Modern Hopfield Networks as Memory for Iterative Learning on Tabular Data

Bernhard Schäfl[†,*] **Lukas Gruber**[†] **Angela Bitto-Nemling**[†,‡] **Sepp Hochreiter**[†,‡]
[†]ELLIS Unit Linz and LIT AI Lab, Institute for Machine Learning,
Johannes Kepler University Linz, Austria
[‡]Institute of Advanced Research in Artificial Intelligence (IARAI)

Abstract

While Deep Learning excels in structured data as encountered in vision and natural language processing, it failed to meet its expectations on tabular data. For tabular data, Support Vector Machines (SVMs), Random Forests, and Gradient Boosting are the best performing techniques. We suggest "Hopular", a novel Deep Learning architecture for medium- and small-sized datasets, where each layer is equipped with continuous modern Hopfield networks. Hopular's novelty is that every layer can directly access the original input as well as the whole training set via stored data in the Hopfield networks. Therefore, Hopular can step-wise update its current model and the resulting prediction at every layer like standard iterative learning algorithms. In experiments on small-sized tabular datasets with less than 1,000 samples, Hopular surpasses Gradient Boosting, Random Forests, SVMs, and in particular several Deep Learning methods. In experiments on medium-sized tabular data with about 10,000 samples, Hopular outperforms XGBoost, CatBoost, LightGBM and a state-of-the art Deep Learning method designed for tabular data. Thus, Hopular is a strong alternative to these methods on tabular data.

1 Introduction

Deep Learning has led to tremendous success in vision and natural language processing, where it excelled on large image and text corpora (LeCun et al., 2015; Schmidhuber, 2015). While it yielded competitive results on large tabular datasets Avati et al. (2018); Simm et al. (2018); Zhang et al. (2019b); Mayr et al. (2018), so far it could not convince on small tabular data. However, in real-world settings, small tabular datasets with less than 10,000 samples are ubiquitous. They are found in life sciences, when building a model for a certain disease with a limited number of patients, for bio-assays in drug design, or for the effect of environmental soil contamination. The same situation appears in most industrial applications, when a company wants to predict customer behavior, to control processes, to optimize its logistics, to market new products, or to employ predictive maintenance. The omnipresence of small tabular datasets can also be witnessed at Kaggle challenges. On small-sized and medium-sized tabular datasets with less than 10,000 samples, Support Vector Machines (SVMs) (Boser et al., 1992; Cortes & Vapnik, 1995; Schölkopf & Smola, 2002), Random Forests (Ho, 1995; Breiman, 2001) and, in particular, Gradient Boosting (Friedman, 2001) typically outperform Deep Learning methods with Gradient Boosting having the edge. We suggest **Hopular** to learn with modern **Hop**field networks from tab**ular** data. Hopular is a Deep Learning architecture, where each layer is equipped with continuous modern Hopfield networks (Ramsauer et al., 2021; Widrich et al., 2020). Continuous modern Hopfield networks can store two types of data: (i) the whole training set or (ii) the feature embedding vectors of the original input. Like SAINT (Somepalli et al., 2021) and NPTs (Kossen et al., 2021), Hopular can detect feature-feature, feature-target, sample-sample, and sample-

[*]Corresponding author: Bernhard Schäfl <schaefl@ml.jku.at>

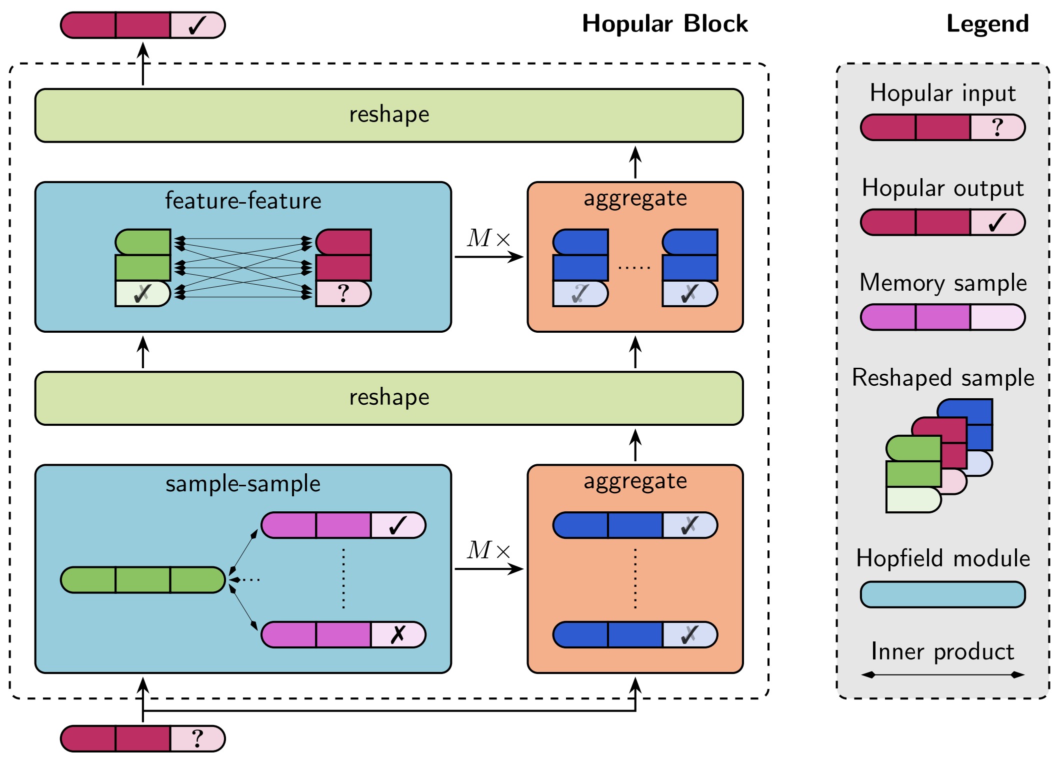

Figure 1: A Hopular Block. The first Hopfield module stores the whole training set and identifies sample-sample relations. The second Hopfield module stores the embedded input features and extracts feature-feature and feature-target relations. The Hopfield modules refine the current prediction by combining the aggregated retrievals of the M Hopfield networks with their respective input.

target dependencies via modern Hopfield networks. Hopular's novelty is that every layer can directly access the original input as well as the whole training set via stored data in the Hopfield networks. In each layer, the stored training set enables similarity-, prototype-, or quantization-based learning methods like nearest neighbor. In each layer, the stored original input enables the identification of dependencies between the features and the target. Consequently, the current model and its prediction can be step-wise improved at every layer via direct access to both the training set and the original input. Therefore, a pass through a Hopular model is similar to standard learning algorithms, which iteratively improve the current model and its prediction by re-accessing the training set. The number of iterations is fixed by the number of layers in the Hopular architecture. Hopular is most closely related to SAINT and Non-Parametric Transformers (NPTs), but in contrast to SAINT and NPTs, the whole training set and the original input are provided via Hopfield networks at every layer and not only at the input.

2 Hopular: Modern Hopfield Networks as Memory for Iterative Learning

Hopular architecture. The Hopular architecture consists of an Embedding layer, several stacked Hopular blocks, and a Summarization layer. As Hopular operates on features as well as on targets, we more generally refer to them as *attributes*.

(i) The input to the Embedding Layer is an original input sample with d attributes, including a masked target. Then a mapping to an e-dimensional embedding space together with positional embedding and encoding of feature type is applied. This also initializes the current prediction vector $\boldsymbol{\xi} \in \mathbb{R}^{d \cdot e}$ – see Figure A.2 of the Appendix.

(ii) The current prediction vector serves as input to a Hopular Block. A Hopular block consecutively applies two different Hopfield modules. Figure 1 illustrates the forward-pass of a single original input sample with the masked target indicated by the question mark (**?**). All current attribute predictions are refined. The masked target is transformed by the Hopular block to a corresponding prediction as indicated by a check mark (✓). Also feature representations can be masked as with BERT pre-training.

(iii) The Summarization Layer summarizes the refined current prediction vector resulting from the stacked Hopular blocks – see Figure A.3 of the Appendix.

