EXPLAINTABLE: EXPLAINING LARGE SCALE MODELS APPLIED TO TABULAR DATA

Javier Sanguino-Baustiste, Tim Engelmann, Natalia Pato-Montemayor & Louis Hart
Department of Computer Science
ETH Zurich
{jsanguino,tengelmann,npatomontema,lohart}@student.ethz.ch

Giulia Lanzillotta, Gregor Bachmann & Thomas Hofmann
Department of Computer Science
ETH Zurich
{giulia.lanzillotta,gregor.bachmann,thomas.hofmann}@inf.ethz.ch

ABSTRACT

The interpretability of Deep Neural Networks (DNNs) is crucial when designing reliable and trustworthy models. However, there is a lack of interpretability methods for DNNs applied to tabular data. In this short paper, we propose a novel feature importance method based on activation maximization, applicable to any Tabular Network. It allows for discarding unimportant features without a significant performance drop. We present some preliminary results for one of the largest-scale architecture in tabular data. In addition, we suggest how it can be applied to Large Language Models (LLM) to systematically study their biases too.

1 INTRODUCTION

Understanding the 'black boxes' that are DNNs is essential for building failproof, trustworthy, and unbiased tools. This is especially important in Large Scale Models when it becomes harder to systematically study the architecture and training dataset due to their size. Whilst interpretability methods for image and text networks have attracted a lot of interest, research for the tabular setting lags behind. This might be due to the reason that tree-based methods still outperform DNNs in most tabular learning problems. Nevertheless, the gap has been closing as larger-scale models with self-attention mechanisms started to emerge lately. Therefore, the need for novel tabular-specific interpretability methods is growing.

In this paper, we focus on finding the input that maximally activates a neuron of interest (e.g. the output neuron). This approach, known as activation maximization (AM), is well established in the image domain, but not sufficiently explored for tabular data. Furthermore, we identify uninformative features, by adding regularization to the optimization problem. We present preliminary results for the most large-scale architectures that exist for tabular data and make a proposal on how our methodology can be adapted to large language models to systematically study their biases.

2 LITERATURE OVERVIEW

Deep Learning is outperforming classical machine learning methods in many applications, yet, its competitiveness on tabular data remains unclear (Gorishniy et al., 2021a; Ye, 2022; Grinsztajn et al., 2022). New models based on transformers like SAINT (Kossen et al., 2022) or FT Transformer (Gorishniy et al., 2021b) have closed this gap. Interpretability methods might deliver new insights for this debate. Sahakyan et al. (2021) have created a comprehensive overview of interpretability methods tailored for tabular data. They differentiate between model-specific and model-agnostic techniques. We are specifically interested in the inner workings of DNNs and fall into the category of model-specific inspection techniques. So far, only limited research has been done in this area and the proposed methods mainly try to infer feature importance from the neuron weights (Grisci et al., 2021; Shrikumar et al., 2017; Shwartz-Ziv & Tishby, 2017; Olden & Jackson, 2002).

For image data, in contrast, there already exist well-established methods to visualize the input that specific layers or neurons respond most strongly to (Molnar, 2022; Olah et al., 2017). Erhan et al. (2009) proposed this class of activation maximization methods first. However, simply maximizing the activation of a neuron does not necessarily lead to interpretable results. Therefore, the literature following this initial proposal evolves around several different forms of regularization, such as gaussian priors, and hand-designed priors (Nguyen et al., 2019; Hada & Carreira-Perpinan, 2021). The most realistic-looking feature visualizations were produced when adding a GAN in front of the inspected model, as proposed by Nguyen et al. (2016a;b). Nevertheless, these kinds of regularization are image-specific and not all are applicable to tabular data or text.

To the best of our knowledge, the transfer of these techniques to tables is still lacking. The closest being Karpathy et al. (2015), who transferred an early version of AM to text data, but only analyzed RNN and LSTM networks.

3 METHODOLOGY

3.1 DATASETS AND MODEL

The experiments are conducted on the recently proposed benchmark for DNNs on tabular datasets by Grinsztajn et al. (2022). We restrict our AM study to binary classification on numerical features, selecting 14 corresponding datasets (we omit 1 dataset because all features were dropped in one of the experiments). For a more detailed description of the data preprocessing and training protocol, the reader is referred to Grinsztajn et al. (2022). By replicating this procedure, we achieve similar accuracy scores as reported by their work.

