# You Say Factorization Machine, I Say Neural Network - It's All in the Activation

Chen Almagor
The Hebrew University of Jerusalem
Jerusalem, Israel
chen.almagor@mail.huji.ac.il

Yedid Hoshen
The Hebrew University of Jerusalem
Jerusalem, Israel
yedid.hoshen@mail.huji.ac.il

## ABSTRACT

In recent years, many methods for machine learning on tabular data were introduced that use either factorization machines, neural networks or both. This created a great variety of methods making it non-obvious which method should be used in practice. We begin by extending the previously established theoretical connection between polynomial neural networks and factorization machines (FM) to recently introduced FM techniques. This allows us to propose a single neural-network-based framework that can switch between the deep learning and FM paradigms by a simple change of an activation function. We further show that an activation function exists which can adaptively learn to select the optimal paradigm. Another key element in our framework is its ability to learn high-dimensional embeddings by low-rank factorization. Our framework can handle numeric and categorical data as well as multiclass outputs. Extensive empirical experiments verify our analytical claims. Source code is available at https://github.com/ChenAlmagor/FiFa

## CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Factorization methods**.

## KEYWORDS

tabular data, neural networks, factorization machines, activation, CTR prediction, machine learning

**ACM Reference Format:**
Chen Almagor and Yedid Hoshen. 2022. You Say Factorization Machine, I Say Neural Network - It's All in the Activation. In *Sixteenth ACM Conference on Recommender Systems (RecSys '22), September 18–23, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3523227.3551499

## 1 INTRODUCTION

Factorization machine (FM) models are a popular family of methods, which play a key role in recommendation systems and online advertising. FM models are a dominant approach for solving benchmark tabular classification tasks, such as click-through-rate (CTR) prediction and user-movie recommendations. Such models learn the effect of feature interactions by low-rank factorization. Due to the powerful ability of deep neural networks to learn feature representations on other modalities, DeepFM (Guo et al. [8]) suggested to integrate the architectures of FM and deep neural networks to leverage a Wide & Deep (Cheng et al. [5]) approach. Since then, various embedding-based neural networks have been proposed to improve the performance. This resulted in a great variety of methods making it non-obvious which method should be used in practice.

In this paper, we first review the previously established connection between polynomial neural networks and basic factorization machine techniques. We extend this connection to more recently introduced factorization machine techniques. Based on this theoretical connection, we propose a neural network-based framework: *fieldwise factorized neural networks (FiFa)*. By changing a single activation layer in FiFa, our framework transforms between being a factorization machine, a shallow ReLU network or a wide-and-deep model. Our framework consists of a set of fieldwise wide yet shallow neural networks, which are aggregated by summation and passed through non-linearity and classification layers. Our framework can handle numeric and categorical data as well as multiclass outputs and can therefore be utilized for a variety of tabular classification tasks.

Our framework is characterized by two main components: i) learning a high-dimensional representation per field by factorized networks, ii) an adaptive activation layer acting on the representation aggregated from all fieldwise networks. Although the theoretical analysis shows that high-dimensional fieldwise embeddings are desired, mapping the high-dimensional features to them requires huge linear layers. Factorizing these layers using low-dimensional layers makes training such fieldwise networks feasible. We also show that for these fieldwise factorized networks, width is preferable to depth. Another key insight is that frameworks with different post-aggregation activation functions can be suitable for different types of tasks. While quadratic activations are optimal for user-product recommendations, ReLU networks are sometimes better for fully-numeric tasks. We thus seek a post-aggregation activation that can enjoy the best of both worlds, and demonstrate that GELU [10] satisfies these requirements. Our theoretical analysis shows that the GELU activation behaves as either quadratic and ReLU activations at different input scales and is thus suitable across the full range of tabular tasks. Consequently, GELU can be used as an efficient alternative for exhaustive search over the optimal activation or an ensemble approach.

We present an extensive experimental analysis to verify our analytical claims. We find that our framework achieves slightly better performance than strong baselines on CTR prediction. In addition, we find that our approach performs comparably to much deeper neural approaches on general tabular classification dataset.


*RecSys '22, September 18–23, 2022, Seattle, WA, USA*

ACM ISBN 978-1-4503-9278-5/22/09…$15.00
https://doi.org/10.1145/3523227.3551499

In an ablation study, we show that the network activation and width play a larger role than depth or attention for tabular data.

Our contributions include: (1) Extending the previously established connection between basic factorization machines and polynomial neural networks, to a set of popular, recently introduced factorization machine techniques. In addition, we provide this connection with a fieldwise interpretation. (2) Presenting a general, theoretically-grounded neural architecture framework for tabular data classification. Our method can automatically switch between many popular methods by simply modifying its activation function. It also makes learning high-dimensional fieldwise representations feasible by low-rank factorization. (3) Highlighting and justifying the relative importance of different architectural features: activation and width (more influential) vs. depth or attention (less influential).

## 1.1 Related Work

**FM-based models.** Factorization machine models are a dominant approach for solving benchmark tabular classification tasks. This popular family of methods includes: matrix factorization (Koren [12]), factorization machine (FM, Rendle [18]), Field-aware Factorization Machines (FFM, Zhang et al. [24]), Field-weighted Factorization Machine (FwFM, Pan et al. [17]) and Field-matrixed Factorization Machines (FmFM, Sun et al. [21]). We will give a more in-depth introduction to FM-based methods in Sec. 2. As a step toward combining deep neural network for this task, Wide & Deep (Cheng et al. [5]) proposed to train a joint network that combines a linear model and a deep neural network. DeepFM (Guo et al. [8]) suggest to learn a low-order feature interactions through the FM component instead of the linear model. Since then, various embedding-based neural networks have been proposed to improve the performance (Deng et al. [6]; He and Chua [9]; Wang et al. [22]; Lian et al. [14]). Unlike He and Chua [9], our framework is able to learn higher dimensional representations, as well as leveraging non-linear embeddings for the numerical fields. It is also more general, as it able to recover range of tabular models by changing the activation.

**The connection between factorization machines and neural networks.** Blondel et al. [2] established the connection between Factorization Machines (Rendle [18]) and polynomial networks (Livni et al. [15]). As a special case, by employing a quadratic activation, factorization machines can be formulated as polynomial networks. In addition, they proposed a lifted approach, based on casting parameter estimation as a low-rank tensor estimation problem. We extended this connection to a set of recently introduced state-of-the-art factorization machine techniques. We also propose a fieldwise interpretation, and propose a general framework based on this theoretical connection. Our framework transforms between being a factorization machine, a shallow ReLU network or a wide-and-deep model, by changing the activation function. Furthermore, we suggest GELU as an activation that automatically adapt to the optimal approach among those models. Also, differently from the previous work, our method is able to effectively handle numerical field and can benefit from high-dimensional embeddings by incorporating low-rank fieldwise factorization.

**Factorization Machines vs. Neural approaches.** Rendle et al. [19] study neural approaches versus dot product similarities. They essentially show that a simple matrix factorization model performs better than MLP and NeuMF models (Zhao et al. [25]) for collaborative filtering, and that approximating dot product by MLP with ReLU activation is hard. In this study, we will extend their experiments by showing that controlling the activation function of our framework recovers the different behaviors, and suggest using GELU activation ([10]) as a bridge between the paradigms.

