kossen, J.; Band, N.; Lyle, C.; Gomez, A. N.; Rainforth, T.; and Gal, Y. 2021. Self-Attention Between Datapoints: Going Beyond Individual Input-Output Pairs in Deep Learning. In *Advances in Neural Information Processing Systems*, volume 34, 28742–28756. Curran Associates, Inc.

Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Deng, D.; Benjamins, C.; Ruhopf, T.; Sass, R.; and Hutter, F. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. ArXiv:2109.09831 [cs, stat].

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.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Shwartz-Ziv, R.; and Armon, A. 2022. Tabular data: Deep learning is not all you need. *Information Fusion*, 81: 84–90.

Somepalli, G.; Goldblum, M.; Schwarzschild, A.; Bruss, C. B.; and Goldstein, T. 2021. SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training. ArXiv:2106.01342 [cs, stat].