We conduct an in-depth analysis of the best-performing architecture, the *FT Transformer* (Gorishniy et al., 2021b). To our knowledge, this is one of the largest models in the field. Its architecture is based on transformers, similar to the ones used for large-scale models applied to images and language. For comparison, we additionally analyze the *RESNET* and *MLP* models, which are also described in the benchmark. Potentially, our method can be applied to bigger models and datasets when they emerge.

3.2 ACTIVATION MAXIMIZATION. REGULARIZATION

In its simplest form, activation maximization can be formulated as:

$$\mathbf{x}^\star = \arg\max_{\mathbf{x}} h_{l,n}(\mathbf{x}) \tag{1}$$

where x is any input to the network and $h_{l,n}(\mathbf{x})$ is the log softmax activation of the neuron n in layer l (utility function of the optimization problem). In this paper, we focus our analysis on the activations at the output layer, i.e., the aim is to study the inputs that maximize the activation for either of both classes. This can give key insights into what the model bases its decisions on.

Equation 1 has no closed-form solution and therefore a gradient method must be used. In particular, we implemented gradient ascent. Moreover, as the objective function is non-convex, it is challenging to find the global maximum. To address this issue, we produce a set of x^* (local maxima) with different initializations (Erhan et al., 2009) and use their average if the standard deviation within the set is low.

By adding a regularizer to the utility function of equation 1, the search space is constrained. Thereby, the solution can be tailored to a specific task. Particularly, we propose the following L1-regularisation term to obtain sparse solutions.

$$\mathbf{x}^\star = \arg\max_{\mathbf{x}} h_{l,n}(\mathbf{x}) + \lambda \|\mathbf{x}\|_1 \tag{2}$$

When using this penalty, features drop to zero if they do not contribute significantly to the activation of the neuron of interest.

In simpler models as logistic regression, a popular method to assess feature importance is to add sparsity to the model (Fonti & Belitser, 2017). However, if one enforces sparsity to the weights of a DNN, the activations of some neurons would become negligible, yet it is highly improbable that certain paths inside the network are always 0. Consequently, it is hard to directly assess feature importance in DNNs by enforcing sparsity in the model. Our method aims to tackle this problem by applying L1 regularization to the features of the input when performing AM rather than to the neurons, and dropping the features that present values close to 0.

To verify the effectiveness of this approach, we train models on the following three sets of features: 1) the full set (i.e. original model), 2) a subset identified through the proposed method, and 3) a set generated when dropping randomly the same number of features as in case 2). In section 4.3, we compare the accuracy of each model for the 14 chosen datasets.

4 Results and Discussion

4.1 Activation Maximization (AM)

For a given model trained on a specific dataset, we run the AM n times for each neuron in the output layer to produce a set of maxima. For our experiments, we choose $n = 100$. The obtained set of samples maximizes the log softmax activation of the neuron of interest. The feature-wise standard deviation of this set is found to be low in general. B.1 shows a boxplot of such a set of samples. For a full list of mean and max standard deviations per model, refer to tables 2 and 3. We conclude that our AM method finds roughly the same sample in each run, independently of its initialization. For the subsequent analysis, the mean of the set of samples is considered.

In Fig 1 we show two such averaged samples, each corresponding to a neuron in the output layer. An interpretation of them can give insights into what features the model bases its decision on.

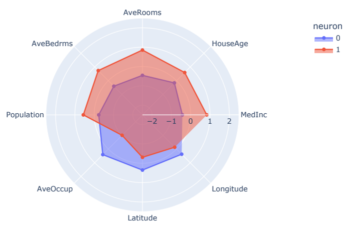


Figure 1: Mean of 100 samples maximizing the activation of the target neuron. The *FT Transformer* model trained on the *california* housing dataset (Pace & Barry, 1997) is used. Features are normally quantile transformed.

4.2 AM results across different DL architectures

When comparing the output of our AM method across different DL architectures, we find that they do not necessarily lead to the same sample. Figure 2 shows that the result of the *FT Transformer* model differs from the *MLP* and *RESNET* model. It should be noted that the architectures of the latter two models are more closely related. Consequently, the produced results are more similar.