**Deep models for general tabular data.** While classical methods are still the industry favorite, some recent work propose to use deep learning for tabular data. For example, TabNet (Arik and Pfister [1]) uses neural networks to mimic decision trees by placing importance on only a few features at each layer, using modified attention layers. Yoon et al. [23] propose VIME, which employs MLPs in a technique for pre-training based on denoising. Transformer models for more general tabular data include TabTransformer (Huang et al. [11]), which uses a transformer encoder to learn contextual embeddings only on categorical features. The main issue with this model is that numerical data do not go through the self-attention block, but are only fed to an MLP. Gorishniy et al. [7] suggests FT-Transformer, which embed numerical fields using a linear layer. We will show that numerical fields should be embedded in a non-linear manner in an high-dimensional space. SAINT (Somepalli et al. [20]) address that issue by non-linearly projecting numerical features to the higher dimensional embedding space and passing them, together with the categorical embeddings, through the transformer blocks. In addition, SAINT propose using attention in the rows level, to explicitly allow data points to attend each other.

**Fieldwise models.** Li et al. [13] present a model for categorical data, that utilizes linear models with variance and low-rank constraints, and is also interpretable in a field-wise manner. Although we share the ideas of fieldswise low-rank factorization, our framework allows non-linearity, as well as handles both numerical and categorical fields. Luo et al. [16] propose NON to take advantage of intra-field information and non-linear interactions. However, our components are much simpler yet effective. Our fieldwise networks rely on low-rank factorization of wider networks and our non-linearity is based on an adaptive activation, while NON utilize only ReLU activation and ensembles multiple heavy aggregation mechanisms, such as attention and deep neural networks.

# 2 BACKGROUND - THE THEORETICAL CONNECTION BETWEEN FACTORIZATION MACHINES AND FIELDWISE NEURAL NETWORKS

In this section, we will briefly overview factorization machine approaches. We will then demonstrate a theoretical connection between them and shallow neural networks, which is based on the established connection between factorization machines and polynomial neural networks (Blondel et al. [2]). This will form the basis for our final, generalized framework in Sec. 3.

## 2.1 A Unified Form for Factorization Machine Models

In this section we describe a common approach for classification of tabular data. For ease of explanation, we detail the case of *binary* classification of multi-field *categorical* data. However, there is no

loss of generality, our final approach applies for categorical and numerical as well as for multi-class tabular datasets.

**Preliminaries.** A training dataset consists of $S$ labelled samples $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})..(x^{(S)}, y^{(S)})\}$ , where $x$ and $y$ are a sample and label respectively. Each sample $x$ is specified by $C$ categorical features $f_1, ..f_C$ and $J$ different fields $F_1, ...F_J$. Each field may contain multiple features, while each feature belongs to only one field. An example of fields are "Country" or "City", whereas features could be "Japan" or "Rome". To simplify the notation, we use index $i$ to represent the feature $f_i$ and $F(i)$ to represent the field which $f_i$ belongs to. $S_F$ denotes the set of features belonging to field $f$ and $J_{F(i)}$ represents their number. We denote $x_i^{(s)} = 1$ if feature $i$ is active for this instance, otherwise $x_i^{(s)} = 0$. We denote the one-hot vector of features per-field $\boldsymbol{x_F} = concat\{x_i \in S_F\}$. We denote the embedding of feature $i$ as $\boldsymbol{e_i} \in \mathbb{R}^K$, where $K$ is the (usually small) feature embedding dimension. We denote by $E_F \in \mathbb{R}^{K \times J_{F(i)}}$ the field embedding matrix whose rows are the embeddings $\boldsymbol{e_i}$ of the features $S_F$ belonging to the field $f$.

**Logistic Regression (LR).** This method is probably the most widely used model for this task. However, linear models lack the capability to represent feature interactions as cross features are often significant (Chapelle et al. [4]). *Degree-2 Polynomials (Poly2)* (Chang et al. [3]) model a general way to address this problem. Such models learn a dedicated weight, $W_{ij}$, for each feature conjunction resulting in a field interaction matrix $W \in \mathbb{R}^{C \times C}$.

**Factorization Machine Models:** Estimating $W$ is hard due to its huge dimensionality and missing values. Factorization-machine models propose to learn the effect of feature conjunctions by low-rank factorization of the interaction matrix $W$. Factorization machine methods approximate the feature interaction strength as the scalar product of their embeddings weighted by a matrix $M_{F(i),F(j)} \in \mathbb{R}^{K \times K}$, that depends on the fields of the two features: $W_{i,j} \approx \boldsymbol{e_i}^T M_{F(i),F(j)} \boldsymbol{e_j}$. Different factorization machine methods are distinguished by their particular choice of matrix $M_{F(i),F(j)}$. We present the choices of $M_{F(i),F(j)}$ taken by four representative factorization machine methods in Eq. 1: FM [12], FFM [24], FwFM [17], FmFM [21]:

$$M_{F(i),F(j)} = \begin{cases} I_K & \Phi_{FM} \\ P_{F(j)}^T P_{F(i)} & \Phi_{FFM} \\ I_K \cdot r_{F(i),F(j)} & \Phi_{FwFM} \\ Entire\ Matrix & \Phi_{FmFM} \end{cases} \tag{1}$$

Where $I_K$ is the identity matrix, $P_{F(i)}$ is a per-field sparse binary projection matrix, $r_{F(i),F(j)}$ is a scalar value learned for every pair of fields $(F(i), F(j))$. For FM and FFM, $M$ is not learned, while for FwFM a scalar per pair of fields is learned, and for FmFM the entire matrix is learned. See Appendix A for detailed formulation of $M$ for each FM-based model.

## 2.2 Factorization-Based models as a Fieldwise factorized neural network

Blondel et al. [2] demonstrated that factorization machines are a special case of polynomial neural networks. In this section, we review this connection in light of the fieldwise interpretation.

Let us denote the second order interaction term as $q_2 = \sum_i \sum_j x_i x_j \cdot \boldsymbol{e_i}^T M_{F(i),F(j)} \boldsymbol{e_j}$. We demonstrated above that the crux of factorization machine methods is in the representation of the interaction matrix $M_{F(i),F(j)}$. Let $M$ be the matrix that consists of the matrices $M_{F(i),F(j)}$ for all pairs of fields. Since there is no order between the fields, for each pair only one matrix is learned, meaning $M_{FiFj} == M_{FjFi}$, while the diagonal block matrices are zeros. See Appendix A for a detailed formulation of M. We observe that $M$ is symmetric matrix, and therefore an eigen-decomposition can be applied to it: $M = U^T \Lambda U$. Here we propose to utilize a reduced-rank factorization of $M$ resulting in a field-wise factorization: $M_{F(i),F(j)} = U_{F(i)}^T \Lambda U_{F(j)}$, where $U_{F(i)} \in \mathbb{R}^{d \times K}$. Note that the full rank dimension of $M$ is equal to the number of fields multiplied by the number of dimensions per-embedding. Using this factorization the feature interaction term $q_2$ can now be written:

$$q_2 = \sum_i \sum_j (x_i U_{F(i)} \boldsymbol{e_i})^T \Lambda (x_j U_{F(j)} \boldsymbol{e_j}) \tag{2}$$

