Transfer Learning for Tabular Data

Transfer Learning for Tabular Data

Leonid Joffe
Silo AI
Helsinki, Finland
leo.joffe@gmail.com

ABSTRACT

Deep learning models for tabular data are restricted to a specific table format. Computer vision models, on the other hand, have a broader applicability; they work on all images and can learn universal features. This allows them to be trained on enormous corpora and have very wide transferability and applicability.

Inspired by these properties, this work presents an architecture that aims to capture useful patterns across arbitrary tables. The model is trained on randomly sampled subsets of features from a table, processed by a convolutional network. This internal representation captures feature interactions that appear in the table. Experimental results show that the embeddings produced by this model are useful and transferable across many commonly used machine learning benchmarks datasets. Specifically, that using the embeddings produced by the network as additional features, improves the performance of a number of classifiers.

KEYWORDS

Tabular Data, Transfer Learning, Feature Engineering, Deep Neural Networks

1 INTRODUCTION

Humans' visual apparatus is very effective at learning and recognising patterns in the natural world. For instance, we will instantly recognise an apple when we see one (Fig. 1). But what is it about that image that makes us recognise the "appleness"? It is probably some collection of key shapes like the round shape, the twig on top and the little leaf, their arrangement relative to each other... We have learned to recognise and associate these visual patterns with an apple. A machine learning (ML) model that recognises apples would similarly learn patterns in data that represent an apple. In a way, learning – whether machine or human – is really a matter of transforming data in a way that reveals patterns, so that they can be identified, generalised and abstracted, learned, and used for inference.

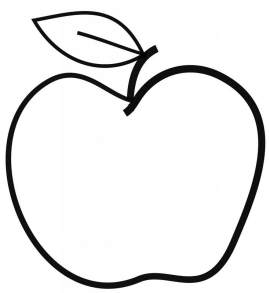

Figure 1: (A collection of contours that clearly represents) an apple.

Let's consider an example of neural networks (NNs) and how they transform data to reveal and learn patterns. As a consequence of training by backpropagation, intermediate representations (i.e. embeddings) of the "NN take on properties that make the [classification] task easier" [4]. For instance, the raw output of a binary classifier NN which takes tabular data as input, is a continuous value typically in $(0, 1)$. Applying a threshold to this value easily partitions the dataset into two classes. The representation is useful by construction because the task of the NN is literally to transform the data in a way that makes it easily separable along a single dimension; to be amenable to binary classification.

Take another example of a useful representation, from Computer Vision (CV). Some of the most impressive recent progress in CV is due to Convolutional Neural Networks (CNNs); a powerful architecture that essentially treats images as patterns of shapes. It learns to recognise corners, lines, swirls and other shapes, wherever they appear on the image (with some debate [2]). This representation happens to hit the sweet spot of transferability and applicability. That is, processing images as collections of shapes (rather than pixels) is very effective for solving various real world problems. This makes sense, since after all, the world is made of contiguous objects, rather than randomly arranged pixels!

This makes CNNs useful for CV transfer learning: reusing previous learning for a new task. First, a large CNN model is trained on a large dataset of images. The model is then taken as a base of a new network, its weights frozen, a simple classifier layer attached and trained to solve a new classification task. This method is more effective than using the classification layer on its own, so it must be the case that the output of the pretrained base network made the classification task easier. In other words, it transforms the data into a better representation than the original! These big pretrained networks are sometimes referred to as feature extractors, because they highlight salient, distinctive features of an image, universally useful for a wide range of CV tasks.

Patterns are obviously present in tabular data as well – hence ML works in the first place. Unlike CNNs that can recognise shapes in any image, deep learning models for tabular data are constrained to a specific data format. That is, if you train a model on a particular data table, inference will only work on another table that has been preprocessed in the same way, and has the same features in the same order. As a consequence, these models could not possibly learn universal patterns that could be transferred and reused across arbitrary tables.

The model proposed in this paper aims to address this problem. The architecture processes arbitrary tabular data in a consistent manner; agnostic to the number, order or type of features in the table. This work is an attempt to demonstrate that artificial neural

networks can expose and capture patterns in a wide range of domains of tabular data, so that they can be recognised, generalised and abstracted, learned and used for inference.

2 BACKGROUND

The design of the proposed model is based on CNNs. A CNN's trainable weight matrix is stored in a "filter", also referred to as a "kernel". In the familiar use case of CV, the filter is slid across an image, and it outputs a representation of interactions of adjacent pixels. Fig. 2 illustrates a convolutional layer with kernel size (field of view) of 2, and a one dimensional image of n pixels $x_1, x_2, x_3, ..., x_n$. The kernel is slid across the image one step at a time, so the features shown to the kernel are (x_1, x_2), (x_2, x_3), (x_3, x_4)..., (x_{n-1}, x_n), i.e. each following feature pair overlaps with the previous one. The output of this layer is a sequence that represents the interactions of pairs of adjacent pixels $y_{(1,2)}, y_{(2,3)}, y_{(3,4)}, ..., y_{(n-1,n)}$. Each application of the filter is a dot product, much like in a regular densely connected layer.

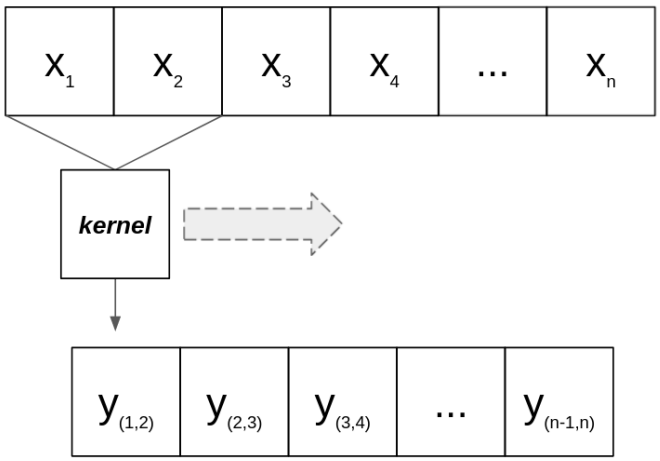

Figure 2: Illustration of a kernel sliding across a 1-d image. The inputs to the kernel are adjacent features $x_1, ..., x_n$. The outputs $y_{(1,2)}, ..., y_{(n-1,n)}$ represent the interactions of input features. This architecture is appropriate for CV tasks because learnable patterns of an image depend on relative positions of pixels.