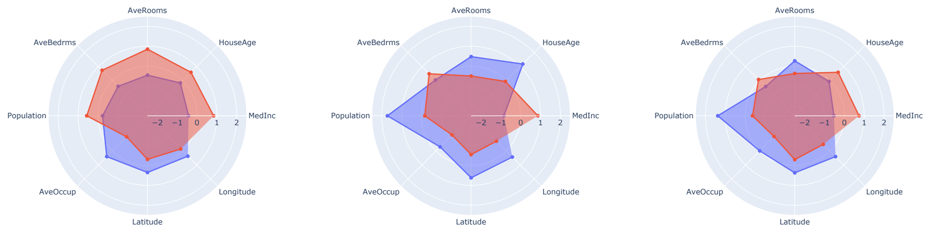


Figure 2: AM with no regularization for *FT Transformer*, *MLP* and *RESNET* models (from left to right), trained on the *california* housing dataset.

4.3 Application: Feature Selection

As outlined in section 3.2 and shown in B.3, adding L1 regularization to the AM draws uninformative features to zero. We propose to use this result to perform feature selection. In particular, we drop all features whose obtained sample value (x^*, as determined by the AM method) is below a given threshold for both output neurons. We run our experiments with a threshold of 0.1 and 0.005. Table 1 presents a summary of the accuracy on the test set for the *FT Transformer* when retraining the model with and without the identified features. The accuracy of the model stays approximately constant when omitting the identified features. A significantly larger decrease is seen when dropping a random set of features. This argues in strong favor of the effectiveness of our proposed approach. Particularly interesting are cases such as the *california* housing dataset, where with a 50% feature drop the accuracy decrease is $< 1\%$.

Table 1: Comparison of *FT Transformer* accuracies: trained on *full* set of features, omitting *our* identified features and omitting the same number of *random* features. Results are shown for both thresholds. The percentage of features dropped is given in the omit column. When choosing a low threshold (0.005) fewer features are dropped. Explicitly mentioned results are highlighted in bold.

Dataset	***Full***	**Threshold 0.1**			**Threshold 0.005**		
		Omit	***Ours***	***Random***	**Omit**	***Ours***	***Random***
jannis	0.8014	72%	0.7816	0.7588	70%	0.7827	0.7628
Higgs	**0.7280**	67%	**0.7217**	**0.6505**	50%	**0.7278**	**0.6765**
pol	0.9906	58%	0.9703	0.8919	54%	0.9745	0.8867
california	**0.8917**	50%	**0.8846**	**0.7914**	50%	**0.8850**	**0.7912**
house_16H	0.8853	44%	0.8602	0.8588	44%	0.8652	0.8613
MiniBooNE	0.9509	38%	0.9432	0.9355	32%	0.9426	0.9369
kdd_ipums_la_97	0.8908	35%	0.8890	0.8908	30%	0.8927	0.8936
MagicTelescope	0.8694	20%	0.8388	0.7680	0%	0.8619	0.8619
covertype	0.9040	20%	0.9006	0.8933	0%	0.9024	0.9024
credit	0.7785	10%	0.7789	0.7400	0%	0.7798	0.7798
phoneme	0.8921	0%	0.8441	0.8441	0%	0.8351	0.8351
electricity	0.8132	0%	0.8078	0.8078	0%	0.8094	0.8094
bank-marketing	0.7984	0%	0.7916	0.7916	0%	0.7912	0.7912
eye_movements	0.5847	0%	0.5879	0.5879	0%	0.5829	0.5829

When the datasets are of bigger size (i.e. high number of features and samples such as in the *Higgs* dataset), more features can be removed without a significant accuracy difference. This is due to the fact that even though a high percentage of features is omitted, the remaining ones still account for most of the variability present in the data and thus the accuracy remains unchanged. In these scenarios, there is a more noticeable difference in performance with respect to the random case.

For a more detailed presentation of the results and a comparison to the *RESNET* and *MLP* architectures, the reader is referred to Tables 4 and 5 in the appendix. The two more complex architectures (i.e. *RESNET* and *FT Transformer*) show a lower decrease in accuracy when dropping the identified features than the MLP architecture. Their complexity allows them to learn successfully with less amount of data. In smaller datasets, such as *bank-marketing* or *covertype*, this effect does not come to play, as a classifier requires a minimum of variability in its features to achieve reasonable accuracy.