We rewrite the sums, as the sum over the field $f$ and the sum over the per-field indices $i \in S_F$:

$$q_2 = \sum_F \sum_{F'} (\sum_{i \in S_F} x_i \boldsymbol{e_i})^T U_F^T \Lambda U_{F'} (\sum_{j \in S_{F'}} x_j \boldsymbol{e_j}) \tag{3}$$

Note that as $\boldsymbol{x_F}$ is one-hot (only a single element has a non-zero value), the product $E_F \boldsymbol{x_F} = \sum_{j \in S_{F'}} x_j \boldsymbol{e_j}$ can be efficiently computed using an embedding layer $E_F \boldsymbol{x_F}$:

$$q_2 = (\sum_F U_F E_F \boldsymbol{x_F})^T \Lambda (\sum_{F'} U_{F'} E_{F'} \boldsymbol{x_{F'}}) \tag{4}$$

As the right-hand left-hand vectors are equal, it yields the simple expression:

$$q_2 = diag(\Lambda) \cdot (\sum_F U_F E_F\ \boldsymbol{x_F})^2 \tag{5}$$

Where the above square is elementwise. As an intuitive explanation, the second order interactions are modelled by several steps: i) embedding of the one-hot per-field feature using linear layer $E_F$. ii) projection of the feature embedding to a higher dimension using the per-field projection matrix $U_F$. iii) summation over the projected embeddings of all fields. iv) computing the scalar product of their square with the diagonal of the matrix $\Lambda$. Note that this can be expressed by a shallow neural network that first learns a per-field representation, then aggregates the representations by summation, passes the result through a non-linearity, and then a linear layer. Therefore, as proven by Blondel et al. [2], Factorization Machines are a particular instantiation of this neural network, when the activation function is quadratic. This will be generalized in the next section.

# 3 FIELDWISE FACTORIZED NETWORKS FOR TABULAR CLASSIFICATION

We propose *fieldwise-factorized neural-networks*, a general framework for tabular classification.

## 3.1 General Framework

In this section, we overview our proposed framework: *fieldwise-factorized neural-networks* (FiFa), while detailing its components in the following sections. An illustration of our framework can

be found in Figure 1. We learn a dedicated neural network $\phi_F$ for each field that takes as input the values of the field $\boldsymbol{x_F}$ (one-hot for categorical fields, scalar for numeric fields) and returns a high-dimensional embedding $\phi_F(\boldsymbol{x_F})$. The architecture of the fieldwise neural network will be described in Sec. 3.2. The aggregated high-dimensional feature is obtained by summing over the fields $\sum_F \phi_f(\boldsymbol{x_F})$. It is then passed through an activation function $\sigma$. Our activation $\sigma$ allows different activation functions for different dimensions. The post-activation results are finally multiplied by output linear layer $W_{out}$, mapping it to the output logits $z_{out}$:

$$z_{out} = W_{out}\sigma(\sum_F \phi_F(\boldsymbol{x_F})) \tag{6}$$

Note that differently from factorization machine, but similarly to other neural networks, our framework is able to handle both binary and multiclass classification tasks.

The framework is characterized by two main components: per-field representation by factorized networks $\phi_F$, and an adaptive activation $\sigma$ on the aggregated representations of all fields. We will describe them in detail in the next sections.

## 3.2 Fieldwise Factorized Networks

The theoretical basis in Sec. 2 suggests that the learned field embeddings should be of high-dimension to factorize a full-rank $M_{F(i),F(j)}$, but that these representations should be low-rank factorized. The motivation behind these design choices is to reduce sample complexity. Modeling field interactions requires a high-dimension while there may not be a sufficient number of samples per-feature for estimating it. Complexity is reduced: i) by learning a per-field representation that does not take other features into account ii) by using low-rank factorization inside these fieldwise networks. Since we are able to use other activation than just than quadratic, $d$ is allowed to be even larger than the full rank dimension. Our framework is able to handle both categorical and numeric fields:

**Categorical fields.** Following Sec. 2.2, we learn fieldwise networks $\phi_F^{cat}(\boldsymbol{x_F}) = U_F E_F \boldsymbol{x_F}$. We choose the per-feature embedding (output of $E_F$) to be low-dimensional as there are often many features and limited data per-feature, while matrix $U_F$ projects this to high-dimension.

**Numeric fields.** Differently from categorical fields, numeric fields are ordered which enables learning more complex functions. We choose to learn a factorized one-hidden layer for each field. Our network first projects the scalar value to a high-dimension $\boldsymbol{t_F} = \boldsymbol{v_F} x_F + \boldsymbol{b_F}$ (where $\boldsymbol{v_t}, \boldsymbol{b_F} \in R^l$). The results are passed through a ReLU network, and mapped to the per-field embedding. We found that a high dimensional $\boldsymbol{t_F}$ is important for achieving strong performance. However, as $\boldsymbol{t_F}$ and the output field-embedding $\phi_F$ have a high-dimension, the second linear layer becomes very large. Instead, we choose to low-rank factorize the second layer. The post-activation $ReLU(\boldsymbol{t_F})$ is projected to a low-dimension, by linear layer $E_F$ and is then projected using linear layer $U_F$ to the high-dimensional field embedding $\phi_F$. Note that as both $E_F$ and $U_F$ are linear, with no intermediate non-linearity - they are equivalent to a single (low-rank) linear layer. The entire network is therefore a factorized one-hidden layer neural network.

In summary, the fieldwise factorized networks for categorical and numeric variables are given by:

$$\phi(\boldsymbol{x_F}) = \begin{cases} U_F E_F \boldsymbol{x_F} & Categorical \\ U_F E_F ReLU(\boldsymbol{v_F} x_F + \boldsymbol{b_F}) & Numeric \end{cases} \tag{7}$$

Note that although SAINT (Somepalli et al. [20] also use a one-hidden layer network for numeric values, its formulation does not use the low-rank factorization and therefore cannot handle high width, which we show is key to the performance of our method. Also note that our formulation can easily handle deeper fieldwise architectures, but we did not observe benefits from deeper networks.

## 3.3 Our framework can express popular tabular classification methods

Despite the simplicity of our framework, it is very general. By changing the fieldwise factorized network $\phi_F$ and activation $\sigma$, it can express several popular tabular classification methods:

*Factorization machine methods:* When choosing a quadratic activation $\sigma(x) = x^2$ and the fieldwise network $\phi_F(\boldsymbol{x_F}) = U_F E_F \boldsymbol{x_F}$ our framework becomes a factorization machine-based model. This proof was detailed in Sec. 2.2. Therefore, factorization machine models are expressible by our framework.

*One-hidden Layer ReLU Networks:* When choosing the activation $\sigma(x) = ReLU(x)$ and the fieldwise networks as simple linear layers $\phi_F(x) = U_F x$, our framework becomes a one-hidden layer neural network. Although more layers can be easily added to our framework after the activation layer $\sigma$, we did not find this beneficial in practice. We will present empirical results in Sec. 4.3 for showing this.