(I) Hopfield Module H_s. The first Hopfield module H_s implements a modern Hopfield network for Deep Learning architectures similar to `HopfieldLayer` (Ramsauer et al., 2021, 2020) with the training set as fixed stored patterns. The current input $\boldsymbol{\xi}$ (which is also the current prediction from the previous layer) to Hopfield module H_s is interacting with the whole training data as described in Eq. (1). Hence, the Hopfield module H_s identifies sample-sample relations and can perform similarity searches like a nearest-neighbor search in the whole training data. H_s can also average over training data that are similar to a mapping of the current prediction vector $\boldsymbol{\xi}$. Let d be the number of attributes, e the embedding dimension of each single attribute, h the dimension of the Hopfield embedding space, and n the number of samples in the training set. The forward-pass for module H_s with one Hopfield network and current prediction vector $\boldsymbol{\xi} \in \mathbb{R}^{d \cdot e}$, learned weight matrices $\boldsymbol{W}_{\boldsymbol{\xi}}, \boldsymbol{W}_{\boldsymbol{X}} \in \mathbb{R}^{h \times (d \cdot e)}$, $\boldsymbol{W}_{\boldsymbol{S}} \in \mathbb{R}^{(d \cdot e) \times h}$, the stored training set $\boldsymbol{X} \in \mathbb{R}^{(d \cdot e) \times n}$, and a fixed scaling parameter β is given as

$$H_s\left(\boldsymbol{\xi}\right) \;=\; \boldsymbol{W}_{\boldsymbol{S}}\, \boldsymbol{W}_{\boldsymbol{X}}\, \boldsymbol{X} \mathrm{softmax}(\beta\, \boldsymbol{X}^T\, \boldsymbol{W}_{\boldsymbol{X}}^T\, \boldsymbol{W}_{\boldsymbol{\xi}}\, \boldsymbol{\xi})\,. \tag{1}$$

H_s may contain more than one continuous modern Hopfield network in similar vein to Multi-Head Self-Attention (Vaswani et al., 2017).

(II) Hopfield Module H_f. The second Hopfield module H_f implements a modern Hopfield network for Deep Learning architectures similar to H_s but with the embedded features of the original input sample as stored patterns. The refined prediction vector from the previous layer is reshaped and transposed to the matrix Ξ, which serves as input to the Hopfield module H_f. Ξ interacts with the embedded features of the original input sample as described in Eq. (2). Therefore, the Hopfield module H_f extracts and models feature-feature and feature-target relations. Current feature and target predictions are adjusted and refined after they are associated with the original input sample feature representations. The matrix $\Xi \in \mathbb{R}^{e \times d}$ is a transposed and reshaped version of current prediction vector $\boldsymbol{\xi}$ with respect to the embedding dimension e. Using the learned weight matrices $\boldsymbol{W}_{\Xi}, \boldsymbol{W}_{\boldsymbol{Y}} \in \mathbb{R}^{h \times e}$, $\boldsymbol{W}_{\boldsymbol{F}} \in \mathbb{R}^{e \times h}$, the embedded original input sample $\boldsymbol{Y} \in \mathbb{R}^{e \times d}$, and a fixed scaling parameter β the forward-pass is

$$H_f\left(\Xi\right) \;=\; \boldsymbol{W}_{\boldsymbol{F}}\, \boldsymbol{W}_{\boldsymbol{Y}}\, \boldsymbol{Y} \mathrm{softmax}\left(\beta\, \boldsymbol{Y}^T\, \boldsymbol{W}_{\boldsymbol{Y}}^T\, \boldsymbol{W}_{\Xi}\, \Xi\right). \tag{2}$$

3 Experiments

3.1 Small-Sized Tabular Datasets

Following (Klambauer et al., 2017), we consider UCI machine learning repository datasets with less than or equal to 1,000 samples as being *small*. We select 21 of these datasets and give an overview in Table A.3.

Results. Table 1 shows the median rank of all compared methods across the datasets of the UCI machine learning repository . Deep Learning methods are indicated by "(DL)" . Hopular has a median rank of 7.5, followed by Support Vector Machines with 9.5, while NPTs, XGBoost, CatBoost, and LightGBM have a median rank of 11, 12, 14, and 14.5 respectively. Hopular with modern Hopfield networks as memory performs better than other DL methods and in particular better than the closely-related NPTs. **Across the UCI datasets, Hopular is the best performing method.**

3.2 Medium-Sized Tabular Datasets

In (Shwartz-Ziv & Armon, 2021), the authors show that XGBoost outperforms various Deep Learning methods that are designed for tabular data on datasets that did not appear in the original papers. We want to know whether XGBoost still has the lead on these medium-sized datasets.

Results. Table 2 reports the results of Hopular, NPTs, XGBoost, CatBoost, and LightGBM on the medium-sized datasets. The evaluation procedure is from (Shwartz-Ziv & Armon, 2021). Hopular is

Table 1: Median rank of compared methods across the datasets of the UCI machine learning repository. Methods are ranked for each dataset according to the accuracy on the respective test set. Hopular achieves the lowest median rank of 7.5. therefore is the best performing method.

Method	**Rank**	**Method**	**Rank**
Hopular (DL)	7.5	Rule-Based Methods	15.0
Support Vector Machines	9.5	Other Ensembles	15.0
Logistic and Multinomial Regression	10.0	BatchNorm (DL)	15.0
Random Forest	11.0	Boosting Methods	15.0
Self-Normalizing Networks (DL)	11.0	Generalized Linear Models	15.5
Non-Parametric Transformers (DL)	11.0	WeightNorm (DL)	15.5
Neural Networks (DL)	11.5	Discriminant Analysis	16.0
XGBoost	12.0	Other Methods	17.5
Multivariate Adaptive Reg. Splines	12.0	ResNet (DL)	19.0
Decision Trees	13.5	LayerNorm (DL)	19.0
MSRAinit (DL)	14.0	Partial Least Squares	19.5
Bagging Methods	14.0	Bayesian Methods	20.0
CatBoost	14.0	Nearest Neighbour	24.0
LightGBM	14.5	Stacking (Wolpert)	28.0
Highway Networks (DL)	14.5		

Table 2: Results of all compared methods on the subset of medium-sized tabular datasets (Shwartz-Ziv & Armon, 2021). For classification tasks (C), the *accuracy* is reported. For regression tasks (R), the *mean squared error* multiplied by a factor of 1000 is reported. The reported deviations are the corresponding *standard error of the mean*.

Dataset		**Hopular**	**NPTs**	**XGBoost**	**CatBoost**	**LightGBM**
sulfur (R)	$\downarrow$	$\mathbf{1.04 \pm 0.02}$	1.24 ± 0.02	1.23 ± 0.00	1.06 ± 0.01	1.16 ± 0.01
colleges (R)	$\downarrow$	$\mathbf{21.18 \pm 0.09}$	25.67 ± 0.23	30.47 ± 0.00	26.40 ± 0.09	25.64 ± 0.09
eye (C)	$\uparrow$	53.56 ± 0.48	53.21 ± 0.12	$\mathbf{57.43 \pm 0.00}$	56.35 ± 0.05	57.34 ± 0.28
gesture (C)	$\uparrow$	$\mathbf{71.20 \pm 0.19}$	67.83 ± 0.06	68.05 ± 0.00	68.86 ± 0.21	69.01 ± 0.09
blastchar (C)	$\uparrow$	80.05 ± 0.11	79.98 ± 0.11	76.78 ± 0.00	$\mathbf{80.13 \pm 0.12}$	79.92 ± 0.21
shrutime (C)	$\uparrow$	86.12 ± 0.09	85.62 ± 0.07	84.58 ± 0.00	$\mathbf{86.39 \pm 0.04}$	86.18 ± 0.02

the best performing method on 3 out of the 6 datasets. The runner-up method, CatBoost, is twice the best method, whereas XGBoost once. The biggest performance difference is achieved by Hopular on the two regression datasets, where the capabilities of an external memory are especially advantageous. Directly deriving the underlying function for regression datasets may be a difficult task, especially in absence of abundant data. Hopular is able to mitigate this shortcoming by incorporating local neighbourhood information and iteratively refining its current prediction by memory lookups. Over the 6 datasets, NPTs and XGBoost are tied in last place with a median rank of 4.5. CatBoost and LightGBM have a rank of 2.5 and 2, respectively, and Hopular is in first place with a median rank of 1.5. **On average over all 6 datasets, Hopular performs better than NPTs, XGBoost, CatBoost, and LightGBM.** We also found that our method needs only a fraction of the memory compared to NPTs, which can be seen in in Section A.5 with additional investigations in Section A.6.

4 Conclusion

Hopular is a novel Deep Learning architecture where every layer is equipped with an external memory. This enables Hopular to mimic standard iterative learning algorithms that refine the current prediction by re-accessing the training set. We validated the usefulness of this property both on small- and medium-sized tabular datasets. Hopular is the best performing method across a broad selection of specifically challenging small-sized UCI datasets. Additionally, Hopular is the best-performing method on medium-sized tabular datasets among which CatBoost and LightGBM achieved very competitive results. This makes Hopular a strong contender to current state-of-the-art methods like Gradient Boosting and other Deep Learning methods specialized in small- and medium-sized datasets.