5 Conclusion

Through this work, we successfully transfer the activation maximization method to one of the largest models applied to tabular data. Moreover, we present a promising application by designing a novel feature selection method. It analyzes feature importance in DNNs based on the activation maximization results. We demonstrate that when retraining a model without the identified uninformative features, the model accuracy decreases only slightly. However, when dropping the same number of random features, we can report a much more significant performance decrease. This difference highlights the relevance of our proposed feature selection method.

So far we have focused on binary classification for the sake of coherence but this method could be easily applied to multi-class classification or regression problems. Also, as demonstrated throughout the paper, our approach is model- and dataset-independent. It could, therefore, be used to study even more complex and larger DNNs once they are applied to tabular data. While we focused on the output layer, the inner workings of such a model can be analyzed as well. Doing so could lead to new insights into the interplay of DNNs and tabular data and might allow the development of sounder model architectures.

5.1 Potential future work: Transfer to Large Language Models

Previous work has explored how tabular data can be converted into text (Borisov et al., 2022) by describing the table as a sentence e.g. a row is represented by "*Feature1* is x, *Feature2* is y...". A pretrained LLM with a classification head can easily be finetuned to perform classification on the presented datasets. Nevertheless, we the AM method would have to be adapted: instead of using lasso, group lasso has to be applied, where one group is a sentence describing the value of one feature. In addition to that, direct gradient ascent cannot be performed due to the discrete nature of tokens, therefore, group lasso has to be applied to a mask that encodes the importance of the tokens.

Transferring the methodology to text could be useful for discovering biases of the large language model. LLM embody their own knowledge of the world and biases on the features description. We could assess what kind of features the model does not take into account when making a prediction. Our method transferred to LLM can be used to study their biases in a more systematic and automatized way.

References

Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. Language models are realistic tabular data generators, 2022.

Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47: 547–553, 11 2009. ISSN 01679236. doi: 10.1016/J.DSS.2009.05.016.

Dumitru Erhan, Y Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*, 12 2009.

Valeria Fonti and Eduard Belitser. Feature selection using lasso. *VU Amsterdam research paper in business analytics*, 30:1–25, 2017.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021a.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021b.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint*, 2022. doi: 10.48550/arxiv.2207.08815.

Bruno Iochins Grisci, Mathias J. Krause, and Marcio Dorn. Relevance aggregation for neural networks interpretability and knowledge discovery on tabular data. *Information sciences*, 559:111–129, 2021. ISSN 0020-0255. doi: 10.1016/j.ins.2021.01.052.

Suryabhan Singh Hada and Miguel A. Carreira-Perpinan. Sampling the "inverse set" of a neuron. In *2021 IEEE International Conference on Image Processing (ICIP)*, pp. 3712–3716, 2021. doi: 10.1109/ICIP42928.2021.9506500.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint*, 2015. doi: 10.48550/arxiv.1506.02078.

Jannik Kossen, Neil Band, Clare Lyle, Aidan N. Gomez, Tom Rainforth, and Yarin Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning, 2022.

Christoph Molnar. *Interpretable Machine Learning*. bookdown, 2 edition, 2022. URL `https://christophm.github.io/interpretable-ml-book`.

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29, 2016a.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint*, 2016b. doi: 10.48550/arxiv.1602.03616.

Anh Nguyen, Jason Yosinski, and Jeff Clune. *Understanding Neural Networks via Feature Visualization: A Survey*, pp. 55–76. Springer International Publishing, Cham, 2019. ISBN 978-3-030-28954-6. doi: 10.1007/978-3-030-28954-6_4. URL `https://doi.org/10.1007/978-3-030-28954-6_4`.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2:e7, 11 2017. ISSN 2476-0757. doi: 10.23915/DISTILL.00007. URL `https://distill.pub/2017/feature-visualization`.

Julian D Olden and Donald A Jackson. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154: 135–150, 2002. ISSN 0304-3800. doi: 10.1016/S0304-3800(02)00064-9.

R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics and Probability Letters*, pp. 291–297, 1997. URL `https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html`.

Maria Sahakyan, Zeyar Aung, and Talal Rahwan. Explainable artificial intelligence for tabular data: A survey. *IEEE access*, 9:135392–135422, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3116481.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International conference on machine learning*, pp. 3145–3153. PMLR, 2017.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint*, 2017. doi: 10.48550/arxiv.1703.00810.

Andre Ye. Deep learning is not 'worse' than trees on tabular data — by andre ye — towards data science, 9 2022. URL `https://towardsdatascience.com/deep-learning-is-not-worse-than-trees-on-tabular-data-1de25ed31d2`.