*Wide-and-Deep:* When selecting a fraction of dimensions of $\sigma$ to have quadratic activations, while the rest are selected to have ReLU activation, our framework becomes a wide-and-deep model (as it is the sum of a ReLU network and a factorization machine). Although the original wide-and-deep method used full rank for the feature interaction $W_{ij}$, later methods (e.g. Guo et al. [8]) use different factorization machine varieties. By modifying the choice of dimensionality of the per-feature embedding $\boldsymbol{e_i}$ and per-field projection $d$, all the above methods are expressible by our model. One caveat is that the deep part of our framework only has a single hidden layer, but as mentioned previously, adding further layers has not improved results in our experiments.

## 3.4 Adaptive Non-Linearity

In Sec. 3.3 we established that by choosing different activation function $\sigma$, our framework can express a range of popular tabular classification methods. We will show in Sec. 4.2, that in different classification problems, either $\sigma(x) = ReLU(x)$ or $\sigma(x) = x^2$ yields significantly better performance. By the analysis in Sec. 3.3 these are cases where either ReLU neural networks or factorization machine achieve better results. By selecting $\sigma$ a concatenation of ReLU and quadratic functions, we may be able to deal with all cases. However this may not be an efficient solution as it doubles the number of parameters. It would naturally be attractive to use a non-linearity that may be able to automatically adapt to the setting most beneficial to a particular dataset. Here we suggest using the GELU activation

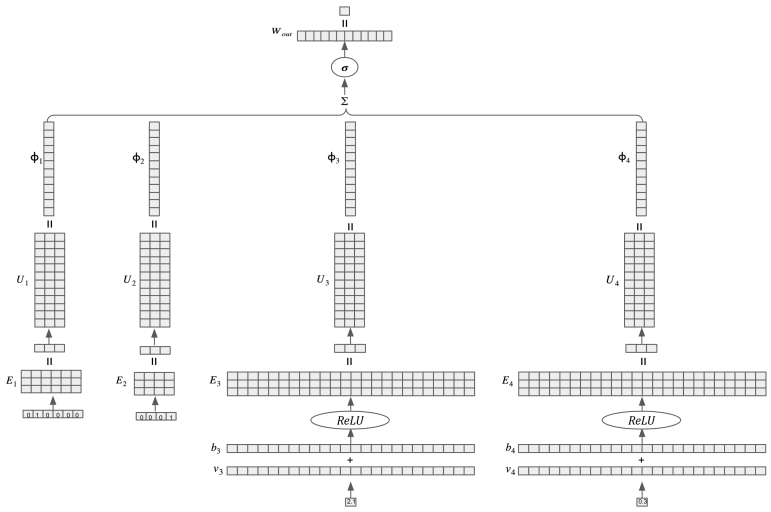


**Figure 1: Fieldwise factorized neural networks (FiFa) architecture. Toy example of binary classification: Fields 3 and 4 are numerical, while fields 1 and 2 are categorical, with six and four categories, respectively.**

function [10] as a bridge between the two paradigms. At the limits, $GELU(x)$ has the following attractive properties:

$$GELU(x) \approx \begin{cases} 0.5x + \sqrt{\frac{1}{2\pi}}x^2 & \|x\| << 1 \\ ReLU(x) & \|x\| \to \infty \end{cases} \tag{8}$$

Therefore, by varying the magnitude of the input, and utilizing the GELU activation, our framework is able to switch between factorization machine and ReLU network behavior. The adaptive behavior will be demonstrated empirically in Sec. 4.2. GELU can be used as an efficient alternative for exhaustive search over the optimal activation function or an ensemble approach (wide-and-deep).

## 3.5 Practical Considerations

The field of deep learning has made significant steps forward by increasing the depth of neural networks and by using attention-based architectures such as transformers. It might be expected that increasing network depth or using transformers might be beneficial for tabular data. Unfortunately, in our experiments, we did not find evidence to support this. We do not claim that it is impossible to gain by using the two mechanisms, but this is not trivial. More optimistically, in our experiments we found that there are lower hanging fruit. Specifically, we obtained significant gains by choosing suitable width $d, l$ and activation $\sigma$, as well as tuning the regularization strength (either dropout or weight decay). We suggest GELU and full rank projections $U_F$ ($d = J \cdot K$) as robust default values. Although tuning the embedding dimension $K$ can also help, we kept it in-line with the baselines.

# 4 EXPERIMENTS

## 4.1 Performance Comparison

*4.1.1 CTR prediction.* CTR prediction is a fundamental tabular binary classification task. FM-based models are the most common approach for tackling it. As our framework generalizes FM-based models, we examined if CTR prediction can benefit from our framework.

**Table 1: Performance comparison: CTR prediction (ROC-AUC %). Average results over four different seeds.**

| | Criteo |
|---|---|
| FmFM | 81.157 ±0.008 |
| DeepFmFM | 81.181 ±0.003 |
| DeepFwFM | 81.087 ±0.017 |
| Ours | 81.250 ±0.002 |

| | Avazu |
|---|---|
| FmFM | 78.635 ±0.011 |
| DeepFmFM | 78.670 ±0.013 |
| DeepFwFM | 78.559 ±0.020 |
| Ours | 78.687 ±0.005 |

**Datasets.** We used Avazu and Criteo which are popular CTR prediction benchmarks.

**Baselines.** We compared the framework implementation against FmFM, which is the shallow state-of-the-art of FM-models, and two deep variations - DeepFwFM and DeepFmFM. We implemented our framework on the FmFM public code-base [1], and used their data processing and baselines implementation. We split the data as described in FmFM paper, and applied the same splits for all the models, We conducted a hyper-parameter search for both our method and FmFM, and found the optimal hyperparameters for both were the same. We thus applied the same hyperparameters for the deep baselines, and observed an improvement. The results can be found in Table 5. Other implementation details are provided in Appendix B.

**Results.** The averaged ROC-AUC results over four different seeds are reported in Table 1.We can observe that even after improving the baselines by better tuning, our framework achieves slightly better performance than all other baselines. Due to the improved results, combined with our framework's ability to generalize FM-base models, we suggest that our framework should be used as a first attempt for CTR prediction tasks.