5 Acknowledgements

The ELLIS Unit Linz, the LIT AI Lab, the Institute for Machine Learning, are supported by the Federal State Upper Austria. We thank the projects AI-MOTION (LIT-2018-6-YOU-212), DeepFlood (LIT-2019-8-YOU-213), Medical Cognitive Computing Center (MC3), INCONTROL-RL (FFG-881064), PRIMAL (FFG-873979), S3AI (FFG-872172), DL for GranularFlow (FFG-871302), EPILEPSIA (FFG-892171), AIRI FG 9-N (FWF-36284, FWF-36235), AI4GreenHeatingGrids(FFG- 899943), INTEGRATE (FFG-892418), ELISE (H2020-ICT-2019-3 ID: 951847), Stars4Waters (HORIZON-CL6-2021-CLIMATE-01-01). We thank Audi.JKU Deep Learning Center, TGW LOGISTICS GROUP GMBH, Silicon Austria Labs (SAL), FILL Gesellschaft mbH, Anyline GmbH, Google, ZF Friedrichshafen AG, Robert Bosch GmbH, UCB Biopharma SRL, Merck Healthcare KGaA, Verbund AG, GLS (Univ. Waterloo) Software Competence Center Hagenberg GmbH, TÜV Austria, Frauscher Sensonic, TRUMPF and the NVIDIA Corporation.

References

Avati, A., Jung, K., Harman, S., Downing, L., Ng, A., and Shah, N. Improving palliative care with deep learning. *BMC Medical Informatics and Decision Making*, 122, 2018. doi: 10.1186/s12911-018-0677-8.

Benedetti, J. K. On the nonparametric estimation of regression functions. *Journal of the Royal Statistical Society*, 39:248–253, 1977.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152. ACM Press, Pittsburgh, PA, 1992.

Breiman, L. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. 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)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1423.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1): 3133–3181, 2014.

Friedman, J. H. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. doi: 10.1214/aos/1013203451.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. Á., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. Bootstrap your own latent - a new approach to self-supervised learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21271–21284. Curran Associates, Inc., 2020.

Ho, T. K. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pp. 278–282, 1995. doi: 10.1109/ICDAR.1995.598994.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 971–980, 2017.

Komer, B., Bergstra, J., and Eliasmith, C. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, volume 9, pp. 50. Citeseer, 2014.

Kossen, J., Band, N., Lyle, C., Gomez, A. N., Rainforth, T., and Gal, Y. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *ArXiv*, 2106.02584, 2021.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521:436–444, 2015.

Mayr, A., Klambauer, G., Unterthiner, T., Steijaert, M., Wegner, J., Ceulemans, H., Clevert, D., and Hochreiter, S. Large-scale comparison of machine learning methods for drug target prediction on chembl. *Chemical Science*, 9:5441–5451, 2018. doi: 10.1039/C8SC00148K.

Nadaraya, E. A. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964. doi: 10.1137/1109020.

Olver, F. W. J., Lozier, D. W., Boisvert, R. F., and Clark, C. W. *NIST handbook of mathematical functions*. Cambridge University Press, 1 pap/cdr edition, 2010. ISBN 9780521192255.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., Kreil, D., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. Hopfield networks is all you need. *ArXiv*, 2008.02217, 2020.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., Kreil, D., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. Hopfield networks is all you need. In *9th International Conference on Learning Representations (ICLR)*, 2021. URL `https://openreview.net/forum?id=tL89RnzIiCd`.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003.

Schölkopf, B. and Smola, A. J. *Learning with kernels - Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, 2002.

Shen, C. and Li, H. On the dual formulation of boosting algorithms. *IEEE transactions on pattern analysis and machine intelligence*, 32:2216–2231, 2010. doi: 10.1109/TPAMI.2010.47.

Shwartz-Ziv, R. and Armon, A. Tabular Data: Deep learning is not all you need. *ArXiv*, 2106.03253, 2021. URL `https://openreview.net/forum?id=vdgtepS1pV`. AutoML Workshop of International Conference on Machine Learning (ICML).

Simm, J., Klambauer, G., Arany, A., Steijaert, M., Wegner, J., Gustin, E., Chupakhin, V., Chong, Y., Vialard, J., Bujinsters, P., Velter, I., Vapirev, A., Singh, S., Carpenter, A., Wuyts, R., Hochreiter, S., Moreau, Y., and Ceulemans, H. Repurposing high-throughput image assays enables biological activity prediction for drug discovery. *Cell Chemical Biology*, 25:611–618, 2018. doi: 10.1016/j.chembiol.2018.01.015.

Somepalli, G., Goldblum, M., Schwarzschild, A., Bruss, C. B., and Goldstein, T. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. *ArXiv*, 2106.01342, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017.

Watson, G. S. Smooth regression analysis. *Sankhya: The Indian Journal of Statistics, Series A (1961-2002)*, 26(4):359–372, 1964.

Weinberger, K. Q. and Tesauro, G. Metric learning for kernel regression. In Meila, M. and Shen, X. (eds.), *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pp. 612–619, San Juan, Puerto Rico, 2007. PMLR.

Widrich, M., Schäfl, B., Pavlović, M., Ramsauer, H., Gruber, L., Holzleitner, M., Brandstetter, J., Sandve, G. K., Greiff, V., Hochreiter, S., and Klambauer, G. Modern Hopfield networks and attention for immune repertoire classification. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020. ArXiv 1904.00962.

Zhang, M., Lucas, J., Ba, J., and Hinton, G. E. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems 32*, 2019a. ArXiv 1907.08610.

Zhang, X., Tang, Z., Hou, J., and Hao, Y. 3d human pose estimation via human structure-aware fully connected network. *Pattern Recognition Letters*, 125:404–410, 2019b. doi: 10.1016/j.patrec.2019.05.020.

A Appendix

A.1 Brief Review of Modern Hopfield Networks

We briefly review continuous modern Hopfield networks. Their main properties are that they retrieve stored patterns with only one update and that they have exponential storage capacity (Ramsauer et al., 2021).

We assume a set of patterns $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \subset \mathbb{R}^d$ that are stacked as columns to the matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ and a state pattern (query) $\boldsymbol{\xi} \in \mathbb{R}^d$ that represents the current state. The largest norm of a stored pattern is $M = \max_i \|\boldsymbol{x}_i\|$. Continuous modern Hopfield networks with state $\boldsymbol{\xi}$ have the energy

$$\mathrm{E} \ = \ - \beta^{-1} \ \log \left(\sum_{i=1}^{N} \exp(\beta \boldsymbol{x}_i^T \boldsymbol{\xi}) \right) + \ \beta^{-1} \log N \ + \ \frac{1}{2} \ \boldsymbol{\xi}^T \boldsymbol{\xi} \ + \ \frac{1}{2} \ M^2 \ . \tag{3}$$

For energy E and state $\boldsymbol{\xi}$, the update rule

$$\boldsymbol{\xi}^{\mathrm{new}} \ = \ f(\boldsymbol{\xi}; \boldsymbol{X}, \beta) = \ \boldsymbol{X} \ \boldsymbol{p} = \ \boldsymbol{X} \ \mathrm{softmax}(\beta \boldsymbol{X}^T \boldsymbol{\xi}) \tag{4}$$

has been proven to converge globally to stationary points of the energy E, which are almost always local minima (Ramsauer et al., 2021). The update rule Eq. (4) is also the formula of the well-known transformer attention mechanism (Vaswani et al., 2017; Ramsauer et al., 2021), therefore Hopfield retrieval and transformer attention coincide.

The *separation* Δ_i of a pattern $\boldsymbol{x}_i$ is defined as its minimal dot product difference to any of the other patterns: $\Delta_i = \min_{j, j \neq i} \left(\boldsymbol{x}_i^T \boldsymbol{x}_i - \boldsymbol{x}_i^T \boldsymbol{x}_j \right)$. A pattern is *well-separated* from the data if $\Delta_i \geq {}^2/_{\beta N} + {}^1/_{\beta} \log \left(2(N-1) N \beta M^2 \right)$. If the patterns $\boldsymbol{x}_i$ are well separated, the iterate Eq. (4) converges to a fixed point close to a stored pattern. If some patterns are similar to one another and, therefore, not well separated, the update rule Eq. (4) converges to a fixed point close to the mean of the similar patterns. This fixed point is a *metastable state* of the energy function and averages over similar patterns.

