Three Unique Architectures for Deep Learning-Based Recommendation Systems

By Martin Anderson

First published ***November 10th, 2021*** *at:*
https://www.width.ai/post/three-deep-learning-based-recommendation-systems
Web-archived version

Deep learning-based recommendation architectures tend to break tasks down into manageable sections, adopting multiple methodologies, in order to compensate for the shortcomings of any single approach.

High-level extraction architectures are useful for categorization, but lack accuracy. Low-level extraction approaches will produce committed decisions about what to recommend, but, since they lack context, their recommendations may be banal, repetitive or even recursive, creating unintelligent 'content bubbles' for the user. High level architectures cannot 'zoom in' meaningfully, and low-level architectures cannot 'step back' to understand the bigger picture that the data is presenting.

In this article we'll take a look at three unique approaches that reconcile these two needs into effective and unified frameworks suitable for recommender systems. Two of these systems focus on both collaborative filtering and content based filtering, while the last one is used strictly as a content based solution.

Wide & Deep Learning Recommendation Systems

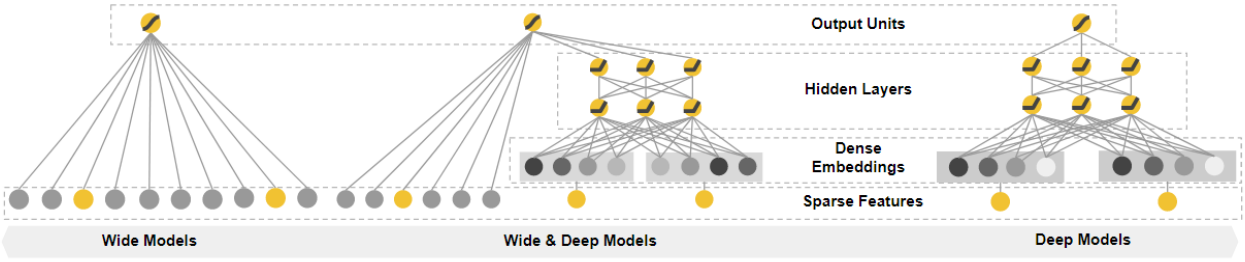
To usefully interpret a customer-input variable as a factor for a recommendation, an effective recommendation system needs to understand the potential *scope* of suitable responses to that input.

For instance, if the system wants to push a result in a domain such as 'used automobiles', there are many, many domain steps between 'car' (the dealership sells other types of vehicle too) and 'Toyota Corolla E140 Narrow'.

Mapping customer variables to the correct *band of specificity* in a model's insight scheme is problematic in most architectures, because training usually either follows a gradient paradigm that has no default mechanism to acknowledge a milestone in specificity (i.e. the jump from 'compact car' to 'subcompact car'); or else operates on purely statistical principles designed to take a customer directly to a result ('Toyota Corolla E140 Narrow'), when they might have wanted to start their exploration at a slightly higher level ('subcompact cars'), or even a much higher level ('Japanese cars').

Memorization And Generalization In A Single Framework

In 2016 Google addressed this problem by creating the Wide & Deep Learning model for recommender systems. The Wide & Deep framework reconciles two models, which address, respectively, memorization and generalization.

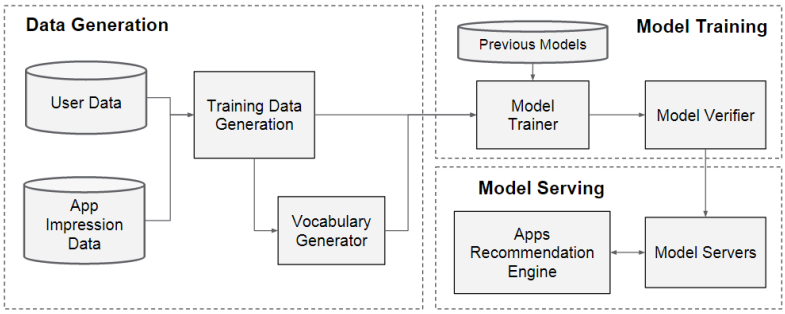


Source: https://arxiv.org/pdf/1606.07792.pdf

Memorization is the process by which a system examines how frequently features or items crop up in relation to each other, and stores this as a pattern of historical data. Under this approach, a user may well end up in the aforementioned 'content bubble', though the recommendations will be faithful to the presented data that the user has generated.

Generalization explores new possibilities for recommendations based on more abstract, higher-level relationships between items or item types, and can widen the range of recommendations at the risk of presenting material that the user may perceive as less relevant to their interests.

For memorization, Wide & Deep trains both a wide linear model together with a feed-forward deep neural network capable of generalization.