The standard usage of CNNs described above captures relationships between *adjacent* variables. Since a feature's position in the table, contrary to pixels in an image, carries no meaning, CNNs are not applicable to tabular data out of the box. Works attempting this have shown underwhelming results: their performance is "no better than SOTA" [3] or XGboost [5].

The order of features in a table may not carry meaning, but some interactions are obviously more important than others. Consider a table with three input features $(weight, eyecolour, height)$, in that order, and a binary label of heart attack. A single layer CNN with a kernel size 2 would capture the interactions of $(weight, eyecolour)$ and $(eyecolour, height)$, whereas the key interaction of $(weight, height)$ will be missed, as shown in Fig. 3. Granted, here one could use a densely connected layer, but that model would not work on another variant of the table where the order of features was changed – hence the whole exercise of this paper.

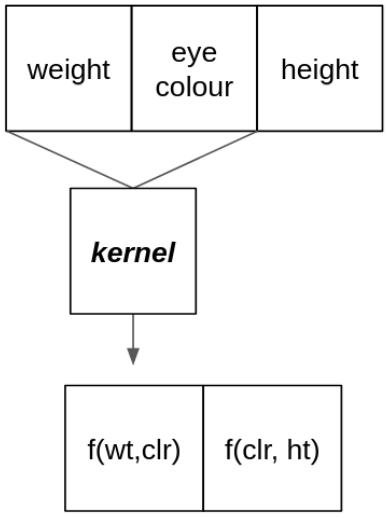

Figure 3: Sliding a kernel across a table of three features $(weight, eyecolour, height)$ captures the interactions of $(weight, eyecolour)$ and $(eyecolour, height)$. This architecture does not capture the $(weight, height)$ interaction which (for the sake of argument) is important for predicting e.g. heart attacks, so it is not fit for purpose.

One way of forcing the model to focus on certain interactions is to use attention, à la transformers [7]. In the above example, an attention mechanism would prioritise *weight* and *height* while giving less weight to *eyecolour*. If feature names are human-readable, then natural language processing (NLP) models can be used to focus certain columns based on their names. This is when an attention mechanism based on column labels works, as demonstrated by Arık and Pfister [1].

In general however, names of tabular variables are less reliable than words in NLP. If a table's variables are renamed or removed, it will still be amenable to the original task; a densely connected NN will perform just as well whether a column is called *weight* or x_1. In NLP however, renaming "*weight*" to "x_1" will break the model. So any model relying on natural language labels of features will not be very general.

To summarise, we wish to build a model that can recognise and learn patterns in arbitrary tabular data, much like CNNs do in image processing tasks. Such a model cannot be based on *i)* densely connected layers, since that is dependent on the number and order of columns. It cannot use *ii)* regular CNNs from CV either, since those models exploit order – something we specifically want to obscure. Finally, it cannot *iii)* use names of labels either, since, unlike words in NLP, labels of table columns are not inherently meaningful.

3 THE MODEL

The purpose of the proposed architecture is to produce a meaningful and convenient embedding of arbitrary tabular data. But how do you capture patterns in something that is of unknown size, type, content and order? Are there "lines" or "swirls" in tabular data? In images, these features are based on the interaction of adjacent pixels – i.e. spatially dependent features.

Much like pixels, tabular data can be represented as interactions of features, and those patterns can be learned. It is proposed that interactions of features can thus form a basis for a universal representation of tabular data. For instance, while weights of patients are not directly comparable with weights of fruit, they both represent a mass of a biological object. Odds are, there will be something similar about their behaviour, and perhaps (a variable that behaves and interacts like) size is a universally useful feature with respect to various real world classification tasks.

The model processes tables in a way that only allows it to learn from feature interactions, rather than absolute values or their position in the table. The method works by sampling tuples of features and applying a series of convolutional filters on the samples. The output is pooled into an n-dimensional vector space – an embedding. The embedding is then fed to a simple densely connected classifier, which solves the original task of the dataset. Crucially, the model is trained with data from various domains and the embedding is a representation which has been useful for solving multiple classification tasks. A high level illustration of the architecture is shown in Fig. 4, and a detailed explanation follows.

3.1 Data Scaling

The first step of the architecture is data scaling. The scaling method, shown in Eq. 1, was chosen empirically. The value in the input data X is turned into a tuple where the first element is a normalisation by column and the second one is by row. So a single data point is represented in two contexts – relative to the column and relative to the row. This gives the model some extra context and it appears to perform better.

Assump all is number already, how about date

$$X = \left(\frac{X}{X_{max\_col}}, \frac{X - X_{min\_row}}{X_{max\_row}} \right) \quad (1)$$

3.2 N-gram Sampling and Convolutions

Tabular data cannot be processed like an image since the order of columns does not carry meaning. In fact, the network should not be distracted by spurious positional interactions. That is, the network must not be allowed to learn patterns based on column position. The mechanism described below is intended to achieve this and it is at the core of the proposed architecture.

Although the order of columns has no significance, clearly input features do interact, and we wish to capture these interactions. To do this, one could consider all permutations of all sizes and use them to train convolutional kernel (of size n – for each n-gram size). This would exhaustively "show" the relationship of every feature with every other feature to the kernel. For instance, the permutations of features of a dataset with columns "A", "B" and "C" are the following:

1-gram : "A", "B", "C"
2-gram : "AB", "AC", "BC", "BA", "CA", "CB"
3-gram : "ABC", "ACB", "BAC", "BCA", "CAB", "CBA"

Unfortunately, the number of permutations grows by $O(n!)$ so exhaustive permutation quickly becomes infeasible. So instead of exhaustive enumeration of all permutations, the proposed solution is to simply sample n-grams of variables.

2-grams represent pairwise relationships, 3-gram captures interactions of triples of features, etc. In practice, the sampler draws $k * n$(-gram) indices and then takes features at those indices. In the above example, if $k = 2, n = 3$, indices could be $(0, 2, 1, 0, 2, 0)$ so the input would be "ACBACA". This sequence is then passed to a convolutional layer with kernel size n and step size n. The kernel consumes tuples "ACB", "ACA" and outputs a representation of their interactions. The output is of size k. Only work with n%2!=0????