The next theorem states that the update rule Eq. (4) typically converges after one update if the patterns are well separated. Furthermore, it states that the retrieval error is exponentially small in the separation Δ_i (for the proof see (Ramsauer et al., 2021)):

Theorem A.1. *With query $\boldsymbol{\xi}$, after one update the distance of the new point $f(\boldsymbol{\xi})$ to the fixed point $\boldsymbol{x}_i^*$ is exponentially small in the separation Δ_i. The precise bounds using the Jacobian $\mathrm{J} = {}^{\partial f(\boldsymbol{\xi})}/_{\partial \boldsymbol{\xi}}$ and its value J^m in the mean value theorem are:*

$$\|f(\boldsymbol{\xi}) \ - \ \boldsymbol{x}_i^*\| \ \leq \ \|\mathrm{J}^m\|_2 \ \|\boldsymbol{\xi} \ - \ \boldsymbol{x}_i^*\| \ , \tag{5}$$

$$\|\mathrm{J}^m\|_2 \ \leq \ 2 \, \beta \, N \, M^2 \ (N-1) \ \exp(- \, \beta \, (\Delta_i \ - \ 2 \ \max\{\|\boldsymbol{\xi} \ - \ \boldsymbol{x}_i\|, \|\boldsymbol{x}_i^* \ - \ \boldsymbol{x}_i\|\} \ M)) \ . \tag{6}$$

For given ϵ and sufficiently large Δ_i, we have $\|f(\boldsymbol{\xi}) \ - \ \boldsymbol{x}_i^*\| < \epsilon$, that is, retrieval with one update. The retrieval error $\|f(\boldsymbol{\xi}) \ - \ \boldsymbol{x}_i\|$ of pattern $\boldsymbol{x}_i$ is bounded by

$$\|f(\boldsymbol{\xi}) \ - \ \boldsymbol{x}_i\| \ \leq \ 2 \ (N-1) \ \exp(- \, \beta \, (\Delta_i \ - \ 2 \ \max\{\|\boldsymbol{\xi} \ - \ \boldsymbol{x}_i\|, \|\boldsymbol{x}_i^* \ - \ \boldsymbol{x}_i\|\} \ M)) \ M \ . \tag{7}$$

The main requirement to modern Hopfield networks to be suited for tabular data is that they can store and retrieve enough patterns. We want to store a potentially large training set in every layer of a Deep Learning architecture. We first define what we mean by storing and retrieving patterns from a modern Hopfield network.

Definition A.2 (Pattern Stored and Retrieved)**.** We assume that around every pattern $\boldsymbol{x}_i$ a sphere S_i is given. We say $\boldsymbol{x}_i$ *is stored* if there is a single fixed point $\boldsymbol{x}_i^* \in \mathrm{S}_i$ to which all points $\boldsymbol{\xi} \in \mathrm{S}_i$ converge, and $\mathrm{S}_i \cap \mathrm{S}_j = \emptyset$ for $i \neq j$. We say $\boldsymbol{x}_i$ *is retrieved* for a given ϵ if iteration (update rule) Eq. (4) gives a point $\tilde{\boldsymbol{x}}_i$ that is at least ϵ-close to the single fixed point $\boldsymbol{x}_i^* \in \mathrm{S}_i$. The retrieval error is $\|\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i\|$.

As with classical Hopfield networks, we consider patterns on the sphere, i.e. patterns with a fixed norm. For randomly chosen patterns, the number of patterns that can be stored is exponential in the dimension d of the space of the patterns (for the proof see (Ramsauer et al., 2021)):

Theorem A.3. *We assume a failure probability $0 < p \leq 1$ and randomly chosen patterns on the sphere with radius $M := K \sqrt{d-1}$. We define $a := {}^2/_{d-1}(1 + \ln(2 \beta K^2 p (d-1)))$, $b := {}^{2K^2\beta}/_5$, and*

$c := {}^{b}/_{W_0(\exp(a+\ln(b)))}$, *where* W_0 *is the upper branch of the Lambert* W *function (Olver et al., 2010, (4.13)), and ensure* $c \geq \left({}^{2}/_{\sqrt{p}}\right)^{4/d-1}$. *Then with probability* $1 - p$, *the number of random patterns that can be stored is:*

$$N \ \geq \ \sqrt{p}\, c^{\frac{d-1}{4}} \ . \tag{8}$$

Therefore it is proven for $c \geq 3.1546$ *with* $\beta = 1$, $K = 3$, $d = 20$ *and* $p = 0.001$ *(*$a + \ln(b) > 1.27$*) and proven for* $c \geq 1.3718$ *with* $\beta = 1$, $K = 1$, $d = 75$, *and* $p = 0.001$ *(*$a + \ln(b) < -0.94$*).*

This theorem motivates to use continuous modern Hopfield networks for tabular data, where we want to store the training set in each layer of a Deep Learning architecture. Even for hundreds of thousands of training samples, the continuous modern Hopfield network is able to store the training set if the dimension of the pattern is large enough.

A.2 Architecture

Hopular architecture and Modern Hopfield Networks. Deep Learning could not convince so far on small tabular datasets, on the other hand iterative learning algorithms, like Gradient Boosting methods, are the best-performing methods in this domain. Therefore, we introduce a DL architecture that is able to mimic and extend these iterative algorithms by reaccessing the whole training set and refining the current prediction in each layer. Modern Hopfield Networks directly access an external memory in a content-based fashion as depicted in Eq. (4). Hopular populates this external memory in two different ways: (a) Hopular uses the training set as an external memory, and (b) Hopular uses the embedded feature representations of the original input sample as external memory. During training, retrieval from the respective memory is learned whereas the type of fixed point of the modern Hopfield network, as described in Section A.1, specifies the type of retrieved pattern. Additionally, modern Hopfield networks can retrieve patterns with only one update – see Theorem A.1.

Furthermore, their exponential storage capacity (Theorem A.3) makes it possible to retrieve patterns from external memories with even hundreds of thousands instances. Because of these properties Hopular can mimic iterative learning algorithms e.g. such based on gradient descent, boosting, or feature selection that refine the current prediction by re-accessing the training set in contrast to other Deep Learning methods for tabular data. Both NPTs and SAINT consider feature-feature and sample-sample interactions via their respective attention mechanisms which solely use the result of the previous layer. In contrast, Hopular not only uses the result of the previous layer but also the original input sample and the whole training set. This imposes a strong inductive bias. For example, our method can implement gradient boosting with a boosting step at each layer. The ability to mimic iterative learning algorithms that are known to perform specifically well on tabular data makes modern Hopfield networks a promising approach for processing tabular data. For the instantiation variant that we use for our experiments the Hopfield module H_s identifies sample-sample relations and can perform similarity searches like a nearest-neighbor search in the whole training data. In the Appendix in Section A.7 we give further intuition on how Hopular can mimic iterative learning algorithms.

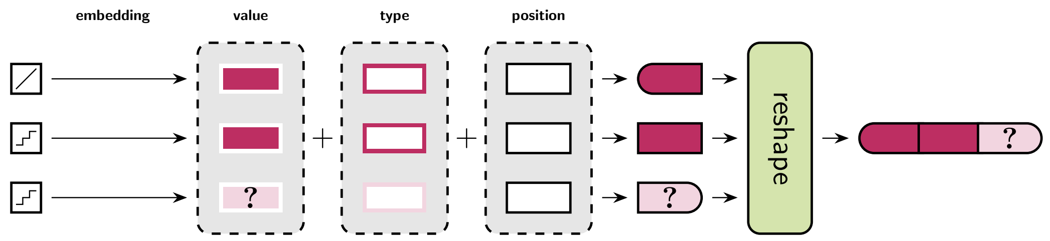

Figure A.2: Embedding Layer. All attributes of an original input sample are mapped to an e-dimensional embedding space. The position of an attribute within a sample and the attribute type are conserved by separate e-dimensional embeddings. All three embedding vectors are summed and serve as the final representation of an input attribute. The input sample is represented by the concatenation of all its attribute representations.

Hopular's Objective and Training Method. Hopular's objective is a weighted sum of the self-supervised loss for predicting masked features and the standard supervised target loss. In the following we explain the feature masking as well as the objective in more detail.

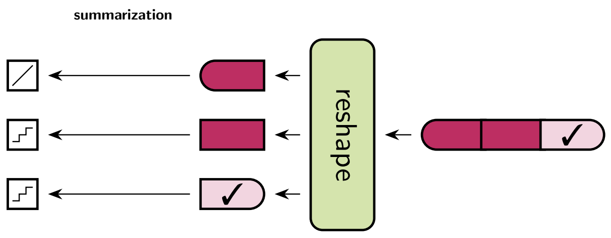

Figure A.3: Summarization Layer. The current prediction vector on the right is mapped to the final prediction vector on the left by separately mapping each current attribute prediction to its respective final prediction. This final prediction vector lives in the same space as the original input sample and is used for the computation of the respective losses.