A APPENDIX

A.1 MAXIMIZING SAMPLE WITHIN THE DATASET

As a comparison, we also find the samples within the dataset that maximize the activation of each of the neurons in the output layer. In that case, the optimization objective is given by:

$$\mathbf{x}^\star = \underset{\mathbf{x}\in X_{train}}{\arg\max}\, h_{l,n}(\mathbf{x}) \tag{3}$$

The reader is referred to figure B.2 for a visualization of the maximizing samples within the *california* housing dataset. The samples show similarity to the ones obtained through AM (see Figure 1)(i.e. high value in the MedInc, HouseAge, and AveRooms features for neuron 1 and negative values for the MedInc and HouseAge for neuron 0). Nonetheless, to enforce the generation of in-distribution data points, one could potentially add a GAN to the AM method.

A.2 Effect of Lasso regularization parameter

When running AM with lasso regularization, the parameter λ has to be chosen. We show that increasing λ pulls the feature values to 0 as expected. Refer to B.3 for a visualization of that effect. Adding some regularization makes it easier to spot the more relevant features, however, adding too much will force all features to 0. We use a value of $\lambda = 0.001$ for our experiments.

B Supplementary Figures and Tables

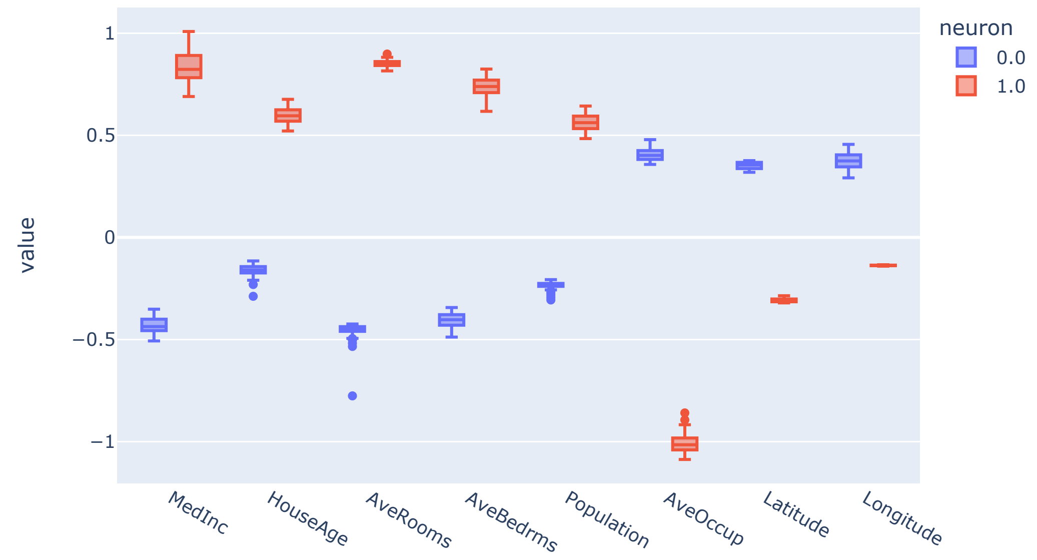


Figure B.1: Boxplot showing the 100 obtained samples maximizing the activation of the target neurons. Here we show the results for the *FT Transformer* model trained on the *california* housing dataset Pace & Barry (1997). Features are normally quantile transformed.

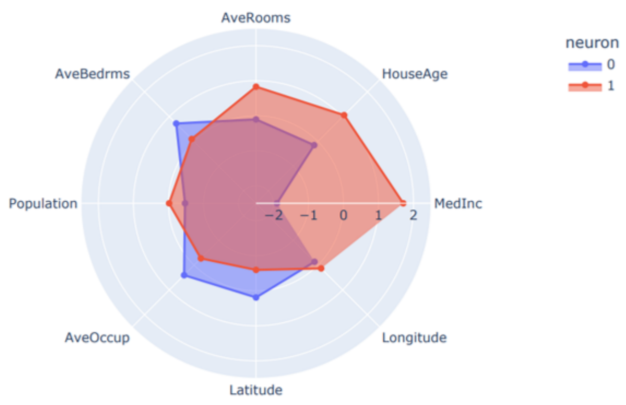
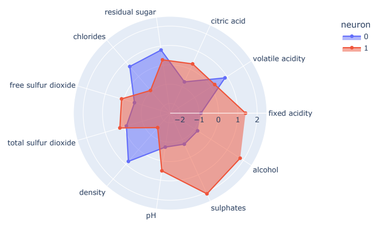
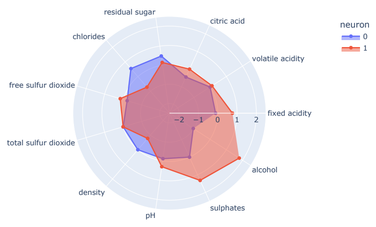