In the current implementation, a fixed number of samples of 2, 3 and 4 -grams are drawn. Those are then processed by convolutional layers with matching kernel sizes and strides, and their outputs are concatenated along the dimension axis. So overall, the output of a sampling & convolution layer is a representation of sampled 2, 3 and 4-gram feature interactions.

3.3 Pooling and Embedding

The output of the stack of sampling convolutional layers is a sequence of random samples. Since the elements of this sequence are the result of random sampling, their order is meaningless, so the temporal dimension needs to be collapsed. Simple flattening does not work since an order is still maintained and the output becomes noise. Taking an average over the whole sequence in turn destroys too much information; the signal weakens and the model's training does not converge.

The proposed solution here is a stack of strided convolutions. Four convolutional layers of kernel size 3 and stride 2 compress the sequence of k interactions from size $(n\_channels, k)$ to a single vector of size $(2^4 * n\_channels, 1)$ (each layer also doubles the number of output channels). In the current implementation, the number of n-gram samples to draw k is 32 and $n\_channels$ is 256. This stack is followed by three densely connected layers, and finally by an embedding layer.

why better ?

To underline: Up to this point, the weights of the layers are shared across all training datasets. This is possible because the model is agnostic of the order and the number of input features.

3.4 Classifiers and Training

The embedding is fed to several simple classifiers (one for each dataset) that provide gradients for backpropagation. This makes the model end-to-end differentiable. For each training pass, a batch of data is drawn from each dataset, a forward pass and backward passes are done on all batches, gradients are accumulated and an optimiser takes a step. The main training loss is the sum of each classifier's cross entropy loss, as is standard for classification tasks. In addition, the embedding is regularised with an L2 penalty.

4 EXPERIMENTS

The quality of the embedding is assessed empirically by comparing the classification accuracy of several off-the-shelf classifiers on original vs. data embedded by the model. This is a feature extraction or a feature engineering set up, analogous to using a pretrained CNN to extract features from images, and then using those features to aid classification.

Experimental results show that the embedding has a significant positive effect on the classification accuracy. This suggests that the model has learnt feature interaction patterns, and is able to

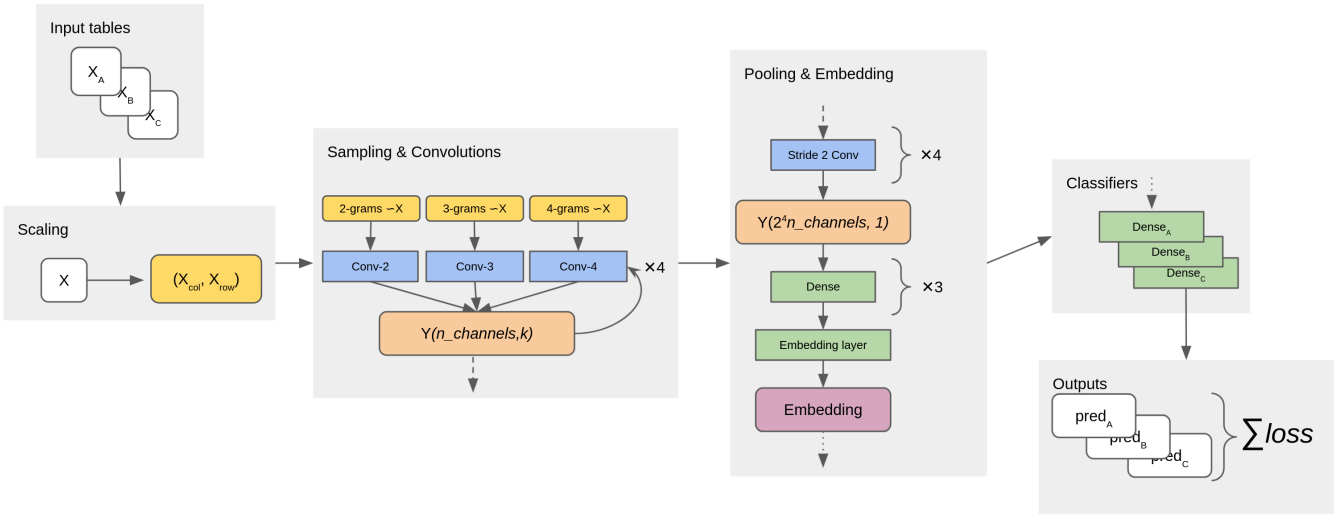

Figure 4: High level view of the proposed model, each stage is detailed in Section 3. The model consumes multiple datasets X_i as inputs. Each dataset is first normalised by column and by row, forming a 2-channel input. Then samples of n-grams of features are processed by a stack of parallel convolutional layers. Then a stack of strided convolutions compresses the temporal dimension. This is followed by several densely connected layers that produce an embedding. Finally, the embedding is consumed by multiple parallel classifiers – one per dataset.

recognise them in previously unseen data. To underline, the following is not an assessment of whether the model is particularly well suited for the use case of feature engineering, nor whether it beats classification benchmarks – but to assess whether some degree of transfer learning is achieved.

34 data sets from OpenML [6] were used for the experiments. Data sets were chosen based on popularity and convenience. Popular data sets are likely of higher quality and they will have numerous benchmark results of top performing ML pipelines. This provides a solid point of comparison.

Convenience means that the data is numeric and does not require extra pre-processing. Much of the data on openml requires reformatting, index encoding of string variables, parsing of dates etc. All of this could be done, but it is best to not include those extra steps so as not to bias the results by pre-processing. So the data sets are all numeric, although some of those numbers represent categorical variables. Statistics of the data can be seen in Table 1.

Experiments use leave-one-out principle. The model is re-trained from scratch for each dataset shown in Table 1. So each dataset is in turn treated as a test set, while the others are the training sets. After each training run, first the test data set is classified by six off the shelf classifiers with default parameters: DecisionTreeClassifier (DT), SVC, BaggingClassifier (B), GradientBoostingClassifier (GB), RandomForestClassifier (RF), SGDClassifier (SGD). Then, the data set is augmented with an embedding produced by the model and the same evaluation is run. Finally, the original data is replaced with the embedding and once again evaluated. Each evaluation is run five times and the median result is taken. Overall, a total of 3060 evaluations are performed: 34 data sets, 6 classifiers, 3 data modes (original, embedded only, combination of the two) and 5 repeated executions (to get an average).