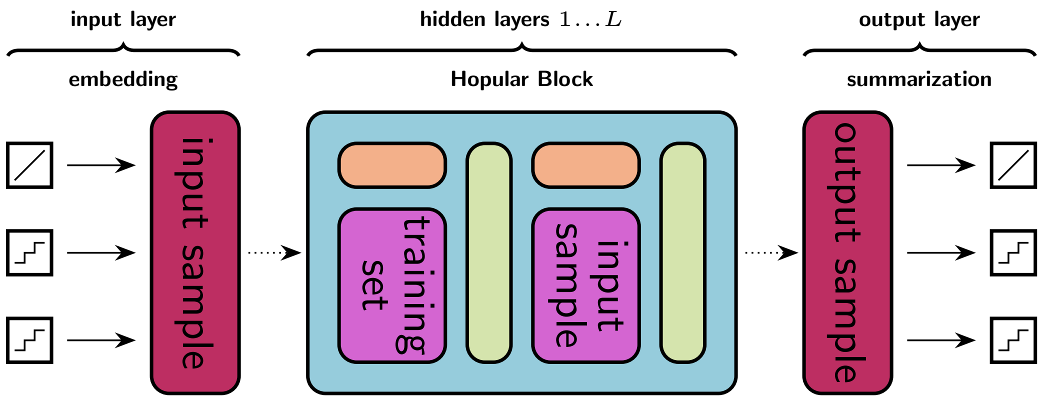

Figure A.4: Architecture overview of Hopular. Hopular consists of three different types of layers or blocks. **(I) Embedding Layer**—each attribute of an original input sample is represented in an e-dimensional space. The original input sample itself is then represented by the concatenation of all of its attribute representations. **(II) Hopular Block**—the input representation is then refined by L consecutive Hopular blocks. This is achieved by applying the two Hopfield modules H_s and H_f in an alternating way. **(III) Summarization Layer**—lastly, this refined current prediction is summarized by an attribute-wise mapping, leading to the final prediction.

Feature Masking. We follow state-of-the-art Deep Learning methods like SAINT (Somepalli et al., 2021) and Non-Parametric Transformers (NPTs) (Kossen et al., 2021) that are tailored to tabular data and use BERT masking (Devlin et al., 2019) of the input features. Masked input features must be predicted during training. Feature masking is an especially beneficial self-supervised approach when handling small datasets as it exerts a strong regularizing effect on the training procedure. The amount of masked features during training is determined by the masking probability, which is a hyperparameter of the model. In Hopular, both features and targets can be masked during training, while for inference only the target is masked.

Objective. Hopular's objective is a weighted sum of the masked feature loss L_f and the supervised target loss L_t. The overall loss L is

$$\mathrm{L} \ = \ \gamma \, \mathrm{L}_f \ + \ (1 \ - \ \gamma) \mathrm{L}_t \ , \tag{9}$$

where L_t and L_f are the negative logloss in case of discrete attributes and the mean squared error in case of continuous attributes with γ as a hyperparameter. In our default hyperparameter setting γ is annealed using a cosine scheduler starting at 1 with a final value of 0. Another essential hyperparameter for Hopular is β in Eq. (1) and Eq. (2). A small β retrieves a pattern close to the mean of the stored patterns, while a large β retrieves the stored pattern that is closest to the initial state pattern (Ramsauer et al., 2021). For module H_s a large β value emphasizes a nearest-neighbor lookup mechanics. For module H_f a large β value leads to less diluted features. Thus, large β values seem to be beneficial for Hopular. Experiments confirm this assumption (see Section 3).

Hopular Pseudocode. Algorithm 1 shows the forward pass of Hopular for an original input sample $\boldsymbol{x}$.

Algorithm 1 Forward pass of Hopular

```
Require: Hopfield modules $H_s$ and $H_f$, embedding layer $E$, summarization layer $S$, number of
    features $d$, number of Hopular blocks $L$ and original input sample $\boldsymbol{x} \in \mathbb{R}^d$
1: $\boldsymbol{x} \leftarrow \text{Mask}(\boldsymbol{x})$
2: $\boldsymbol{\xi} \leftarrow E(\boldsymbol{x})$
3: for $i = 1$ to $L$ do
4:     $\boldsymbol{\xi} \leftarrow \boldsymbol{\xi} + H_{s_i}(\boldsymbol{\xi})$
5:     $\boldsymbol{\Xi} \leftarrow \text{Reshape}(\boldsymbol{\xi}^T)$
6:     $\boldsymbol{\Xi} \leftarrow \boldsymbol{\Xi} + H_{f_i}(\boldsymbol{\Xi})$
7:     $\boldsymbol{\xi} \leftarrow \text{Reshape}(\boldsymbol{\Xi})^T$
8: end for
9: $\boldsymbol{\xi} \leftarrow S(\boldsymbol{\xi})$
```

A.3 Datasets

A.3.1 UCI Dataset Selection

To assess the performance of Hopular and other Deep Learning methods on small datasets, we select a subset of 21 datasets from (Klambauer et al., 2017). The sizes of these datasets range from 200 to 1,000 samples. We put the focus on smaller sizes, therefore we select 13 datasets with 500 samples or less. Additionally, we select four datasets with 500 to 750 samples and four dataset with 750 to 1,000 samples. Small datasets typically have small test sets, which introduce a high variance in their evaluations. This is especially true if they are overly small or unbalanced. Furthermore, some test sets seem to be not sampled iid from the whole population. Thus, the method evaluation may be highly dependent on the chosen train/test split and performance estimates may be skewed. Problematic datasets in (Klambauer et al., 2017) are characterized by having a range of accuracy values across well established methods of greater or equal 0.5 We exclude the problematic datasets *seeds*, *spectf*, *libras*, *dermatology*, *arrythmia*, and *conn-bench-vowel-deterding*. The dataset *spect* is excluded as its description in (Fernández-Delgado et al., 2014) is in conflict with the available UCI version regarding the number of attributes and samples. The dataset *heart-hungarian* is excluded as the dataset description is insufficient to distinguish between categorical and continuous attributes, which is required by some methods. Since *breast-cancer-wisc* is practically solved (0.9859 accuracy), it is excluded as it does not allow to distinguish the performances of the compared methods. We drop *heart-va*, since the best reported method has only a low accuracy of 0.4.

A.3.2 Small-Sized Dataset Description

Below we give more precise descriptions of the datasets used in our small-sized experiments:

conn-bench-sonar-mines-rocks or *Connectionist Bench (Sonar, Mines vs. Rocks)*: A classification setting of 208 instances with 60 continuous features per instance. The task is to discriminate between sonar sounds from metal vs. rocks.

glass or *Glass Identification*: A classification setting of 214 instances with 9 continuous features per instance. The task is to discriminate between 6 types of glass.

statlog-heart: A classification setting of 270 instances with 6 continuous and 7 categorical features per instance. The task is to predict the presence or absence of a heart disease.

breast-cancer: A classification setting of 286 instances with 9 categorical features per instance. The task is to predict the presence or absence of breast cancer.

heart-cleveland or *Heart Disease*: A classification setting of 303 instances with 6 continuous and 7 categorical features per instance. The task is to predict the presence or absence of a heart disease.

haberman-survival: A classification setting of 306 instances with 3 continuous features per instance. The task is to predict whether patients survived longer than 5 years or not.

Table A.3: Overview of small-sized datasets with their number of instances, number of continuous features, and number of categorical features. All small-sized datasets are classification tasks.