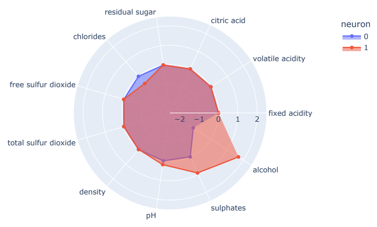
Figure B.2: Samples within the training *california* housing dataset that maximize the last layer of the *FT Transformer* model. Features are normally quantile transformed.



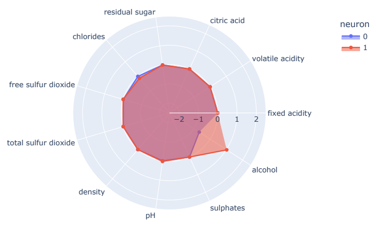
(a) $\lambda = 0$



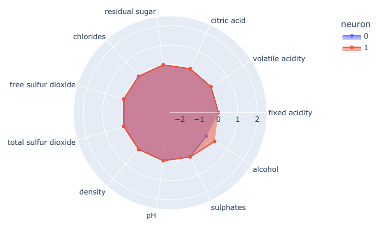
(b) $\lambda = 0.001$



(c) $\lambda = 0.01$



(d) $\lambda = 0.1$



(e) $\lambda = 1$

Figure B.3: Effect of choice of regularization parameter λ. Here we show the results for the *FT Transformer* model trained on the *wine* dataset (Cortez et al., 2009). Features are normally quantile transformed.

Table 2: Mean standard deviation across features (per neuron), shown for each dataset, model, and form of regularization.

	FT Transformer		**MLP**		**RESNET**	
Dataset	**no reg**	**lasso**	**no reg**	**lasso**	**no reg**	**lasso**
bank-marketing	0.0133	0.0345	0.2898	0.3109	0.0134	0.0095
california	0.0301	0.0299	0.027	0.0079	0.035	0.0288
credit	0.0189	0.0062	0.019	0.0117	0.8012	0.5355
electricity	0.025	0.0064	0.1830	0.0906	0.1130	0.0846
eye_movements	0.0148	0.0066	0.6935	0.1935	0.2270	0.1196
house_16H	0.0146	0.0050	0.0711	0.0338	0.0136	0.0073
jannis	0.0406	0.0053	0.1164	0.0271	0.0556	0.0087
kdd_ipums_la_97	0.0212	0.0148	0.3562	0.1678	0.0638	0.0237
MagicTelescope	0.1274	0.03315	0.1123	0.1013	0.1123	0.1126
phoneme	0.0207	0.0424	0.0887	0.0589	0.2367	0.2111
pol	0.0673	0.0097	0.0426	0.0055	0.1873	0.1448
wine	0.0127	0.0132	0.0100	0.0060	0.0192	0.0112
covertype	0.1764	0.0944	0.0513	0.0747	0.1376	0.0455
Higgs	0.0641	0.0169	0.1164	0.047	0.3002	0.0757
MiniBooNE	0.0384	0.0093	0.0410	0.0302	0.0375	0.0238

Table 3: Max standard deviation across features (per neuron), shown for each dataset, model, and form of regularization.