The following hyperparameters were used throughout the experiments. These were chosen empirically, and while some qualitative observations on their effect can be made, a more comprehensive ablation study is out of scope of this paper.

- ELU activation
- Decoder width 512, dropout of 0.25
- 4 convolutional layers, of which the first random samples
- Kernel sizes of (2, 3, 4)
- 256 filters
- $k = 16$ samples
- 1000 training steps of 256 samples across all datasets
- Embedding size $4 < emb < 32$
- 5 evaluation runs each at 0.75 train-test split

5 RESULTS

The classification accuracy on data augmented with embeddings generated by the model exceeded the OpenML benchmark in 10 cases out of 34 (++). Furthermore, accuracy of another 5 datasets improved over the original data benchmarks (+). In 12 cases, the augmentation did not affect the classifiers' performance significantly. Another 2 datasets performed marginally worse, and 2 more saw a significant decrease in accuracy. Finally, 2 datasets are very simple with any representation (their benchmark score is 1.0), so those are of little interest. A full per dataset analysis of results is provided in the appendices. A summary of significant results is presented in Table 1. A more in depth, dataset wise analysis of results is presented in the appendices.

Several general observations can be made. First, a systematic experimental error seems unlikely. If this were the case, suspicious results would be consistent, e.g. the model would consistently outperform the benchmarks – both original data and openml. This is not the case. Second, some of these datasets are more similar to one another than others. For example, the *kc* datasets have the same variables and the same structure, but different labels. In other words, *kc1* and *kc2* are like a single split dataset. There are also

Name	Openml id	#rows	#features	#classes	Benchmark	Orig. data	Exp. data	
abalone	183	4177	9	28	0.281	0.251	0.255	+/-
Australian	40981	690	15	2	0.881	0.862	0.873	+/-
banknote-authentication	1462	1372	5	2	1.	0.994	0.994	**
blood-transfusion-service-center	1464	1372	5	2	0.802	0.770	0.765	+/-
climate-model-simulation-crashes	1467	540	21	2	0.930	0.933	0.933	+/-
cmc	23	1473	10	3	0.574	0.558	0.778	++
diabetes	37	768	9	2	0.788	0.755	0.771	+/-
eeg-eye-state	1471	14980	15	2	0.987	0.930	0.993	+
first-order-theorem-proving	1475	6118	52	6	0.639	0.622	0.586	–
GesturePhaseSegmentationProcessed	4538	9873	33	5	0.735	0.663	0.731	+
gina_agnostic	1038	3468	971	2	0.963	0.934	0.940	+/-
heart-h	51	294	14	2	0.850	0.689	0.689	+/-
hill-valley	1479	1212	101	2	0.765	0.888	1.	++
ilpd	1480	583	11	2	0.743	0.712	0.719	+/-
kc1	1067	2109	22	2	0.873	0.860	0.966	++
kc2	1063	522	22	2	0.870	0.832	0.893	++
mfeat-morphological	18	2000	7	10	0.772	0.714	0.958	++
mfeat-zernike	22	2000	48	10	0.907	0.826	0.932	++
monks-problems-2	334	601	7	2	1.	0.987	0.709	–
mozilla4	1046	15545	6	2	0.960	0.951	0.949	+/-
one-hundred-plants-margin	1491	1600	65	100	0.886	0.798	0.880	+
one-hundred-plants-shape	1492	1600	65	100	0.736	0.583	0.783	++
one-hundred-plants-texture	1493	1599	65	100	0.873	0.820	0.860	+
ozone-level-8hr	1487	2534	73	2	0.950	0.940	0.946	+/-
pc1	1068	1109	22	2	0.945	0.935	0.953	++
pc3	1050	1563	38	2	0.909	0.898	0.903	+
pc4	1049	1458	38	2	0.929	0.904	0.910	+/-
phoneme	1489	5404	6	2	0.923	0.907	0.878	-
qsar-biodeg	1494	1055	42	2	0.894	0.875	0.932	++
satimage	182	6430	37	6	0.929	0.914	0.904	-
spambase	44	4601	58	2	0.963	0.954	0.977	++
steel-plates-fault	1504	1941	34	2	1.	1.	1.	**
wdbc	1510	569	31	2	0.984	0.965	0.958	+/-
wilt	40983	4839	6	2	0.990	0.984	0.984	+/-

Table 1: Statistics and results of data sets used in experiments. This table shows that the model managed to greatly improve the performance of off the shelf classifiers in a number of cases. The first five columns are summary properties of the datasets. *Benchmark*, *Orig. data* and *Exp. data* -columns are accuracy measures of the best accuracy for this dataset achieved on OpenML, and the accuracy of the best off-the-shelf classifier on original and embedded data respectively. The last column shows whether the accuracy improved above the OpenML benchmark (++), improved over original data (+), made no significant difference (+/-), decreased performance (-), decreased the accuracy significantly (–) or were deemed useless (**). A more in depth analysis of results is presented in appendices.

datasets that are "twinned" in a different way. For instance, the *mfeat-* datasets have the exact same labels but very different features (*mfeat-morpological* has 6 features while *mfeat-zernike* has 47). These similarities are likely beneficial for the model's performance. Of course, this does not constitute overfitting since the model did not memorise rows of data.

Finally, the results suggest that there are universal, transferable patterns in tabular data, that they can be learnt, and this knowledge can be leveraged across data sets. To underline, more than simply outperforming a benchmark (again, this work is not intended to beat benchmarks, but to introduce the concept), these results indicate that the network was able to learn useful features from training data and usefully recognise them in a previously unseen data. In other words, with the given set of training data, the model does perform transfer learning.

6 CONCLUSION

Deep learning models for tabular data are not readily transferable from one table to another. A neural network trained on a particular table will only work on a table with the exact same features. Transfer learning however has been very successful in other deep learning fields, and this paper presents an approach for doing transfer learning on tables. The model works by sampling tuples of features from multiple input tables, encoding them and using the embedding to solve the original classification task. The weights of the encoder are shared across all input tables while the decoders are table specific. The embedding thus ought to be a universally useful representation of the input data.