*4.1.2 Comparing FiFA against deeper methods on general tabular classification datasets.* In this section, we evaluate our framework

[1] https://github.com/yahoo/FmFM

**Table 2: General tabular data classification. Left: Performance comparison - average over all datasets. Above line - results taken from SAINT paper evaluated on unknown seeds. Below line - mean and standard deviation of SAINT results computed using the authors' code averaged over seeds 1-10, and of our method evaluated on exactly the same data. Right: Inference time comparison (milliseconds per batch on average). FiFa is faster than SAINT variations.**

| | All |
|---|---|
| MLP | 84.106 |
| VIME | 78.482 |
| TabNet | 86.515 |
| Tab Transformer | 88.654 |
| SAINT | 91.884 |
| SAINT-i | 91.716 |
| SAINT-s | 90.669 |
| SAINT | 91.596 ±0.108 |
| SAINT-i | 91.286 ±0.142 |
| SAINT-s | 89.821 ±0.230 |
| Ours | 91.635 ±0.158 |

| | Income | Forest | Bank |
|---|---|---|---|
| SAINT-s | 6.890 | 7.621 | 5.764 |
| SAINT-i | 1.502 | 3.327 | 1.348 |
| SAINT | 2.449 | 4.480 | 2.107 |
| Ours | 0.660 | 0.933 | 0.575 |

**Table 3: GELU as adapted activation. HR@10 on Movielens dataset**

| Emb. size | ReLU | GELU | Quad. | MF |
|---|---|---|---|---|
| 16 | 0.6876 | 0.6972 | 0.6974 | 0.6937 |
| 192 | 0.7164 | 0.7217 | 0.7280 | 0.7278 |

against deeper neural approaches. The evaluation is performed on general tabular classification tasks where deep approaches reported their most significant gains. Our focus here is only to evaluate the utility of deeper neural architecture rather than finding the universally optimal learning methods (where tree-based methods are typically very competitive). MLP represents standard neural networks. VIME (Yoon et al. [23]) and TabNet (Arik and Pfister [1]) represent specialized neural architectures. Tab Transformer (Huang et al. [11]) and SAINT (Somepalli et al. [20]) are chosen to represent transformer based approaches. While Tab Transformer deals with numerical features in a naive manner, SAINT shares the same approach as our method regarding numerical and categorical fields embedding. The difference lies in SAINT using multiple attention blocks compared to our simple Pooling+Linear classification approach. In this comparison, we did not include ensemble methods (differently from FT-Transformer [7]).

**Evaluation protocol.** We evaluated our method on the same datasets as SAINT [2] . The datasets include 14 binary classification tasks and 2 multiclass classification tasks. We did not include Arcene and Arrythmia datasets from the reported results as they are very small datasets and their results have high variance (Arrhythmia contains 452 samples and Arcene contains 200 samples). In addition, we use exactly the same pre-processing as in the official SAINT implementation. Details of these datasets and the pre-processing are provided in Appendix B.

**Hyperparameters.** Our optimization hyper-parameters follow SAINT. We run a grid search on the hyperparameters presented in Table 8 with seed=1, and selected the best model by the validation set. For fair comparison, we used the same $E_F$ dimensionality as SAINT defaults.

**Baselines.** We compare our framework against multiple SAINT variations, as well as against the deep learning methods reported in SAINT: multi-layer perceptrons, VIME (Yoon et al. [23]), TabNet (Arik and Pfister [1]), and TabTransformer (Huang et al. [11]). The numbers were copied from the SAINT paper. In order to compare the performance on the same exact seeds, we re-run the SAINT models on 10 different seeds (1-10). For a fair comparison, we trained SAINT with our best numerical embedding hidden dimension $l$ and their default selection (100), and reported the best of the two settings.

**Results.** The averaged performance for all methods is reported in Table 2, the results per dataset are detailed in Table 10. We reported ROC-AUC for binary classification, and class prediction accuracy or multi-class classification. Our results are slightly better or comparable to SAINT variations, except from on multiclass tasks, where SAINT-s achieves poor performance. Our method also outperforms all of SAINT's baselines on average. SAINT-s and our framework use exactly the same learning setting (optimization hyper-parameters, datasets). The difference between the architectures is the post-$E_F$ architecture. While we use a set of linear projections that are only summed and passed through activation and classification layers, SAINT-s uses a massive six layer Transformer. This highlights that for robust and accurate neural models for tabular data, it does not appear that complex methods have an edge over well-tuned simple models such as ours. We also examined the *runtime* between our framework and SAINT. We computed the average inference time over all batches of size 256. The results on three datasets are reported in Table 2. We can observe that using a transformer slows inference without increasing accuracy.

## 4.2 GELU as an automatically adapted activation

We conducted an empirical study on the advantage of using GELU activation, by examining two cases - a case where quadratic activations are preferred over ReLU, and a case where ReLU activations are preferred. We demonstrate that by using GELU, we can get the closest result to the preferred activation.

*Quadratic is better than ReLU.* Rendle et al. [19] showed that a simple matrix factorization model performs better than MLP and NeuMF models (Zhao et al. [25]) for collaborative filtering. We applied our method on the authors' dataset and settings. We tested the effect of different choices of $\sigma$ on the ability of our framework

[2]https://github.com/somepago/saint

**Table 4: Ablation studies. Left: General framework architecture ablations Right: Numeric fieldwise networks ablations**

| Variation | Binary (ROC-AUC) | Multiclass (Accuracy) | All |
|---|---|---|---|
| Full method | 93.16 | 84.02 | 91.85 |
| Shared | 92.55 | 70.47 | 89.40 |
| Deep | 92.89 | 82.57 | 91.41 |
| Low rank | 92.64 | 78.97 | 90.69 |
| Quadratic | 92.70 | 82.31 | 91.21 |
| ReLU | 93.05 | 82.58 | 91.55 |
| GELU | 93.00 | 82.61 | 91.51 |

| Variation | Num. DS |
|---|---|
| Full method | 91.40 |
| (i) $U_F E_F x_F$ | 90.74 |
| (ii) $l = K$ | 91.17 |
| (iii) $U_F E_F (\boldsymbol{v_F} x_F + \boldsymbol{b_F})$ | 90.80 |

to express a dot product between fields. Dedicated choices of initialization and regularization were used, see details in Appendix B. We report the results in Table 3, for embedding sizes $K = 16$ and $K = 192$, the minimal and maximal embedding sizes that were tested in Rendle et al. [19]. Note that the reported result for matrix factorization of Rendle et al. [19] is 0.7294 (on embedding size of 192). As expected, we can observe that for each embedding size, quadratic activation performs the best, while ReLU performs poorly. However, GELU achieves similar results to the quadratic performance, demonstrating its robustness and adaptive behaviour.

*ReLU is better than Quadratic.* On the other hand, from our experiments on the 'forest cover' dataset (which is fully numeric), we observed that a quadratic activation performs dramatically worse than ReLU ( 94.84% and 99.4%, respectively). However, GELU obtains 99.47% which is in line with ReLU.

## 4.3 Ablation studies

In this section, we explore the contributions of the individual component of our framework. The ablations were conducted on the evaluation datasets used in Somepalli et al. [20]. Due to resource constraints, we run the experiments on seed=1 only. The results are reported in Table 4, averaged over the datasets, more details are found in Appendix C.

**Per-field Projections.** To examine the importance of the per-field networks, we evaluated sharing the projection matrix $U_F$ across fields. We can observe that all datasets (besides HTRU2), gain from the fieldwise projections. In multi-class datasets, the difference is more significant.

**Low rank projections.** In this experiment we reduced the dimension of the projection matrix $U_F$ to that of the output of $E_F$. Overall, we can observe, that the factorized high-dimensional representation is crucial, especially in Forest, Volkert and MNIST.

