# Reproducibility of *Adaptive Factorization Network: Learning Adaptive-Order Feature Interactions*

Samo Pahor

August 31, 2021

**Abstract**

We attempt to reproduce the Adaptive Factorization Network model, which is a novel approach for context-aware prediction, specifically click-through-rate prediction. The model utilizes a logarithmic neuron layer, which is able to adaptively capture feature interactions of arbitrary order, thereby outperforming low-order models, such as logistic regression and factorization machines.

## 1 Introduction

This report covers the reproducibility of the findings published in the article *Adaptive Factorization Network: Learning Adaptive-Order Feature Interactions* [1]. Our main objective is attempting to verify that the Adaptive Factorization Network (AFN) model achieves better performance than proposed baselines on four popular test datasets. To adequately present the model reproducibility, we first present the model itself, in particular its novel use of the logarithmic neuron layer. Afterwards, we also present the main comparative baselines. In the following section, we describe the four datasets used to perform model training and evaluation. Finally, we present our results and conclude the reproducibility report with our key findings. Notably, we avoid using the codebase featured in the original paper, opting instead to write our own implementation of all relevant algorithms.[1]

## 2 Proposed model description

In this section, we describe the proposed machine learning model: Adaptive Factorization Network. The model is composed out of 5 distinct parts:

[1] All code written for the purpose of this report is available at `https://github.com/Kahno/mlds2_reproducibility`.

- Input Layer,
- Embedding Layer,
- Logarithmic Transformation Layer,
- Feed-forward Hidden Layers,
- Prediction.

We describe each part in the following subsections.

## 2.1 Input Layer

The input layer of the model absorbs both categorical as well as numeric features. While the numeric features can be used directly in the following layers, the categorical features first require a mapping to a dense numeric space, so they are forwarded to the embedding layer.

## 2.2 Embedding Layer

Categorical features obtained from the input layer are embedded into dense numeric vectors. For the model to work, our feature embedding have to be positive real values, because they are exposed to a logarithmic transformation in the following layer. In addition, it is advisable to increment embedding with zero values by a small positive amount (e.g. $1e{-}7$) to avoid numerical instability in the following layer.

## 2.3 Logarithmic Transformation Layer

The logarithmic transformation layer is the main innovation of the AFN model. It gives the model the ability to approximate an arbitrary order of feature interactions by utilizing the concept of *logarithmic neurons*. For a given set of positive embeddings $\{e_1, e_2, \cdots, e_m\}$, the output of the $j$-th logarithmic neuron equals:

$$y_j = \exp(\sum_{i=1}^{m} w_{ij} \ln e_i) = e_1^{w_{1j}} \odot e_2^{w_{2j}} \odot \cdots \odot e_m^{w_{mj}}.$$

In the above equation, $w_{ij}$ are the weights of the $j$-th neuron, which determine the interaction of the $i$-th feature. These weights are applied as power terms to the corresponding embeddings in an element-wise fashion. Similarly both the exponential as well as logarithmic functions are applied in an element-wise fashion, while $\odot$ denotes an element-wise product operation.

### 2.4 Feed-forward Hidden Layers

The logarithmically transformed embeddings are finally fed into a series of fully connected hidden layers. The outputs of each logarithmic neuron get concatenated into a single vector:

$$z_0 = [y_1, y_2, \cdots, y_N]$$

which gets fed into $L^2$ hidden layers.

$$z_1 = ReLU(W_1 z_0 + b_1)$$

$$\cdots$$

$$z_L = ReLU(W_L z_{L-1} + b_L)$$

### 2.5 Prediction

The final prediction is obtained by taking the output of the final fully connected hidden layer, multiplying it with a weight vector, adding a bias and finally applying the logistic function:

$$\hat{y} = \sigma(w_p^\top z_L + b_p).$$

## 3 Baseline models

We compare AFN with the following baseline methods, which are likewise reproduced from scratch in Tensorflow 2:

- Linear Regression,
- Factorization Machines [3],
- DeepFM [2].

While the original article featured comparisons to various other models, we limit ourselves to the two basic baselines (LR and FM) and a popular FM upgrade (DeepFM) to preserve simplicity. We are also able to eliminate an additional level of bias by developing the mentioned models from scratch, which would be unfeasible for all models listed in the original article. The following sections feature short descriptions of each baseline model.

[2] Here, $L$ is a model hyperparameter.