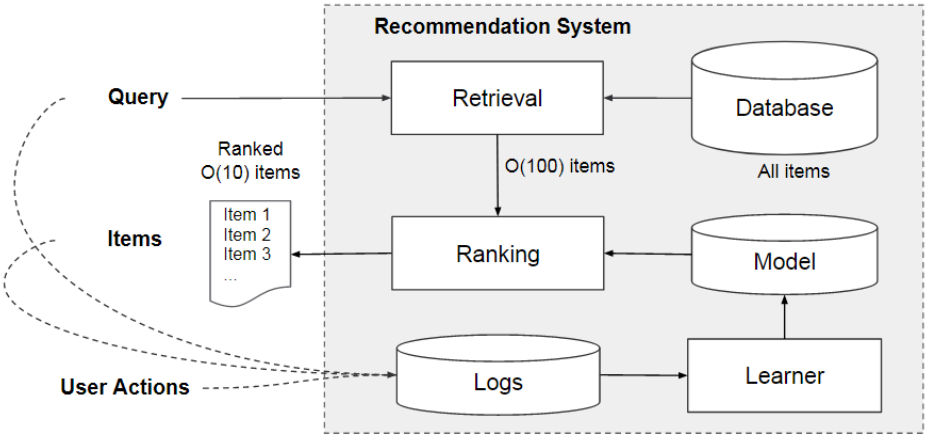
Retraining Google's Wide-Deep Model

The two facets of the system are melded via a weighted sum of their output logs acting as the prediction, subsequently fed to a single common logistic loss function for training. This represents *joint* rather than *ensemble* training, since the latter only collates data from two hermetic models at inference time.

The system is well-suited to high volume regression and classification tasks, and for utilizing sparse datasets with large categorical features from which many feature values can be derived. Wide & Deep is therefore a good match for recommender systems, as well as for the generation of ranking algorithms.

Wide & Deep was initially trained on 500 billion examples from install statistics at Google Play, with the results later used as a pre-weighted template in order to avoid retraining the model from zero when training subsequent datasets.

Under this framework, the 'wide' section of the architecture is represented by the cross-product transformations of apps installed on the users' machines, whereas the 'deep' element learns a 32-dimensional embedding vector for each feature in a category.



Production-Level Architecture

After concatenating the dense features and embeddings, the system obtains a model of around 1200 dimensions, which is then passed to three ReLU layers, and ultimately to the final output.

Before live deployment, the output from a trained model is checked in a staging environment against output from its predecessor, to ensure consistent performance and quality of results. Google ultimately implemented the model in Google Play, and reported a notable increase in engagement.

Wide & Deep Evolves

Besides open-sourcing the code for Wide & Deep (since forked by various others), Google incorporated the framework as a high-level API in TensorFlow. Since the original Wide & Deep research is trained on the proprietary Google Play dataset, it will be necessary to use your own data, or test your system on open source datasets.

Besides general take-up of Wide & Deep since 2016, a number of research teams and vendors have introduced innovations to the framework, notably in the wake of TensorFlow2, which appeared a year after the original Wide & Deep paper.

In April 2021, NVIDIA was able to accelerate a Wide & Deep training workflow for the Kaggle Outbrain Click Prediction Challenge dataset from 25 hours to ten minutes under a TensorFlow2 implementation, and with the use GPU preprocessing under NVIDIA's NVTabular feature engineering and preprocessing library.

Additionally, Wide & Deep has been forked to Keras, and adopted in a variety of commercial and research environnments, including hotel chain OYO, which adapted it to improve the company's ranking system.

xDeepFM: Improving DeepFM With Compressed Interaction Networks

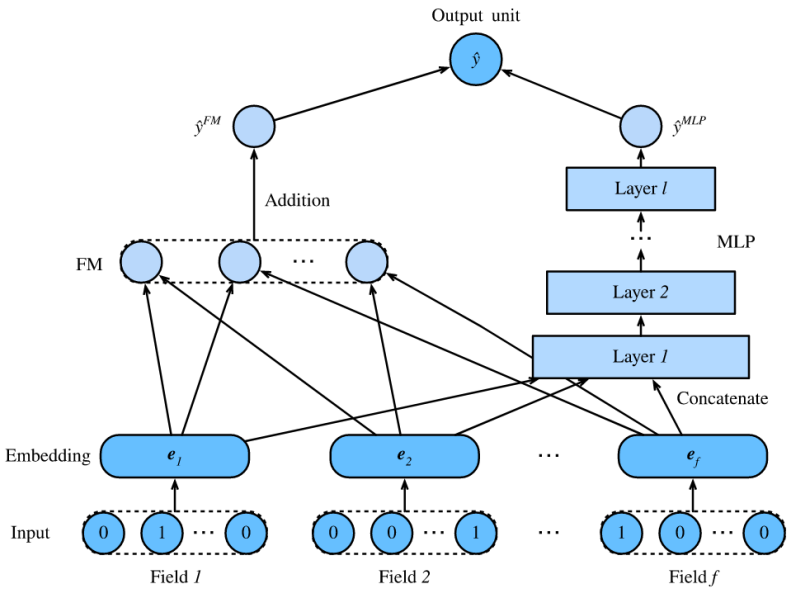
Efficient recommender systems require a performant and accurate method for generating 'feature interactions', where weight is assigned to certain factors in the data flow between the user and the item.