Dataset	**Size** (N)	**# cont. features**	**# cat. features**
conn-bench	208	60	0
glass	214	9	0
statlog-heart	270	6	7
breast-cancer	286	0	9
heart-cleveland	303	6	9
haberman-survival	306	3	0
vertebral-column2	310	6	0
vertebral-column3	310	6	0
primary-tumor	330	0	17
ecoli	336	5	0
horse-colic	368	8	19
congressional-voting	435	0	16
cylinder-bands	512	20	19
monks-2	601	6	0
statlog-australian-credit	690	5	9
credit-approval	690	6	9
blood-transfusion	748	4	1
energy-y2	768	7	0
mammographic	961	1	5
led-display	1,000	0	6
statlog-german-credit	1,000	23	0

vertebral-column2, vertebral-column3 or *Vertebral Column Dataset*: Two classification settings of 310 instances each with 6 continuous features per instance. The task is to classify patients into either 2 or 3 classes.

primary-tumor: A classification setting of 330 instances with 17 categorical features per instance. The task is to predict the class of primary tumors.

ecoli: A classification setting of 336 instances with 5 continuous and 2 categorical features per instance. The tasks is to classify proteins into 8 classes.

horse-colic: A classification setting of 368 instances with 8 continuous and 19 categorical features per instance. The task is to predict the survival or death of a horse.

congressional-voting: A classification setting of 435 instances with 16 categorical features per instance. The task is to predict political affiliation.

cylinder-bands: A classification setting of 512 instances with 20 continuous and 19 categorical features per instance. The task is to classify the band type.

credit-approval: A classification setting of 690 instances with 6 continuous and 9 categorical features per instance. The task is to determine positive or negative feedback for credit card applications.

blood-transfusion or *Blood Transfusion Service Center*: A classification setting of 748 instances with 4 continuous and 1 categorical feature per instance. The task is to predict whether a person donated blood or not.

statlog-german-credit: A classification setting of 1,000 instances with 23 continuous features per instance. The goal is to determine credit-worthiness of customers.

mammographic or *Mammographic Mass*: A classification setting of 961 instances with 1 continuous and 5 categorical features per instance. The task is to discriminate between benign and malignant mammographic masses.

led-display: A classification setting of 1,000 instances with 6 categorical features per instance. The task is to classify decimal digits from light-emiting diodes with noise.

statlog-australian-credit: A classification setting of 690 instances with 5 continuous and 9 categorical features. The task to grant customers credit-approval or not.

energy-y2 or *Energy efficiency Data Set*: A classification setting of 768 instances with 7 continuous features per instance. The task is to predict the cooling load for a given building.

monks-2 It is part of the *Monk's Problems Data Set*. A classification task for 601 instances with 6 categorical features. The task is to discriminate between two classes.

A.3.3 Medium-Sized Dataset Description

Table A.4: Medium-sized datasets with their number of instances, number of continuous features, and number of categorical features. Classification tasks are marked with (C), whereas regression tasks are marked with (R).

Dataset	**Size** (N)	**# cont. features**	**# cat. features**
blastchar (C)	7,048	3	17
colleges (R)	7,064	33	12
gesture-phase (C)	9,873	31	0
shrutime (C)	10,000	2	9
sulfur (R)	10,082	5	0
eye-movements (C)	10,936	19	3

Below we give more precise descriptions of the datasets used in our medium-sized experiments:

shrutime: A classification setting of 10,000 instances with 2 continuous and 9 categorical features per instance. The task is to predict whether a bank account is closed or not.

blastchar: A classification setting of 7,048 instances with 3 continuous and 17 categorical features per instance. The task is to predict customer behavior.

gesture or *gesture-phase* or *Gesture Phase Segmentation*: A classification setting of 9,873 instances with 31 continuous features per instance. The task is to classify gesture phases.

eye or *eye-movements*: A classification setting of 10,936 instances with 19 continuous and 3 categorical features per instance. The task is to discriminate between correct, irrelevant or relevant answers.

colleges: A regression setting of 7,064 instances with 33 continuous and 12 categorical features per instance. The task is to predict pell grant percentages for colleges in the USA.

sulfur: A regression setting of 10,082 instances with 5 continuous features per instance. The task is to predict H2S concentration in a factory module.

A.4 Hyperparameter selection process

For the hyperparameter selection process for NPTs we follow (Kossen et al., 2021) and take exactly the same hyperparameter settings that were successfully used among several datasets. We use these hyperparameter settings for experiments on small- and medium-sized datasets. For small-sized datasets we additionally use these settings with an increased embedding dimension of 128. Especially for such datasets the discrimination among similar samples can be a challenging task. This problem can be mitigated by mapping to a higher-dimensional embedding space where the samples have greater distances between each other. NPTs follow a masking procedure similar to (Devlin et al., 2019) which is realized by feature and label masking probabilities. Following the strategy in (Kossen et al., 2021) we use the LAMB (You et al., 2020) optimizer for all NPT experiments, extended by a Lookahead (Zhang et al., 2019a) wrapper with fixed values. For LAMB we use $\beta_L = (0.9, 0.999)$, $\epsilon = 1e{-}6$ and for Lookahead $\alpha = 0.5$, $k = 6$. The hyperparameter settings for NPTs are shown in Table A.5.

For a fair comparison we upper bound Hopular's capacity by the capacity of NPTs which results in the settings shown in Table A.6. As Hopular provides an additional adjustable scaling factor for β, we also test scaling factors of 100 and 1000 to further emphasize nearest-neighbor search. In our default setting the weighting term γ for our objective in Eq. (9) is annealed using a cosine scheduler starting at 1 with a final value of 0. For medium-sized datasets we also perform experiments with an initial γ value of 0.5. We use the original BERT masking as in (Devlin et al., 2019). Since we store the training data in H_s we have to make sure that the model does not just learn to retrieve the

Table A.5: Complete listing of all evaluated hyperparameter settings for NPTs. For all experiments a learning rate of 0.001 as well as a dropout probability of 0.1 is used. Settings marked with an asterisk (*) are not performed on *conn-bench-sonar-mines-rocks* due to out-of-memory issues.

dataset group	**# netw. layers**	**# att. heads**	**label mask. prob.**	**feature mask. prob.**	**learn. rate scheduler**	**emb. dim.**	
small and medium	8	8	1.0	0.15	cosine	32	
	16	8	1.0	0.15	cosine	32	
	8	16	1.0	0.15	cosine	32	
	16	16	1.0	0.15	cosine	32	
	8	8	0.1	0.15	cosine	32	
	8	8	0.5	0.15	cosine	32	
	8	8	1.0	0.20	cosine	32	
	8	8	1.0	0.15	cosine cyclic	32	
small	8	8	1.0	0.15	cosine	128	
	16	8	1.0	0.15	cosine	128	*
	8	16	1.0	0.15	cosine	128	
	16	16	1.0	0.15	cosine	128	*
	8	8	0.1	0.15	cosine	128	
	8	8	0.5	0.15	cosine	128	
	8	8	1.0	0.20	cosine	128	
	8	8	1.0	0.15	cosine cyclic	128	

Table A.6: Complete listing of all evaluated hyperparameter settings for Hopular. For all experiments a learning rate of 0.001 was used. The dropout probabilities p_i, p_h and p_o refer to the embedding layer, Hopular Block and summarization layer, respectively. The three settings of the second group (*medium-sized*) were performed in a non-exhaustive way w.r.t. to all medium-sized datasets.

dataset group	**# Hop. blocks**	**# Hop. nets**	**β-scaling factor**	**mask prob.**	**replace prob.**	**weight decay**	**dropout** p_i	p_h	p_o
small and medium	4	8	$10^{\{0,2,3\}}$	0.025	0.175	0.1	0.1	0.1	0.01
	8	8	$10^{\{0,2,3\}}$	0.025	0.175	0.1	0.1	0.1	0.01
	4	16	$10^{\{0,2,3\}}$	0.025	0.175	0.1	0.1	0.1	0.01
	8	16	$10^{\{0,2,3\}}$	0.025	0.175	0.1	0.1	0.1	0.01
medium	8	16	$10^{\{0\}}$	0.000	0.000	0.0	0.0	0.0	0.00

original input sample from the training set (like a database query). This is why we independently of BERT masking always mask the corresponding sample in the training set. We use default values for masking and dropout. For the medium-sized datasets we also test two different settings of weight decay, and of dropout probabilities in the Embedding layer, Hopular block and Summarization layer. In contrast to NPTs, we always mask all labels. In our experiments the Hopfield dimension h (as described in Section 2) is fixed by the embedding size e, the number of features d and the number of Hopfield networks M such that $h = d \cdot e/M$. The LAMB (You et al., 2020) optimizer is used for all Hopular experiments, extended by a method similar to Lookahead (Zhang et al., 2019a) but without synchronization of fast and slow weights. This is analogous to the exponential moving average used in (Grill et al., 2020). For LAMB we use $\beta_L = (0.9, 0.999)$, $\epsilon = 1e{-}6$ and for Lookahead $\alpha = 0.005$, $k = 1$. NPTs and Hopular are both trained for 10,000 epochs with early stopping.

For XGBoost and CatBoost we use the package `hyperopt` and apply the same Bayesian hyperparameter optimization procedure as described in Shwartz-Ziv & Armon (2021). For all Boosting methods we thereby evaluate 1,000 different hyperparameter settings. More precisely, the hyperparameters and their search spaces for XGBoost are defined in the following.

