XDeepFM

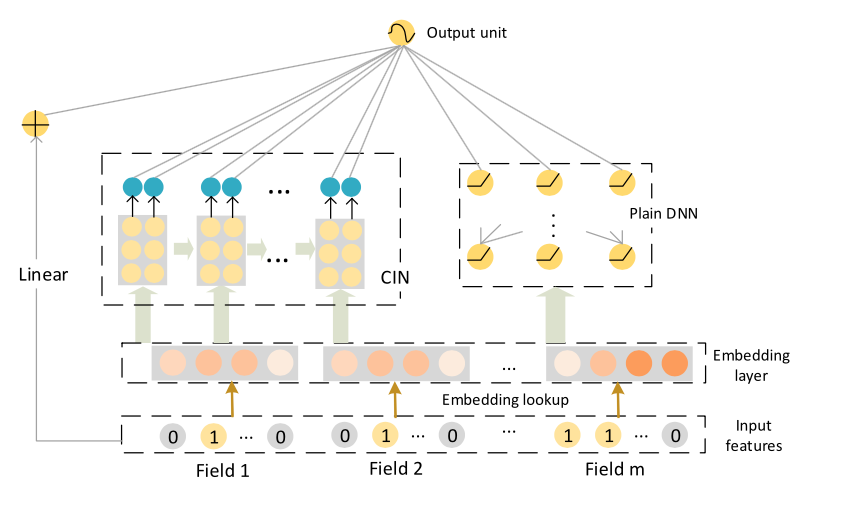
Extreme Deep Factorization Model utilizes deep learning along with a factorization machine. The model provided the highest accuracy among all the implemented models. It combines feature interactions, both implicit and explicit.

The model was primarily used for Recommender Systems. In recommender systems with a very large number of features to train, the results can get very skewed due to the huge number of influences the features have. In certain data sets, the dimension of the features is huge but the input features or actual usable features are very sparse. Other alternatives for classic recommender systems with sparse input features is logistic regression with “Follow the Regularized Leader”.

XDeepFM provides a very high accuracy when it comes to cross features or multi-way features. These are features that combine categorical raw features to give very specific training scenarios. However, cross feature engineering has disadvantages.

* Getting significant information or trainable data from cross features is accompanied by a high cost. The computation is heavy and the exploratory data analysis needs to find out specific patterns to choose the features to enable cross features. This is specific to each dataset and data scientists have their work cut out.
* In massive datasets with hundreds of features, it is not possible to extract all cross features manually to test for accuracy.
* Finally, the patterns that data scientists pick out for cross featuring can miss out on many hidden or unseen features which may have been actually significant to improving the accuracy.

Due to these problems, we present with a factorization machine. The machine embeds each feature in a latent vector to create pairwise interactions of every feature.

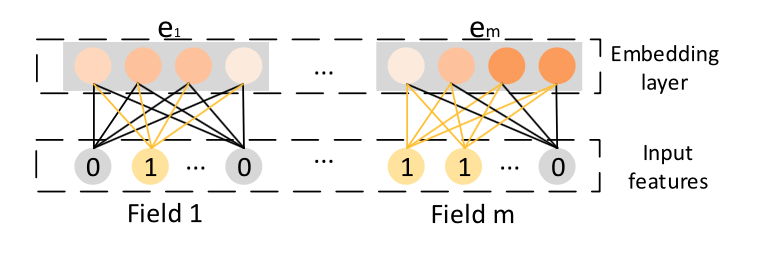


Factorization Machine

The factorization machine is a vital part of XdeepFM. It helps us consolidate important features from the model rather than letting data scientists do EDA and hand pick the features. The machine compensates for scarcity of the input model and dataset.

The machine works by creating a matrix of n dimensions where n is the number of user inputs. However, majority of the matrix is zeroes as the input is scarce. If k is the number of features we need to consolidate, we cross multiply the n matrices to get the final matrix with embedded feature values.

In our model, an embedding layer is applied on the raw input to compress the sparse matrix to a dense low dimensional matrix. This reduces the cost of training by a significant margin.



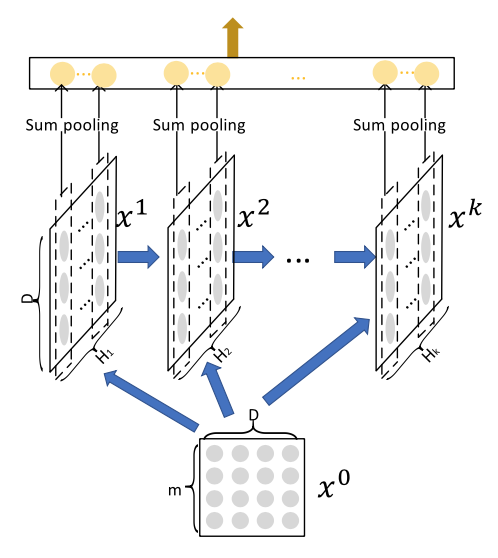
Compressed Interaction Network

While the factorization machine learns features implicitly, the Compressed Interaction Network learns other features explicitly and the extent of growth grows with the depth of the network. The interactions for the CIN will stay at a constant level of complexity and will not grow with interaction depth.

The Compressed Interaction Network takes features from both Convolutional Neural Networks and Recurrent Neural Networks.

Convolutional Neural Networks work based on moving a filter along the network layer. By taking the dot product, it consolidates the entire layer into a required dimension that can be controlled by the sliding window filter. When it is passed upon multiple layers, we get multiple outputs that can be stacked on one another. These outputs are called single depth slice. The CNN then has a very cost-effective method called pooling. Essentially, it takes a single depth layer and applies a logical relationship over an entire section of the depth. This ensures a lower dimension and easier calculation. It also reduces the number of parameters required.

In our model, the CIN does pooling at each logical layer to reduce the number of parameters. After the pooling is done, it passes the output as the input to the next layer which encompasses the basic structure of a Recurrent Neural Network.

With our dataset, the hidden layer size is a 128 x 128 matrix and the cross layer is a 3D matrix with the same dimensions. We have set the learning rate to 0.001 so as to not overfit with just 1 epoch. This ensures efficiency of the model and at the same time, does not spend additional resources like LGBM on HasDetections.