Experiments show that embeddings produced by the network, when used as engineered features, can significantly aid off-the-shelf classifiers. This suggests that the model highlights features of the data that are useful for classification, i.e. it performs transfer learning.

There are various aspects to study in future work. Practical directions include running experiments on more datasets, conducting ablation studies, tuning hyperparameters, data preprocessing, and experimenting with additional features for the network such as attention and regularisation. Studying the embeddings could be another fruitful direction: what does the embedding and distances within that vector space really represent? Finally, notions of capacity of the network and sufficiency of sampling are related to information theoretic concepts and could be explored from that point of view.

A PER DATASET ANALYSIS OF RESULTS

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.197	0.197	0.213	0.207	0.191	0.2
SV	0.245	0.211	0.247	0.243	0.205	0.249
B	0.232	0.224	0.233	0.226	0.223	0.227
GB	0.239	0.216	0.245	0.248	0.213	0.252
RF	0.246	0.243	0.244	0.255	0.232	0.243

Table 2: abalone (openml best *0.281*): No significant difference, all variants of data about the same performance, within 5% of benchmark. No difference with LReLU. The dataset has 1 categorical and 7 numeric features, and 28 classes. Labels are very imbalanced.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.780	0.688	0.821	0.798	0.659	0.809
SV	0.867	0.803	0.867	0.855	0.775	0.861
B	0.861	0.757	0.850	0.855	0.734	0.855
GB	0.867	0.786	0.879	0.855	0.775	0.85
RF	0.861	0.792	0.873	0.873	0.78	0.85
SGD	0.838	0.699	0.838	...	...	...

Table 3: australian (openml best *0.881*): Augmentation has no significant effect, results within 3% of original. The best was the augmented dataset running with RF, came within 1% of the top benchmark. No significant difference with LReLU. This dataset has 14 input features of which half are categorical. Some of both the numeric and categorical features appear to be heavily skewed. There are 2 classes of labels. This is a smallish dataset with only 690 rows.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.985	0.93	0.988	0.983	0.878	0.983
SV	0.994	0.875	0.994	0.994	0.875	0.994
B	0.983	0.939	0.991	0.985	0.892	0.985
GB	0.991	0.939	0.997	0.994	0.889	0.994
RF	1.000	0.956	0.994	0.991	0.913	0.994
SGD	0.985	0.799	0.988	...	...	...

Table 4: banknote-authentication (openml best *1.0*): All the same with augmentation, within 3%. The benchmark peak is 1.0, so the dataset is not all that interesting. The dataset has 4 reasonably well balanced numeric input features, and 2 decently balanced classes.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.738	0.706	0.690	0.722	0.733	0.717
SV	0.765	0.759	0.765	0.749	0.749	0.749
B	0.749	0.738	0.738	0.759	0.738	0.733
GB	0.775	0.77	0.749	0.759	0.733	0.770
RF	0.759	0.759	0.743	0.765	0.754	0.749
SGD	0.749	0.759	0.749	...	...	...

Table 5: blood-transfusion-service-center (openml best *0.802*): All the same with augmentation, within 3%. No significant difference with LReLU. The performance hit of pure embeddings is somewhat less than in most other datasets. This dataset has 4 input features and 2 target classes. Although the inputs are listed as numerical, there are very few unique values. That is to say, although the numeric values' magnitude might be significant, they are not continuous. This might have an effect of the model's performance; e.g. the capacity may be too large, or the discrete values could impose some sort of sparsity in the internal representation.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.874	0.837	0.881	0.881	0.830	0.867
SV	0.933	0.933	0.933	0.911	0.911	0.911
B	0.874	0.911	0.889	0.896	0.881	0.896
GB	0.904	0.919	0.904	0.904	0.904	0.911
RF	0.926	0.933	0.926	0.911	0.911	0.911
SGD	0.933	0.933	0.933	...	...	...

Table 6: climate-model-simulation-crashes (openml best *0.930*): All the same with augmentation, within 3%. Again, pure embedding fares surprisingly well. No significant difference with LReLU. This dataset has 1 categorical variable and 19 numerical ones. Of those 19, 17 are in the 0.-1. range. There are 2 very skewed target labels. The dataset is quite small, with only 540 rows.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.591	0.558	0.469	0.672	0.640	0.474
SV	0.518	0.640	0.528	0.499	0.675	0.512
B	0.631	0.607	0.518	0.734	0.718	0.534
GB	0.672	0.623	0.561	0.748	0.737	0.558
RF	0.637	0.642	0.537	0.778	0.745	0.526
SGD	0.425	0.634	0.458	...	...	...

Table 7: cmc (openml best *0.574*): A significant improvement in performance. Interestingly, augmented does best with RF, but pure embedding does better than augmentation (i.e. it cleared out noise?) for SVC and SGD. LReLU performs considerably worse across the board, particularly with RF. This dataset has 9 features of which 7 are categorical. Interesting, why did the model work so well here...

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.688	0.589	0.708	0.688	0.630	0.677
SV	0.760	0.661	0.760	0.750	0.724	0.755
B	0.734	0.630	0.734	0.740	0.688	0.755
GB	0.745	0.620	0.771	0.755	0.667	0.755
RF	0.750	0.667	0.755	0.771	0.688	0.755
SGD	0.646	0.609	0.641	...	...	...

Table 8: diabetes (openml best *0.788*): All the same with augmentation, within 3%. The best result is achieved by the augmented dataset running with RF 3% of the top benchmark. LReLU performs somewhat worse. 8 numeric, discrete input variables, binary classification. Quite small, with only 768 rows. The labels are quite well balanced though.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.969	0.969	0.831	0.983	0.984	0.821
SV	0.547	0.911	0.547	0.563	0.944	0.563
B	0.98	0.974	0.898	0.989	0.988	0.895
GB	0.95	0.938	0.811	0.971	0.968	0.814
RF	0.987	0.982	0.923	0.993	0.991	0.93
SGD	0.551	0.675	0.468	...	...	...