- *Learning rate:* Log-Uniform distribution $[-7, 0]$
- *Max depth:* Discrete uniform distribution $[1, 10]$
- *Subsample:* Uniform distribution $[0.2, 1]$
- *Colsample bytree:* Uniform distribution $[0.2, 1]$
- *Colsample bylevel:* Uniform distribution $[0.2, 1]$
- *Min child weight:* Log-Uniform distribution $[-16, 2]$
- *Alpha:* Uniform choice $\{0, \text{Log-Uniform } [-16, 2]\}$
- *Lambda:* Uniform choice $\{0, \text{Log-Uniform } [-16, 2]\}$
- *Gamma:* Uniform choice $\{0, \text{Log-Uniform } [-16, 2]\}$
- *Number of estimators:* 1000

It is important to mention that the package `hyperopt` defines the Log-Uniform distribution by the exponents of the respective interval boundaries – e.g. Log-Uniform$[-7, 0]$ is defined on $[e^{-7}, e^0]$. The hyperparameters and their search spaces for CatBoost are defined in the following.

- *Learning rate:* Log-Uniform distribution $[-5, 0]$
- *Random strength:* Discrete uniform distribution $[1, 20]$
- *Max size:* Discrete uniform distribution $[0, 25]$
- *L2 leaf regularization:* Log-Uniform distribution $[\log 1, \log 10]$
- *Bagging temperature:* Uniform distribution $[0, 1]$
- *Leaf estimation iterations:* Discrete uniform distribution $[1, 20]$
- *Number of estimators:* 1000

For LightGBM we use the default hyperparameter ranges as specified by `hyperopt-sklearn` (Komer et al., 2014).

- *Learning rate:* Log-Uniform distribution $[\log 0.0001, \log 0.5] - 0.0001$
- *Max depth:* Discrete uniform distribution $[1, 11]$
- *Number of leaves:* Discrete uniform distribution $[2, 121]$
- *Gamma:* Log-Uniform distribution $[\log 0.001, \log 5] - 0.0001$
- *Min child weight:* Log-Uniform distribution $[\log 1, \log 100]$
- *Subsample:* Uniform distribution $[0.5, 1]$
- *Colsample bytree:* Uniform distribution $[0.5, 1]$
- *Colsample bylevel:* Uniform distribution $[0.5, 1]$
- *Alpha:* Log-Uniform distribution $[\log 0.0001, \log 1]$
- *Lambda:* Log-Uniform distribution $[\log 1, \log 4]$
- *Boosting type:* Uniform choice {gbdt, dart, goss}
- *Number of estimators:* 1000

A.5 Memory footprint and runtime estimates

In table A.7 we show the memory footprint of Hopular and NPTs for all medium-sized datasets ranging from the smallest to the largest model. In all cases the whole training set is stored in the memory of module H_s. Even in the full batch setting where all the data is used as model input there is no prohibitive memory increase. In contrast, NPTs have a much higher memory memory consumption in the full batch setting. There, for 3 datasets the larger models even run out of memory on an NVIDIA A100 80GB GPU.

In table A.8 we perform measurements on training and inference times. We show the step time for medium-sized datasets during training. Inference times are assumed to be much lower, as no gradient computation and parameter updates need to be performed.

Table A.7: Memory footprint of Hopular and NPTs in *gibibytes (GiB)* for medium-sized datasets ranging from our smallest to largest model. Settings with a memory footprint of 80.00+ are not performed due to out-of-memory issues.

Dataset		Hopular		NPTs	
		single sample	full batch	single sample	full batch
sulfur (R)	↓	2.55 to 3.18	7.54 to 13.14	1.55 to 1.59	35.95 to 80.00+
colleges (R)	↓	3.13 to 3.90	6.58 to 11.62	3.98 to 6.09	27.13 to 74.56
eye-movements (C)	↓	2.68 to 3.28	10.19 to 18.21	2.11 to 2.67	45.92 to 80.00+
gesture-phase (C)	↓	2.77 to 3.41	8.92 to 15.61	2.73 to 3.90	40.95 to 80.00+
blastchar (C)	↓	2.38 to 2.75	4.83 to 7.61	1.97 to 2.38	20.49 to 56.17
shrutime (C)	↓	2.60 to 3.23	7.53 to 13.05	1.66 to 1.79	36.30 to 78.75

Table A.8: Step time of Hopular and NPTs in *milliseconds (ms)* during training.

Dataset		Hopular		NPTs	
		single sample	full batch	single sample	full batch
sulfur (R)	↓	52.71 ± 0.02	629.55 ± 0.04	59.44 ± 0.08	159.86 ± 0.28
colleges (R)	↓	120.15 ± 0.09	824.34 ± 0.17	118.13 ± 0.13	321.32 ± 0.25
eye-movements (C)	↓	76.94 ± 0.02	$1{,}141.37 \pm 0.03$	84.21 ± 0.08	338.53 ± 0.18
gesture-phase (C)	↓	95.40 ± 0.03	$1{,}155.47 \pm 0.06$	99.38 ± 0.08	384.58 ± 0.16
blastchar (C)	↓	73.69 ± 0.02	503.45 ± 0.08	81.74 ± 0.11	167.26 ± 0.25
shrutime (C)	↓	61.90 ± 0.02	652.81 ± 0.04	68.18 ± 0.08	182.11 ± 0.16

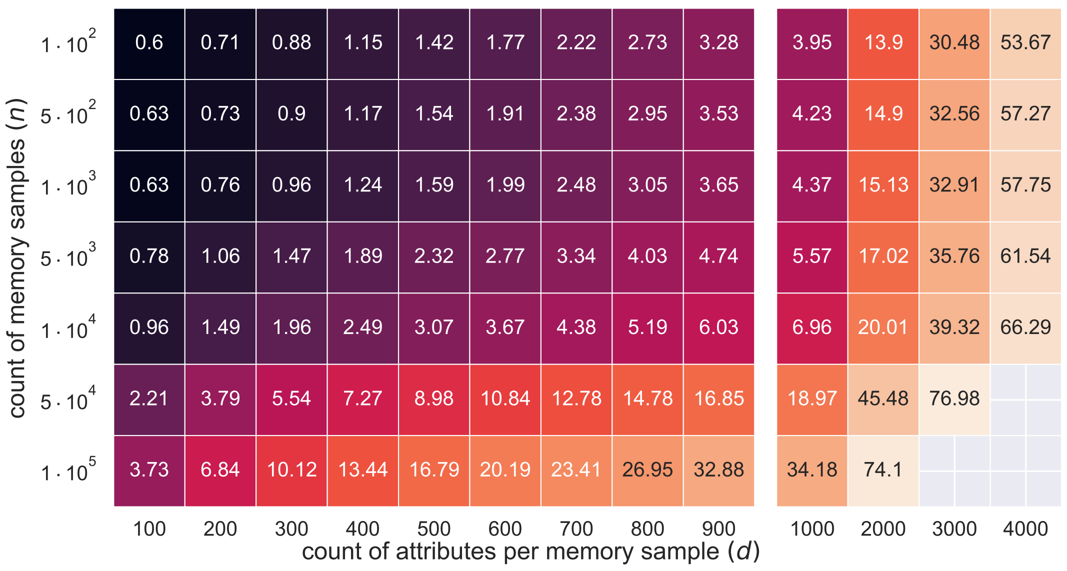

Figure A.5: Memory footprints of Hopular in *gibibytes (GiB)* measured on an artificially generated dataset. Missing entries denote settings which do not fit on an NVIDIA A100 80GB GPU, i.e. run out of memory. Best viewed in color.

A.6 Hopular: Memory footprint scaling

In figure A.5 we show the memory footprints of Hopular on the basis of an artificially generated dataset $\boldsymbol{X} \in \mathbb{R}^{(d \cdot e) \times n}$ with an attribute embedding size of $e = 8$. The used reference model consists of a single Hopular block, with a single continuous modern Hopfield network for each Hopfield module. The memory footprints are in *gibibytes (GiB)* and measured for a batch size of 1. The

Hopfield module H_s stores the complete dataset $\boldsymbol{X}$. The results clearly show that the driving factor behind the memory consumption is the attribute count d, and not the training set size n.