	FT Transformer		**MLP**		**RESNET**	
Dataset	**no reg**	**lasso**	**no reg**	**lasso**	**no reg**	**lasso**
bank-marketing	0.0427	0.1810	1.1990	0.7403	0.0304	0.0231
california	0.0687	0.1631	0.0506	0.0152	0.0850	0.0839
credit	0.1914	0.0340	0.1128	0.0213	2.7217	1.9119
electricity	0.1984	0.0238	0.6851	0.3919	0.2387	0.2171
eye_movements	0.1399	0.0644	2.6373	0.7949	0.5978	0.4863
house_16H	0.0414	0.0238	0.5527	0.1066	0.0499	0.0216
jannis	0.4650	0.0240	0.4629	0.1735	0.2623	0.0779
kdd_ipums_la_97	0.1358	0.1451	2.9366	1.6144	0.2169	0.1152
MagicTelescope	1.8802	0.2519	0.3178	0.5808	0.7629	0.5379
phoneme	0.0427	0.1147	0.2393	0.1661	1.3351	0.8926
pol	0.5152	0.0593	0.1876	0.0494	0.3214	0.3123
wine	0.0666	0.0688	0.0111	0.0080	0.0520	0.0195
covertype	0.3095	0.2268	0.2574	0.1795	0.5036	0.2858
Higgs	0.4786	0.2210	0.6019	0.1979	1.4085	0.9185
MiniBooNE	0.5711	0.0361	0.2654	0.5728	0.5762	0.5509

Table 4: Comparison of *RESNET* accuracies: trained on *full* set of features, omitting *our* identified features and omitting the same number of *random* features. Results are shown for both thresholds. The percentage of features dropped is given in the omit column. When choosing a low threshold (0.005) fewer features are dropped.

Dataset	***Full***	**Threshold 0.1**			**Threshold 0.005**		
		Omit	***Ours***	***Random***	**Omit**	***Ours***	***Random***
jannis	0.8081	76%	0.7737	0.7406	74%	0.7802	0.7412
Higgs	0.7295	58%	0.7273	0.6749	49%	0.7300	0.6770
MiniBooNE	0.9483	52%	0.9401	0.9312	40%	0.9421	0.9361
kdd_ipums_la_97	0.8963	50%	0.8789	0.8807	20%	0.8798	0.8798
house_16H	0.8934	31%	0.8761	0.8662	31%	0.8761	0.8662
california	0.8873	25%	0.8873	0.8342	0%	0.8771	0.8771
pol	0.9490	11%	0.9533	0.9315	0%	0.9504	0.9504
covertype	0.8904	10%	0.8907	0.8865	0%	0.8842	0.8842
MagicTelescope	0.8712	10%	0.8641	0.8338	0%	0.8648	0.8648
eye_movements	0.5616	5%	0.5704	0.5353	0%	0.5566	0.5566
phoneme	0.9085	0%	0.8426	0.8426	0%	0.8426	0.8426
electricity	0.8114	0%	0.8026	0.8026	0%	0.8026	0.8026
bank-marketing	0.7871	0%	0.7858	0.7858	0%	0.7858	0.7858
credit	0.7770	0%	0.7723	0.7723	0%	0.7723	0.7723

Table 5: Comparison of *MLP* accuracies: trained on *full* set of features, omitting *our* identified features and omitting the same number of *random* features. Results are shown for both thresholds. The percentage of features dropped is given in the omit column. When choosing a low threshold (0.005) fewer features are dropped.

Dataset	***Full***	**Threshold 0.1**			**Threshold 0.005**		
		Omit	***Ours***	***Random***	**Omit**	***Ours***	***Random***
wine	-	100%	-	-	100%	-	-
jannis	0.8004	83%	0.7354	0.7213	19%	0.7655	0.7718
pol	0.9533	77%	0.9381	0.8423	69%	0.9429	0.8593
MiniBooNE	0.9504	56%	0.9344	0.9286	34%	0.9406	0.9353
Higgs	0.7095	54%	0.7016	0.6716	13%	0.7106	0.6968
covertype	0.8522	50%	0.4984	0.4984	50%	0.4984	0.4984
credit	0.7768	20%	0.5104	0.5104	0%	0.7809	0.7809
kdd_ipums_la_97	0.8917	15%	0.8624	0.8881	10%	0.8817	0.8780
eye_movements	0.5960	15%	0.5666	0.5385	5%	0.5785	0.5735
california	0.8768	13%	0.8736	0.8411	13%	0.8736	0.8411
MagicTelescope	0.8591	10%	0.8527	0.8196	10%	0.8527	0.8196
house_16H	0.8761	6%	0.8659	0.8659	6%	0.8659	0.8659
phoneme	0.8606	0%	0.8276	0.8276	0%	0.8276	0.8276
electricity	0.8136	0%	0.8056	0.8056	0%	0.8056	0.8056
bank-marketing	0.7916	0%	0.7862	0.7862	0%	0.7862	0.7862