Table 9: eeg-eye-state (openml best *0.987*): A significant improvement in performance reaching within 1% of the benchmark. Pure embedding doesn't do any worse than the augmentation. No significant difference with LReLU. This dataset has 14 continuous numeric input features. The data is quite dense, with an order of 200-600 unique features out of 14980 rows. The binary labels are quite well balanced.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.492	0.411	0.547	0.497	0.411	0.544
SV	0.533	0.448	0.533	0.531	0.452	0.531
B	0.554	0.461	0.593	0.547	0.461	0.606
GB	0.56	0.468	0.58	0.546	0.467	0.57
RF	0.595	0.501	0.622	0.586	0.51	0.622
SGD	0.408	0.409	0.395	...	...	...

Table 10: first-order-theorem-proving (openml best *0.639*): A consistent decrease of performance over even the original data. LReLU performs marginally better, but still worse than original data. This dataset has 51 numeric features, of which 17 have fewer than 150 unique values in 6118 rows of data. The 6 target labels are decently except for one category accounting for over 40% of the data. This imbalance might skew the performance; could perhaps be corrected by some sort of a class-balanced sampling mechanism in training. Then again, the class weighting on the classifier output does that, in a way. That said, perhaps this signal gets lost due to the accumulation of gradients.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.555	0.477	0.515	0.573	0.505	0.5
SV	0.615	0.547	0.53	0.655	0.562	0.535
B	0.661	0.577	0.611	0.674	0.604	0.607
GB	0.632	0.517	0.588	0.64	0.514	0.595
RF	0.716	0.641	0.67	0.731	0.677	0.663
SGD	0.45	0.318	0.438	...	...	...

Table 11: GesturePhaseSegmentationProcessed (openml best *0.735*): An improvement over original data to the tune of 10-20%. With augmentation and RF, the model gets to within 1% of benchmark. LReLU consistently worse, particularly for DT and SV. This dataset has 32 floating numerical features and 5 reasonably well distributed target labels. This dataset is "preprocessed" *vs.* "RAW", which is also available on openML (it however is far less popular and needs preproccessing). So it may be that preprocessing has a bias on the performance of the architecture, just not sure how...

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.843	0.526	0.848	0.857	0.536	0.855
SV	0.925	0.602	0.925	0.92	0.587	0.922
B	0.907	0.549	0.904	0.905	0.544	0.908
GB	0.93	0.589	0.93	0.924	0.571	0.93
RF	0.928	0.596	0.934	0.94	0.585	0.934
SGD	0.821	0.573	0.791	...	...	...

Table 12: gina_agnostic (openml best *0.963*): No significant difference, everything within 3%. Best up there at a believable 5% within benchmark. No significant difference with LReLU. This dataset has a whopping 970 input features of numeric variables. The values are in $[0, 255]$ and the data is sparse. This really looks like image data (i.e. with sparse positional dependencies) and hence the underwhelming performance of the model.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.622	0.662	0.608	0.608	0.608	0.622
SV	0.689	0.662	0.689	0.662	0.662	0.662
B	0.649	0.649	0.649	0.676	0.649	0.662
GB	0.689	0.662	0.662	0.662	0.649	0.662
RF	0.676	0.676	0.676	0.676	0.689	0.689
SGD	0.581	0.676	0.635	...	...	...

Table 13: heart-h (openml best *0.850*): No significant change, everything within 3%. All results well below benchmark though, so this dataset probably needs preprocessing. No significant difference with LReLU. This dataset has 13 input features, of which 7 are categorical. Many of the categorical features are sparse.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.993	0.993	0.541	0.997	1.	0.528
SV	0.502	1.	0.502	0.505	1.	0.505
B	0.993	0.993	0.525	1.	1.	0.578
GB	0.993	0.993	0.538	1.	1.	0.561
RF	1.	1.	0.571	1.	1.	0.584
SGD	0.908	1.	0.888	...	...	...

Table 14: hill-valley (openml best *0.765*): A significant improvement, all the way up to 1.0 for pure embedding and augmented under B, GB and RF. The original data performance is poor, at <0.6. Interestingly, SVC with augmented data is also low. LReLU performs somewhat better, particularly with B. This dataset has 100 numeric input features, with 2 balanced class labels. There seems to be a strong temporal correlation; values across rows vary greatly, but less so across columns. This suggests that the data may be a time series of different measures.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.637	0.644	0.644	0.678	0.644	0.712
SV	0.719	0.705	0.719	0.712	0.712	0.712
B	0.705	0.692	0.692	0.699	0.705	0.692
GB	0.685	0.678	0.692	0.692	0.692	0.699
RF	0.719	0.685	0.699	0.705	0.719	0.712
SGD	0.719	0.705	0.705	...	...	...

Table 15: ilpd (openml best *0.743*): All about the same, within 3%. No variants reach benchmark. No significant difference with LReLU. This dataset has 10 input features (1 categorical) and 2 poorly balanced classes. There are only 583 rows of data.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.949	0.951	0.843	0.943	0.949	0.835
SV	0.852	0.956	0.852	0.852	0.951	0.852
B	0.953	0.96	0.858	0.958	0.958	0.858
GB	0.949	0.956	0.867	0.964	0.96	0.848
RF	0.958	0.966	0.867	0.966	0.964	0.86
SGD	0.847	0.93	0.845	...	...	...
DT	0.863	0.832	0.794	0.847	0.87	0.786
SV	0.855	0.87	0.855	0.832	0.855	0.832
B	0.878	0.87	0.84	0.878	0.863	0.824
GB	0.87	0.885	0.847	0.878	0.878	0.824
RF	0.878	0.901	0.847	0.893	0.885	0.824
SGD	0.275	0.847	0.466	...	...	...

Table 16: kc1 & kc2 (openml best *0.873 & 0.870*): In both cases the performance improves dramatically, well over the benchmark. These datasets are an interesting case because they have the same structure. That is, they might be performing well because the model was trained on one of them and tested on the other. Nonetheless, they are completely different datasets with different labels, so even if we did overfit to the *kind* of data, the model could not have overfitted to the labels of either. It's a kind of transfer learning that's somewhere between generalising to arbitrary data and generalising to a "twinned" dataset. No significant difference with LReLU. Both of these datasets have 21 numeric features but some half of them with < 100 unique values. Some of the variables are sparse, the 2 target classes are uneven.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.850	0.836	0.652	0.920	0.898	0.662
SV	0.462	0.868	0.460	0.440	0.918	0.448
B	0.886	0.866	0.684	0.944	0.928	0.698
GB	0.896	0.870	0.688	0.944	0.934	0.704
RF	0.892	0.888	0.692	0.958	0.944	0.714
SGD	0.122	0.518	0.212	...	...	...