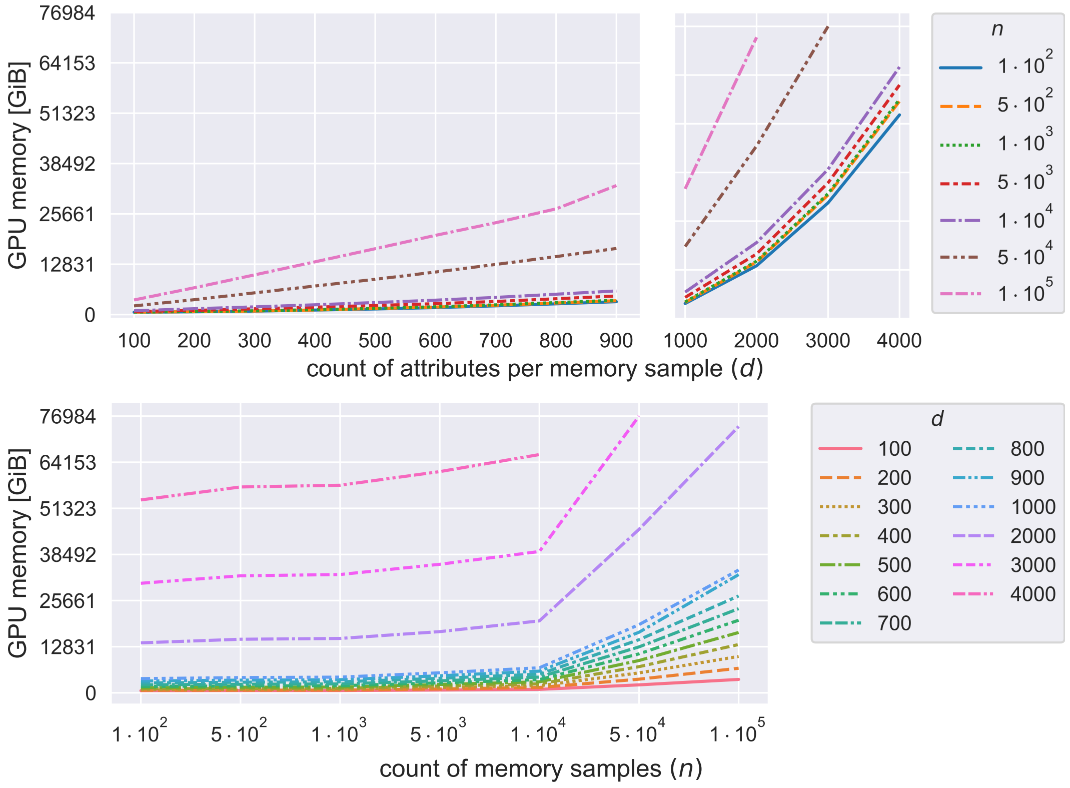

Figure A.6: Memory footprints of Hopular in *gibibytes (GiB)* measured on an artificially generated dataset with a *(TOP)* varying $d \in [100, 4000]$, and *(BOTTOM)* varying $n \in [10^2, 10^5]$. Missing entries denote settings which do not fit on an NVIDIA A100 80GB GPU, i.e. run out of memory. Best viewed in color.

A.7 Hopular Intuition: Mimicking Iterative Learning

In our first example we consider Nadaraya-Watson kernel regression (Watson, 1964; Nadaraya, 1964; Benedetti, 1977; Weinberger & Tesauro, 2007). The training set is $\{(\boldsymbol{z}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{z}_N, \boldsymbol{y}_N)\}$ with inputs $\boldsymbol{z}_i$ summarized by the input matrix $\boldsymbol{Z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N)$ and labels $\boldsymbol{y}_i$ summarized in the label matrix $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$. The kernel function is $k(\boldsymbol{z}_i, \boldsymbol{z})$. The estimator $\boldsymbol{g}$ for $\boldsymbol{y}$ given $\boldsymbol{z}$ is:

$$\boldsymbol{g}(\boldsymbol{z}) \ = \ \sum_{i=1}^{N} \boldsymbol{y}_i \ \frac{k(\boldsymbol{z}_i, \boldsymbol{z})}{\sum_{i=1}^{N} k(\boldsymbol{z}_i, \boldsymbol{z})} \ . \tag{10}$$

By using the RBF kernel we get:

$$k(\boldsymbol{z}_i, \boldsymbol{z}_j) \ = \ \exp(-\ \beta/2 \ \|\boldsymbol{z}_i \ - \ \boldsymbol{z}_j\|^2) \ = \ \exp(-\ \beta/2 \ (\boldsymbol{z}_i^T \boldsymbol{z}_i \ - \ 2 \ \boldsymbol{z}_i^T \boldsymbol{z}_j \ + \ \boldsymbol{z}_j^T \boldsymbol{z}_j)) \ . \tag{11}$$

For normalized vector $\boldsymbol{z}_i$ we have $\boldsymbol{z}_i^T \boldsymbol{z}_i = \|\boldsymbol{z}_i\|^2 = 1$, therefore

$$k(\boldsymbol{z}_i, \boldsymbol{z}_j) \ = \ \exp(-\ \beta \ (1 \ - \ \boldsymbol{z}_i^T \boldsymbol{z}_j)) \ = \ c \ \exp(\beta \ \boldsymbol{z}_i^T \boldsymbol{z}_j) \ . \tag{12}$$

We obtain for Nadaraya–Watson kernel regression with the RBF kernel and normalized inputs:

$$\boldsymbol{g}(\boldsymbol{z}) \ = \ \boldsymbol{Y} \ \mathrm{softmax}(\beta \ \boldsymbol{Z}^T \ \boldsymbol{z}) \ . \tag{13}$$

Metric learning for kernel regression learns the kernel k which is the distance function (Weinberger & Tesauro, 2007). A Hopular Block can do the same in Eq. 1 via learning the weight matrices $\boldsymbol{W}_{\boldsymbol{X}}$ and $\boldsymbol{W}_{\boldsymbol{\xi}}$. If we set in Eq. 13:

$$\boldsymbol{Z}^T = \boldsymbol{X}^T \, \boldsymbol{W}_{\boldsymbol{X}}^T, \quad \boldsymbol{z} = \boldsymbol{W}_{\boldsymbol{\xi}} \, \boldsymbol{\xi}, \quad \boldsymbol{Y} = \boldsymbol{W}_{\boldsymbol{S}} \, \boldsymbol{W}_{\boldsymbol{X}} \, \boldsymbol{X} \tag{14}$$

then we obtain Eq. 1, with the fixed label matrix $\boldsymbol{Y}$.

In the second example we show how Hopular can realize a linear model with the AdaBoost Objective. The AdaBoost objective for classification with a binary target $y \in \{-1, +1\}$ can be written as follows – see Eq. 3 and Eq. 4 in (Shen & Li, 2010):

$$\mathrm{L} \ = \ \ln \sum_{i=1}^{N} \exp(- \, y_i \, g(\boldsymbol{z}_i)) \ . \tag{15}$$

We use this objective for learning the linear model:

$$g(\boldsymbol{z}_i) \ = \ \beta \, \boldsymbol{\xi}^T \boldsymbol{z}_i \ . \tag{16}$$

The objective multiplied by β^{-1} with $\boldsymbol{Y}$ as the diagonal matrix of the targets $\{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_N\}$ becomes:

$$\mathrm{L} \ = \ \beta^{-1} \, \ln \sum_{i=1}^{N} \exp(- \, \beta \, y_i \, \boldsymbol{\xi}^T \boldsymbol{z}_i) \ = \ \mathrm{lse}(\beta \, , \, - \boldsymbol{Y} \, \boldsymbol{Z}^T \, \boldsymbol{\xi}) \ , \tag{17}$$

where lse is the log-sum-exponential function. The gradient of this objective is:

$$\frac{\partial \mathrm{L}}{\partial \boldsymbol{\xi}} \ = \ - \, \boldsymbol{Z} \, \boldsymbol{Y} \ \mathrm{softmax}(- \, \beta \, \boldsymbol{Y} \, \boldsymbol{Z}^T \, \boldsymbol{\xi}) \ . \tag{18}$$

This is Eq. 1 with:

$$\boldsymbol{Y} \, \boldsymbol{Z}^T = \boldsymbol{X}^T \, \boldsymbol{W}_{\boldsymbol{X}}^T, \quad \boldsymbol{W}_{\boldsymbol{\xi}} = \boldsymbol{I}, \quad \boldsymbol{W}_{\boldsymbol{S}} = \boldsymbol{I} \tag{19}$$

Thus, a Hopular Block can implement a gradient descent update rule for a linear classification model using the AdaBoost objective function. The current prediction $\boldsymbol{\xi}$ comes from the previous layer.

These are two additional examples among the standard iterative learning algorithms which Hopular can mimic.

A.8 Source code

Source code is available at: `https://github.com/ml-jku/hopular`