**Numeric fieldwise networks.** We tested the following alternatives: (i) a $K$-dimensional linear layer for each field, (ii) applying non-linearity, without a high-dimensional projection, meaning $\boldsymbol{v_F}$ dimension is equal to $E_F$ dimension, (iii) projecting to high-dimensional space and reduce to $K$-dimensional, without a non-linearity. The averaged results over the datasets that contain continuous fields are reported in Table 4. We can observe that the non-linearity is an important aspect, and boosts the performance. In addition, by controlling the width for the high-dimensional space, we gain a further improvement over the low-dimensional non-linear network.

**Deeper model.** We added 2 extra layers after the activation $\sigma$ with equal width as field embedding, $\phi(\boldsymbol{x_F})$, and matching the model activation and dropout. Since quadratic activation performs poorly when deepening the network, we tested all activations for the datasets that selected a quadratic in their shallow variation, and report the best. While we observed improvements on specific datasets, such as Volkert and Forest, in most of the datasets the results were comparable or even worse than our shallow model with the linear classification head.

**Activation function.** In the original selected models, the selected activations are divided as follow: 6 datasets with a quadratic activation, 4 datasets with ReLU activation, and 4 with GELU activation. In order to explore the effect of the activation, we vary the activation for each dataset, while keeping the other hyperparameters fixed. The averaged results are reported in Table 4. Although in many datasets the specific activation was immaterial, in some datasets, the activation made a significant difference. This suggests that activation functions should be tuned per dataset.

## 5 DISCUSSION AND CONCLUSIONS

We presented a theoretically grounded, general architecture framework for handling tabular data classification. Our framework can be extended in many ways:

*Combining with improved regularization.* Ample evidence exists that improved regularization and hyper-parameter selection methods hold the key for increasing the performance, and can therefore improve our method too.

*Activation for tabular tasks.* We highlighted the importance of the activation function and demonstrated both theoretically and empirically, that GELU activation is suitable across the full range of tabular tasks. This suggests that developing an activation function for tabular classification is very promising.

*Exploring the learned representations.* Our framework learns fieldwise representations as a by-product. Future work should examine if the representations are useful for transfer learning for related tasks.

**Acknowledgements.** Chen Almagor was supported by a Facebook award and by a grant from the Israeli Prime Minister's Office. This research was also supported by Oracle GPU credits.

## REFERENCES

[1] Sercan Ö. Arik and Tomas Pfister. 2021. TabNet: Attentive Interpretable Tabular Learning. In *AAAI*.

[2] Mathieu Blondel, Masakazu Ishihata, Akinori Fujino, and Naonori Ueda. 2016. Polynomial Networks and Factorization Machines: New Insights and Efficient Training Algorithms. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 850–858. http://proceedings.mlr.press/v48/blondel16.html

[3] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. 2010. Training and Testing Low-degree Polynomial Data Mappings via Linear SVM. *J. Mach. Learn. Res.* (2010).

[4] Olivier Chapelle, Eren Manavoglu, and Rómer Rosales. 2014. Simple and Scalable Response Prediction for Display Advertising. *ACM Trans. Intell. Syst. Technol.* (2014).

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016.*

[6] Wei Deng, Junwei Pan, Tian Zhou, Deguang Kong, Aaron Flores, and Guang Lin. 2021. DeepLight: Deep Lightweight Feature Interactions for Accelerating CTR Predictions in Ad Serving. In *WSDM.*

[7] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting Deep Learning Models for Tabular Data. *CoRR* abs/2106.11959 (2021). arXiv:2106.11959 https://arxiv.org/abs/2106.11959

[8] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *IJCAI.*

[9] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *SIGIR.*

[10] Dan Hendrycks and Kevin Gimpel. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR* (2016).

[11] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar S. Karnin. 2020. TabTransformer: Tabular Data Modeling Using Contextual Embeddings. *CoRR* (2020).

[12] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD.*

[13] Zhibin Li, Jian Zhang, Yongshun Gong, Yazhou Yao, and Qiang Wu. 2020. Field-wise Learning for Multi-field Categorical Data. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/7078971350bcefbc6ec2779c9b84a9bd-Abstract.html

[14] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. In *KDD.*

[15] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. 2014. On the Computational Efficiency of Training Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (Eds.). 855–863. https://proceedings.neurips.cc/paper/2014/hash/3a0772443a0739141292a5429b952fe6-Abstract.html

[16] Yuanfei Luo, Hao Zhou, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. 2020. Network On Network for Tabular Data Classification in Real-world Applications. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (Eds.). ACM, 2317–2326. https://doi.org/10.1145/3397271.3401437

[17] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. 2018. Field-weighted Factorization Machines for Click-Through Rate Prediction in Display Advertising. In *WWW.*

[18] Steffen Rendle. 2010. Factorization Machines. In *ICDM.*

[19] Steffen Rendle, Walid Krichene, Li Zhang, and John R. Anderson. 2020. Neural Collaborative Filtering vs. Matrix Factorization Revisited. In *RecSys.*

[20] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. 2021. SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training. *CoRR* (2021).

[21] Yang Sun, Junwei Pan, Alex Zhang, and Aaron Flores. 2021. FM2: Field-matrixed Factorization Machines for Recommender Systems. In *WWW.* 2828–2837.

[22] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *ADKDD.*

[23] Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela van der Schaar. 2020. VIME: Extending the Success of Self- and Semi-supervised Learning to Tabular Domain. In *NeurIPS.*

[24] Zhiyuan Zhang, Yun Liu, and Zhenjiang Zhang. 2018. Field-Aware Matrix Factorization for Recommender Systems. *IEEE Access* (2018).

[25] Xinke Zhao, Wei Zeng, and Yixin He. 2021. Collaborative filtering via factorized neural networks. *Appl. Soft Comput.* (2021).


# A FM-BASED MODELS FORMULATIONS

**Embedding form:** FM, FwFM and FmFM, learn $C$ vectors, one for each feature, $\boldsymbol{e_i} \in \mathbb{R}^K$.

FFM learns $(J-1)$ vectors for each feature, $\boldsymbol{e_i}^{F(j)} \in R^{\frac{K}{J}}$, overall $C \cdot (J-1)$ vectors. Note that they select much higher $K$ than all other approaches, therefore $\frac{K}{J} > 1$. $\boldsymbol{e_i} \in R^K$ places these $(J-1)$ vectors in the relevant indices, while $\mathbf{0}_{\frac{K}{J}}$ in its field indices. For example, $\boldsymbol{e_1} = [\mathbf{0}_{\frac{K}{J}}, \boldsymbol{e_1^2}, ..., \boldsymbol{e_1^J}]$, $\boldsymbol{e_2} = [\boldsymbol{e_2^1}, \mathbf{0_K}, ..., \boldsymbol{e_2^J}]$. Then, $P_F(j) \in \mathbb{R}^{K \times K}$ ($\mathbb{R}^{J \cdot \frac{K}{J} \times K}$) extracts the embedding $\boldsymbol{e_i}^{F(j)}$. For example, $P_2$ is of the form: $P_2 = \begin{bmatrix} 0_{\frac{K}{J}} \\ I_{\frac{K}{J}} \\ 0_{\frac{K}{J}} \\ \cdot \\ \cdot \\ 0_{\frac{K}{J}} \end{bmatrix}$