Table 17: mfeat-morphological (openml best *0.772*): A significant improvement over original data, and over the benchmark. LReLU performs worse, particularly with SV. This dataset has only 6 input features, 3 of them with < 10 unique values. The 10 target classes are perfectly balanced across 2000 rows. The dataset is also ordered by label, and I hope that that isn't causing some systematic error in the experiments.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.758	0.576	0.670	0.794	0.704	0.66
SV	0.826	0.882	0.828	0.824	0.880	0.826
B	0.850	0.752	0.744	0.874	0.810	0.730
GB	0.912	0.806	0.772	0.932	0.858	0.770
RF	0.916	0.842	0.782	0.932	0.874	0.776
SGD	0.786	0.600	0.798	...	...	...

Table 18: mfeat-zernike (openml best *0.907*): A significant improvement over original data, slight improvement over benchmark. LReLU performs consistently worse, particularly with DT. This is another mfeat dataset with the same labels, but completely different features. In this case all 47 numeric variables are continuous. There is a very curious relationship between these results: mfeat-morpological on its own with regular classifiers, but greatly aided by the embedding. This latter dataset, mfeat-zernike, has many more features, and it is easier for off-the-shelf classifiers to tackle. However, the embedding still increases the performance, just not quite to the same degree. Could it be that we are hitting a capacity limit here due to insufficient sampling of the many features?

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.662	0.609	0.96	0.629	0.616	0.987
SV	0.642	0.669	0.689	0.636	0.662	0.662
B	0.656	0.636	0.954	0.702	0.702	0.954
GB	0.629	0.623	0.808	0.689	0.656	0.795
RF	0.675	0.623	0.907	0.709	0.702	0.907
SGD	0.642	0.642	0.623	...	...	...

Table 19: monks-problems-2 (openml best *1.0*): A significant decrease of performance with both augmented and only embeddings. No significant difference with LReLU, also performs poorly. This dataset has 6 numeric attributes but they all have < 5 unique values. It also only has 601 points with 2 classes, and with a benchmark best of 1.0, it is not very interesting as a whole.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.924	0.811	0.930	0.926	0.800	0.928
SV	0.868	0.733	0.870	0.866	0.686	0.868
B	0.946	0.854	0.951	0.943	0.840	0.949
GB	0.945	0.728	0.945	0.944	0.702	0.944
RF	0.951	0.867	0.954	0.949	0.864	0.951
SGD	0.860	0.674	0.886	...	...	...

Table 20: mozilla4 (openml best *0.960*): No significant difference, within 3%. No significant difference with LReLU. 5 features with 1 categorical, 2 classes.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.458	0.28	0.463	0.518	0.39	0.45
SV	0.77	0.535	0.743	0.835	0.603	0.75
B	0.665	0.435	0.63	0.738	0.583	0.643
GB	0.523	0.348	0.503	0.593	0.478	0.49
RF	0.843	0.613	0.825	0.88	0.748	0.798
SGD	0.42	0.17	0.25	...	...	...

Table 21: one-hundred-plants-margin (openml best *0.886*): A significant improvement over original data, almost reaching the benchmark. In all of these plant datasets, LReLU performs worse. Particularly badly affected are DT, GB and RF. All of these three datasets have 64 numeric features and 100 class labels. The labels appear to be identical (save for a missing row in hundred-plants-texture). This means that the results can be compare w.r.t. input data representation (like in kc datasets). This one's features have the lowest level of unique values of the three.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.463	0.368	0.413	0.563	0.445	0.415
SV	0.488	0.493	0.310	0.588	0.593	0.355
B	0.623	0.488	0.518	0.673	0.578	0.515
GB	0.473	0.393	0.418	0.498	0.463	0.415
RF	0.710	0.588	0.59	0.783	0.67	0.583
SGD	0.103	0.085	0.01	...	...	...

Table 22: one-hundred-plants-shape (openml best *0.736*): A significant improvement over original data using RF, above the benchmark. This one's features have the highest level of unique values.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.493	0.285	0.483	0.503	0.323	0.510
SV	0.713	0.198	0.725	0.778	0.510	0.728
B	0.683	0.450	0.690	0.693	0.503	0.690
GB	0.545	0.35	0.540	0.560	0.390	0.563
RF	0.850	0.615	0.825	0.860	0.675	0.820
SGD	0.473	0.055	0.355	...	...	...

Table 23: one-hundred-plants-texture (openml best *0.873*): An improvement over original data with RF, almost reaching the benchmark. This one's features have a medium level of unique values.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.904	0.888	0.913	0.909	0.896	0.894
SV	0.938	0.938	0.938	0.94	0.94	0.94
B	0.937	0.929	0.937	0.942	0.937	0.937
GB	0.934	0.931	0.942	0.938	0.938	0.935
RF	0.937	0.937	0.934	0.946	0.943	0.94
SGD	0.934	0.938	0.934	...	...	...

Table 24: ozone-level-8hr (openml best *0.950*): No significant change, within 3% of original. No significant difference with LReLU. 72 numeric features and a very heavily skewed binary classification.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.957	0.942	0.906	0.942	0.942	0.885
SV	0.946	0.946	0.946	0.935	0.942	0.935
B	0.96	0.957	0.928	0.953	0.953	0.932
GB	0.957	0.96	0.935	0.95	0.946	0.928
RF	0.964	0.96	0.939	0.946	0.953	0.935
SGD	0.932	0.946	0.932	...	...	...

Table 25: pc1 (openml best *0.945*): A small, but consistent improvement over the original, to the tune of 3-6%. More importantly, using B and RF (with embedding only) the model beats the benchmark – whereas the original data is consistently below it. No significant difference with LReLU. Although the names are similar, this dataset is not similar to pc3 and pc4, nor do the labels match. This one has 21 numeric features, although some of them have < 50 unique values. The 2 classes are very uneven.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.854	0.808	0.841	0.847	0.829	0.862
SV	0.903	0.903	0.903	0.898	0.898	0.898
B	0.893	0.898	0.89	0.885	0.89	0.893
GB	0.893	0.898	0.89	0.887	0.895	0.89
RF	0.903	0.908	0.903	0.9	0.903	0.898
SGD	0.885	0.903	0.885	...	...	...