### 3.1 Logistic Regression

Logistic regression is a type of binary classification approach that computes a sum of weigh values for each observed feature (we can call this part the linear term) and adds it to a global bias term. The final prediction is obtained by applying the logistic function to the sum of the bias and linear terms. Feature weights as well as the bias are trained via gradient descent or similar approaches.

$$\hat{y} = \sigma(w^\top x + b)$$

### 3.2 Factorization Machines

Factorization machines can be considered an upgrade to linear regression. Similarly to LR, our sum consists of the linear and bias terms. FMs differ by also computing the interaction term, which equals the sum of all combinations of factor vectors of each feature. Factor vectors are obtained from a separate embedding weight matrix.

$$\hat{y} = \sigma(\sum_{i<j}^{m} \langle e_i, e_j \rangle + w^\top x + b)$$

### 3.3 DeepFM

If FMs are considered an upgrade to LRs, then DeepFM can be treated as an upgrade to FMs. Besides considering the prediction from the FM model, DeepFM also trains a separate deep neural network that outputs a prediction based on the same embedding already used for the FM term. The predictions are added together and a sigmoid activation is applied similar to the above cases.

$$y_{\text{FM}} = \sum_{i<j}^{m} \langle e_i, e_j \rangle + w^\top x + b$$

$$\hat{y} = \sigma(y_{\text{FM}} + y_{\text{DNN}})$$

## 4 Datasets

We performed the reproducibility experiments by training the AFN model as well as the baseline models on the following four datasets:

- Criteo dataset,
- Avazu dataset,
- Frappe dataset,
- Movielens dataset.

Each dataset contains anonymized features. Criteo contains both categorical as well as numerical features, while the rest only feature categorical features. All datasets are available in the LIBSVM format. Respective dataset details can be seen in the table below.

| Dataset | Criteo | Avazu | Frappe | Movielens |
|---|---|---|---|---|
| No. of train samples | 33,003,326 | 28,300,276 | 202,027 | 1,404,801 |
| No. of test samples | 4,587,167 | 8,085,794 | 28,860 | 200,686 |
| No. of features | 40 | 22 | 10 | 3 |
| Also numerical | Yes (12) | No | No | No |

# 5 Results

For each model, we ran the entire training procedure three times and reported the average value.[3] We evaluated model performance using:

- area under the ROC curve (AUC),
- logloss.

The results can be seen in the table below. The AFN model indeed appears to be the most successful model across all datasets. Also, the general performance trend is favored towards models which have the ability to model higher orders of feature interaction; LR, which is a first-order model, generally exhibits the lowest performance. Conversely, DeepFM, which has the ability to model both low and high order interactions due to its dual nature, exhibits the best performance out of the three baseline models.

| Model | Criteo | | Avazu | | Frappe | | Movielens | |
|---|---|---|---|---|---|---|---|---|
| | AUC | logloss | AUC | logloss | AUC | logloss | AUC | logloss |
| LR | 0.729 | 0.508 | 0.738 | 0.373 | 0.556 | 0.726 | 0.778 | 0.564 |
| FM | 0.747 | 0.497 | 0.740 | 0.373 | 0.744 | **0.583** | 0.772 | 0.544 |
| DeepFM | 0.746 | 0.497 | 0.742 | 0.371 | 0.711 | 0.590 | 0.792 | 0.524 |
| AFN | **0.757** | **0.489** | **0.749** | **0.368** | **0.832** | 0.640 | **0.841** | **0.480** |

# 6 Conclusion

Based on our experiments, we can conclude that AFN is indeed able to outperform popular baseline models, based on AUC as well as logloss. However, an important downside of the AFN model is its substantially higher training

[3] This approach matches the original article.

time, compared to all three baseline models. While LR, FM and DeepFM all had roughly the same average training time, AFN's training time was **3.5** times higher. The original article doesn't focus on the training time aspect of AFNs, and only briefly compares it to one of the baselines.

Increasing the number of logarithmic neurons further increases the time discrepancy between AFN and the baselines, which indicates that this is the performance bottleneck. Due to this, as well as the higher model complexity of AFN, we are not entirely convinced that the model can reliably outperform simpler models in a production setting, where prediction and model serving speed is also a factor.

# References


[1] W. Cheng, Y. Shen, and L. Huang. Adaptive factorization network: Learning adaptive-order feature interactions, 2020.

[2] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. DeepFM: A factorization-machine based neural network for CTR prediction, 2017.

[3] S. Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000, 2010.