**Interaction matrix form:** For every FM-based model, M takes the following form: $M = \frac{1}{2} \cdot \begin{bmatrix} 0_K & M_{1,2} & M_{1,3} & ... & M_{1,J} \\ M_{1,2}^T & 0_K & M_{2,3}^T & ... & M_{2,J}^T \\ \cdot & \cdot & \cdot & ... & \cdot \\ \cdot & \cdot & \cdot & ... & \cdot \\ \cdot & \cdot & \cdot & ... & \cdot \\ M_{1,J}^T & M_{2,J}^T & M_{3,J}^T & ... & 0_K \end{bmatrix}$

# B EXPERIMENTAL SETTINGS AND IMPLEMENTATION DETAILS

## B.1 Performance Evaluation

*B.1.1 CTR Prediction.* **Implementation details** We implemented our framework on FmFM public code-base [3], and used their data processing and baselines implementation. We split the data as described in FmFM paper, and applied the same splits for all the models. For our implementation, we adopted FmFM model implementation, and changed the main component to a Tensorflow linear layers, an activation between them, and bias for the per-field projection layer. For the reported framework instance performance, in Criteo we used ReLU activation and a full rank per-field projection layer, and for Avazu we used GELU activation and factor of 5 of the full rank dimension. In this settings, we did not use a dropout between the layers.

**Hyperparameters search** We found that the hyperparameters in this task are crucial, included the standard deviation of the weights initialization. Therefore, we performed an hyperparameters search both for our framework and for FmFM, though obtained that they should use the same hyperparameters. Thus, we applied this hyperparameters for the deep baselines too. We conducted this hyperparameters comparison on a specific seed, and observed an improvement over the reported hyperparameters in all the baseline models. The results can be found in Table 5. The exact hyperparameters of each baseline is reported in Table 6.

**Table 5: CTR prediction (ROC-AUC %): Hyperparameters comparison. I results reported by authors, II results of running the model on our data split with paper's hyperparameters, III result of running the model on our data split with our tuning for the model. DeepFwFM results are without a pruning mechanism.**

| | Criteo | | |
|---|---|---|---|
| Model | I | II | III |
| FmFM | 81.09 | 81.11 | 81.166 |
| DeepFmFM | - | - | 81.175 |
| DeepFwFM | 81.16 | 81.105 | 81.130 |
| Ours | - | - | 81.246 |

| | Avazu | | |
|---|---|---|---|
| Model | I | II | III |
| FmFM | 77.63 | 77.343 | 78.635 |
| DeepFmFM | - | - | 78.647 |
| DeepFwFM | 78.93 | 78.519 | 78.596 |
| Ours | - | - | 78.701 |

[3] https://github.com/yahoo/FmFM

**Table 6: CTR prediction baselines hyperparameters: learning rate($\gamma$), regularization strength ($\lambda$), standard deviation of the weights normal initialization ($\sigma$), batch size ($bs$). I results reported by authors, II results of running the model on our data split with paper's hyperparameters, III result of running the model on our data split with our tuning for the model.**

| Model | Criteo | | |
|---|---|---|---|
| | I | II | III |
| FmFM | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.01, bs = 1024$ | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.01, bs = 1024$ | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.2, bs = 2000$ |
| DeepFmFM | - | - | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.2, bs = 2000$ |
| DeepFwFM | $\gamma = 1e^{-3}, \lambda = 3e^{-7}, \sigma = 0.01, bs = 2048$ | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.01, bs = 2048$ | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.2, bs = 2000$ |
| Ours | - | - | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.2, bs = 2000$ |

| Model | Avazu | | |
|---|---|---|---|
| | I | II | III |
| FmFM | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.01, bs = 1024$ | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.01, bs = 1024$ | $\gamma = 1e^{-3}, \lambda = 2e^{-6}, \sigma = 0.2, bs = 5000$ |
| DeepFmFM | - | - | $\gamma = 1e^{-3}, \lambda = 2e^{-6}, \sigma = 0.2, bs = 5000$ |
| DeepFwFM | $\gamma = 1e^{-3}, \lambda = 6e^{-7}, \sigma = 0.01, bs = 2048$ | $\gamma = 1e^{-4}, \lambda = 1e^{-5}, \sigma = 0.01, bs = 2048$ | $\gamma = 1e^{-3}, \lambda = 2e^{-6}, \sigma = 0.2, bs = 5000$ |
| Ours | - | - | $\gamma = 1e^{-3}, \lambda = 2e^{-6}, \sigma = 0.2, bs = 5000$ |

**Table 7: General tabular datasets**

| Dataset | Task | #Fields | #Categ. | #Numer. | Size | #Positives | #Negatives | % of positives |
|---|---|---|---|---|---|---|---|---|
| Income | Binary | 14 | 8 | 6 | 32,561 | 7,841 | 24,720 | 24.08 |
| Bank | Binary | 16 | 9 | 7 | 45,211 | 5,289 | 39,922 | 11.7 |
| BlastChar | Binary | 20 | 17 | 3 | 7,043 | 1,869 | 5,174 | 26.54 |
| Credit | Binary | 29 | 0 | 29 | 284,807 | 492 | 284,315 | 0.17 |
| Forest | Binary | 49 | 0 | 49 | 495,141 | 283,301 | 211,840 | 57.22 |
| HTRU2 | Binary | 8 | 0 | 8 | 17,898 | 1,639 | 16,259 | 9.16 |
| KDD99 | Binary | 39 | 3 | 36 | 494,021 | 97,278 | 396,743 | 19.69 |
| Shoppers | Binary | 17 | 2 | 15 | 12,330 | 1,908 | 10,422 | 15.47 |
| Philippine | Binary | 308 | 0 | 308 | 5,832 | 2,916 | 2,916 | 50 |
| QSAR Bio | Binary | 41 | 0 | 41 | 1,055 | 356 | 699 | 33.74 |
| Shrutime | Binary | 11 | 3 | 8 | 10,000 | 2,037 | 7,963 | 20.37 |
| Spambase | Binary | 57 | 0 | 57 | 4,601 | 1,813 | 2,788 | 39.4 |
| Volkert | Multiclass(10) | 147 | 0 | 147 | 58,310 | - | - | - |
| MNIST | Multiclass(10) | 784 | 784 | 0 | 60,000 | - | - | - |

**Table 8: Saint datasets - Architecture hyperparameters search. $d_{factor}$ and $l_{factor}$ stands for factors of the full rank dimensionality of $E_F$ and $t_F$ respectively.**

| Parameter | Values |
|---|---|
| $d_{factor}$ | $\{0.1, 0.25, 0.5, 0.75, 1, 2, 3, 4, 5\}$ |
| $l_{factor}$ | $\{1, 2, 3, 4\}$ |
| Activation | {ReLU, Quadratic, GELU} |
| Dropout | $\{0, 0.1, 0.25, 0.5, 0.75\}$ |

*B.1.2 General tabular data.* **Datasets** Detailed information on the evaluated datasets is reported in Table 7. It can be seen that the evaluated datasets are diverse, in terms of their size and the amount of fields, and contains both categorical and numerical features. Each of these datasets is publicly available from either UCI [4] or AutoML [5]. We use the exact processing as is SAINT implementation, i.e. all the continuous features are Z-normalized, and all categorical features are label-encoded before the data is passed on to the embedding layer.