Table 26: pc3 (openml best *0.909*): All within 3%. Interestingly, the best performance is achieved with pure embedding using RF, SV and SGD. The latter is interesting, since in most other cases it's the runt. No significant difference with LReLU. This dataset and the next have the same 37 numeric features and heavily skewed binary labels.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.877	0.825	0.882	0.868	0.792	0.885
SV	0.879	0.879	0.879	0.879	0.879	0.879
B	0.893	0.885	0.904	0.901	0.871	0.901
GB	0.912	0.882	0.901	0.91	0.877	0.904
RF	0.904	0.885	0.904	0.901	0.874	0.904
SGD	0.597	0.879	0.879	...	...	...

Table 27: pc4 (openml best *0.929*): All within 3%, but nothing quite reaches the benchmark. No significant difference with LReLU. Similar to pc3.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.842	0.623	0.874	0.83	0.622	0.861
SV	0.839	0.712	0.839	0.839	0.706	0.839
B	0.862	0.705	0.894	0.871	0.702	0.893
GB	0.852	0.714	0.856	0.852	0.703	0.856
RF	0.878	0.728	0.901	0.878	0.708	0.907
SGD	0.774	0.708	0.777	...	...	...

Table 28: phoneme (openml best *0.923*): All within 3% but original data is consistently best here. No significant difference with LReLU. The 5 continuous numeric features of this dataset are standardised to $\mathcal{N}(0., 1.)$. This is probably why the original data performs so well.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.818	0.784	0.807	0.883	0.871	0.822
SV	0.807	0.848	0.811	0.811	0.917	0.814
B	0.883	0.856	0.86	0.894	0.890	0.867
GB	0.883	0.867	0.871	0.932	0.913	0.867
RF	0.890	0.875	0.883	0.920	0.909	0.875
SGD	0.761	0.814	0.761	...	...	...

Table 29: qsar-biodeg (openml best *0.894*): A considerable improvement, well above the benchmark. LReLU performs worse here with DT, SV and GB affected more. This data has 41 numeric variables of which 24 have < 25 unique values. Some of the low variance features are sparse. The 2 target classes are quite well balanced.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.828	0.545	0.847	0.835	0.566	0.854
SV	0.896	0.699	0.896	0.899	0.657	0.899
B	0.882	0.665	0.897	0.887	0.664	0.897
GB	0.9	0.69	0.908	0.899	0.688	0.902
RF	0.908	0.728	0.915	0.904	0.703	0.914
SGD	0.825	0.493	0.82	...	...	...

Table 30: satimage (openml best *0.929*): Everything within 3%, but augmentation is consistently slightly worse than original. No significant difference with LReLU. This dataset has 36 floating point variables, although they all have < 100 unique features in 6430 rows. Indeed all features are standardised to $\mathcal{N}(0., 1.)$, so odds are the original data was integers.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.936	0.925	0.908	0.937	0.899	0.913
SV	0.702	0.914	0.702	0.702	0.864	0.702
B	0.96	0.957	0.937	0.961	0.936	0.939
GB	0.981	0.966	0.943	0.977	0.941	0.944
RF	0.983	0.98	0.954	0.975	0.964	0.954
SGD	0.787	0.775	0.789	...	...	...

Table 31: spambase (openml best *0.962*): An improvement over original data best, and benchmark. No significant difference with LReLU, though SV is somewhat higher. The data has 57 very sparse numeric features. Quite well balanced classification problem.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.949	0.938	1.	0.965	0.934	1.
SV	0.663	0.992	0.663	0.671	0.979	0.669
B	0.975	0.973	1.	0.971	0.967	1.
GB	1.	0.984	1.	1.	0.975	1.
RF	0.996	0.988	0.994	0.996	0.988	0.994
SGD	0.496	0.938	0.66	...	...	...

Table 32: steel-plates-fault (openml best *1.0*): Reaching the benchmark of 1. easily, with augmented and original data. GB and RF are best. No significant difference with LReLU. Not a very interesting dataset since the benchmark is 1.0.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.937	0.881	0.909	0.937	0.881	0.93
SV	0.916	0.944	0.916	0.902	0.937	0.902
B	0.937	0.888	0.965	0.958	0.888	0.951
GB	0.958	0.93	0.965	0.958	0.916	0.965
RF	0.958	0.937	0.958	0.958	0.93	0.965
SGD	0.853	0.867	0.881	...	...	...

Table 33: wdbc (openml best *0.984*): Everything within 3%. No significant difference with LReLU. 30 numeric features, binary classification. Not much to say, everything performs quite well.

clf	X:EMB	EMB	X	X:EMB	EMB	X
act.	*Leaky ReLU*			*ELU*		
DT	0.973	0.959	0.979	0.969	0.954	0.978
SV	0.949	0.958	0.949	0.945	0.955	0.945
B	0.979	0.969	0.984	0.976	0.963	0.98
GB	0.986	0.96	0.984	0.984	0.959	0.983
RF	0.981	0.97	0.984	0.979	0.965	0.982
SGD	0.944	0.949	0.945	...	...	...

Table 34: wilt (openml best *0.990*): Everything within 3%. No significant difference with LReLU. 5 numeric variables, very imbalanced binary classification and high benchmarks. Not a lot else to say.

REFERENCES

[1] Sercan O Arık and Tomas Pfister. 2020. Tabnet: Attentive interpretable tabular learning. *arXiv* (2020).

[2] Valerio Biscione and Jeffrey Bowers. 2020. Convolutional Neural Networks are not invariant to translation, but they can learn to be. (2020).

[3] Ljubomir Buturović and Dejan Miljković. 2020. A novel method for classification of tabular data using convolutional neural networks. *bioRxiv* (2020).

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[5] Baohua Sun, Lin Yang, Wenhan Zhang, Michael Lin, Patrick Dong, Charles Young, and Jason Dong. 2019. Supertml: Two-dimensional word embedding for the precognition on structured tabular data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 0–0.

[6] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.