**Implementation details:** We implemented our framework in the SAINT public code-base [6]. We used dropout before $U_F$ and after the activation, and a bias for $U_F$ and for the numerical $E_F$ and $t_F$. As in factorization machine methods, we observe minor benefits from adding linear per-field terms to the final logits.

**Training:** Our optimization hyper-parameters follow SAINT. We used the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, decay = 0.01, and with a learning rate of 0.0001 with batches of size 256 (except for MNIST, which they use a batch size of 32). We trained for 100 epochs. We split the data into 65%, 15%, and 25% for training, validation, and test splits, respectively.

**Hyperpararmter search:** We performed architecture search on the hyperparameters presented in Table 8 with seed=1. We selected the best model by the validation set. Our experiments employed a fixed embedding and projection sizes to all of the fields (although tuning this might improve results).

*B.1.3 Activation study.* Rendle et al. [19] public code is optimized for simplicity and not for efficiency, therefore we have implemented an efficient keras version of it, using Dot layer, and adjusted the hyperparamters to fit their reported results. We used Adam optimizer, batch size of $10k$ and L2 regularization strength of $2.5e^{-7}$, while kept the negative sampling on 8, and a learning rate of 0.002. Next, since matrix factorization is a special case of our framework, we implemented it by our framework, using quadratic activation and the eigen vectors and values of $0.5 \cdot \begin{bmatrix} 0_K & I_K \\ I_K & 0_K \end{bmatrix}$ for the per-field projections and for the final classification layer, respectively. The eigenvectors were partitioned with respect to the fields. When freezing these weights, we get an exact implementation of a dot product, which also reflected by a replication of the performance between both of the implementations. In order to explore the ability of other activation to approximate the results of a dot product based model, we used the eigen decomposition as an initialization and regularization, and searched the best regularization strength for each activation. The regularization strength of the reported results are reported in Table 9. We reported this experiments results on embedding sizes of 16 and 192, the minimal and maximal embedding sizes that were tested in Rendle et al. [19].

**Table 9: GELU as adapted activation - regularization strength of the reported results**

| Emb. size | MF initialization & regularization | | |
|---|---|---|---|
| | ReLU | GELU | Quad. |
| 16 | $\lambda = 1e^{-5}$ | $\lambda = 1e^{-5}$ | $\lambda = 1$ |
| 192 | $\lambda = 5e^{-3}$ | $\lambda = 5e^{-4}$ | $\lambda = 1$ |

# C RESULTS PER DATASET

**Performance comparison:** The performance comparison per dataset are reported in Table 10. The results are averaged over seeds 1-10. We reported also the standard deviation.

**Ablation studies:** Table 11 reports the results of the ablation studies per dataset.

[4] http://archive.ics.uci.edu/ml/datasets.php
[5] https://automl.chalearn.org/data
[6] https://github.com/somepago/saint

**Table 10: Performance comparison per dataset, Averaged results over seeds 1-10.**

| | ROC-AUC | | | | | | |
|---|---|---|---|---|---|---|---|
| | Credit | HTRU2 | QSAR Bio | Shrutime | Spambase | Philippine | KDD99 |
| SAINT | 97.89 ±0.85 | 97.98 ±0.42 | 93.28 ±1.4 | 86.37 ±1.07 | 98.37 ±0.44 | 81.22 ±1 | 100 ±0 |
| SAINT-i | 97.92 ±0.85 | 98.02 ±0.35 | 92.9 ±1.79 | 86.23 ±1.23 | 98.2 ±0.34 | 80.56 ±1.48 | 100 ±0 |
| SAINT-s | 98.09 ±0.72 | 97.99 ±0.44 | 92.64 ±1.41 | 86.18 ±1.11 | 98.28 ±0.21 | 78.46 ±1.81 | 100 ±0 |
| Ours | 97.01 ±1.31 | 98.13 ±0.39 | 92.89 ±1.48 | 86.47 ±1.15 | 98.31 ±0.39 | 80.81 ±1.88 | 100 ±0 |

| | ROC-AUC | | | | | Accuracy | |
|---|---|---|---|---|---|---|---|
| | Bank | Blastchar | Forest | Shoppers | Income | Volkert | MNIST |
| SAINT | 92.93 ±0.4 | 84.03 ±0.93 | 98.2 ±0.24 | 93.2 ±0.47 | 91.44 ±0.36 | 69.57 ±0.4544 | 97.86 ±0.1035 |
| SAINT-i | 93.12 ±0.4 | 83.98 ±0.9 | 94.74 ±0.97 | 92.89 ±0.41 | 91.4 ±0.38 | 70.34 ±0.2749 | 97.71 ±0.1774 |
| SAINT-s | 93.67 ±0.25 | 83.99 ±0.98 | 99.71 ±0.02 | 93.06 ±0.51 | 91.56 ±0.3 | 49.71 ±3.156 | 94.16 ±0.3232 |
| Ours | 93.43 ±0.27 | 84.09 ±0.94 | 99.49 ±0.02 | 92.97 ±0.48 | 91.54 ±0.36 | 70.22 ±0.3784 | 97.53 ±0.1266 |

**Table 11: Ablations per dataset**

| | ROC-AUC | | | | | | | | | | | Accuracy | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Credit | HTRU2 | QSAR Bio | Shrutime | Spambase | Philippine | KDD99 | Bank | Blastchar | Forest | Shoppers | Income | Volkert | MNIST |
| Ours | 97.73 | 98.42 | 94.12 | 86.6 | 98.44 | 82.47 | 100 | 93.09 | 82.38 | 99.47 | 93.53 | 91.62 | 70.35 | 97.7 |
| Low-rank | 96.95 | 98.57 | 93.49 | 86.69 | 98.09 | 81.41 | 100 | 92.88 | 82.41 | 96.35 | 93.2 | 91.67 | 65.27 | 92.66 |
| Shared | 95.93 | 98.42 | 93.48 | 85.51 | 98.02 | 81.02 | 100 | 92.09 | 82.3 | 99.42 | 92.96 | 91.5 | 62.44 | 78.5 |
| Deep | 97.48 | 98.25 | 93.54 | 85.39 | 98.24 | 81.47 | 100 | 93.11 | 82.3 | 99.58 | 93.54 | 91.77 | 71.9 | 93.24 |
| Quadratic | 97.73 | 98.47 | 93.81 | 86.6 | 98.11 | 82.47 | 100 | 92.95 | 82.38 | 94.84 | 93.4 | 91.62 | 66.93 | 97.7 |
| ReLU | 97.89 | 98.42 | 94.12 | 85.99 | 98.44 | 81.76 | 100 | 93.06 | 82.25 | 99.4 | 93.53 | 91.74 | 69.93 | 95.22 |
| GELU | 97.71 | 98.42 | 93.77 | 85.93 | 98.4 | 81.73 | 100 | 93.09 | 82.26 | 99.47 | 93.49 | 91.69 | 70.35 | 94.88 |