For example, click through rates (CTR) for food portals might increase around major meal times, indicating a 'feature' that can prompt a call-to-action in a recommender system that has some interest in this feature and this sector.

In general, linear deep learning models have difficulty learning feature interactions, and require a degree of manual input and labeling, making it hard either to automate feature generation or to roll out a recommender system at scale, in the absence of ongoing human intervention.

DeepFM

Addressing some of these issues, a 2017 paper led by Huawei researchers proposed a new architecture, entitled DeepFM, which combines the power of factorization with superior feature extraction.

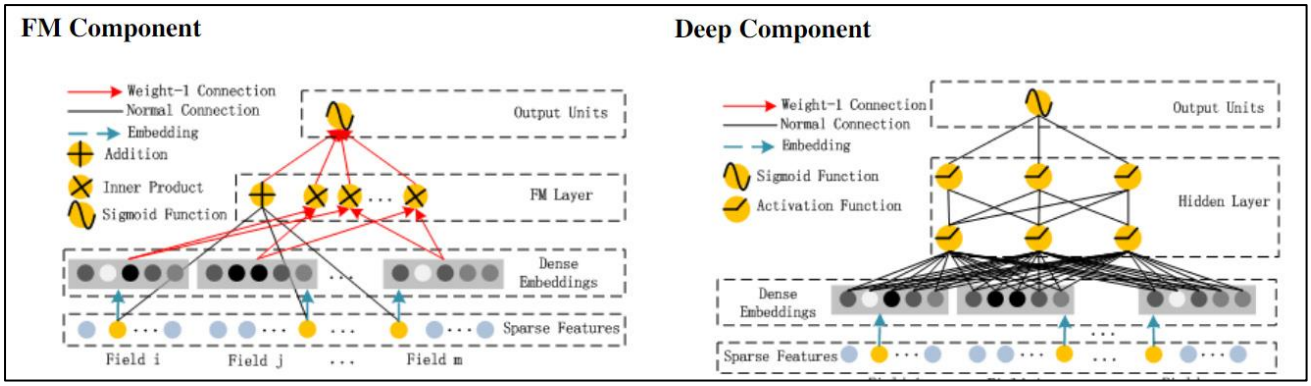


Source: https://d2l.ai/chapter_recommender-systems/deepfm.html

The deep learning component in DeepFM (which has since been ported to TensorFlow) is a multi-layered perceptron capable of identifying high-level features and non-linear interactions, while the factorization machine (FM) portion of the architecture captures low-level feature interactions.

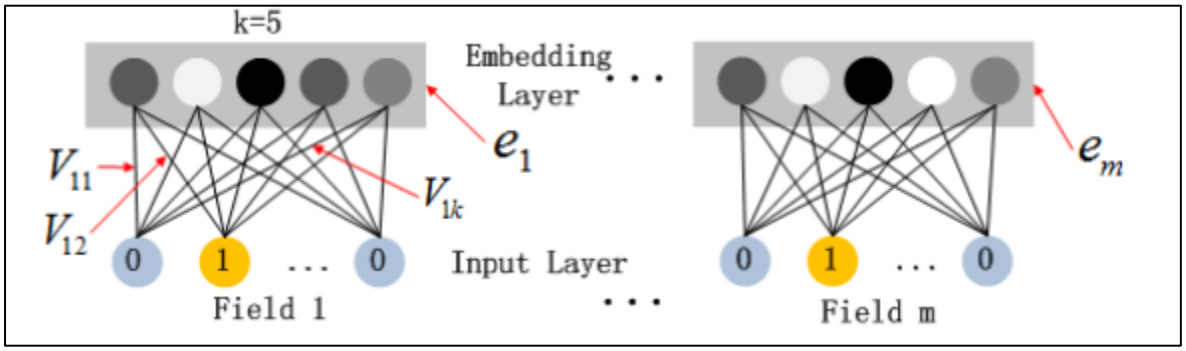
(It should be noted that it is also possible to use deep learning to capture lower-level features, as with the work expounded in Neural Factorization Machines for Sparse Predictive Analytics, in the same year that DeepFM was released)

In contrast to Google's Wide & Deep model, DeepFM utilizes a shared input that straddles the low-level and high-level extraction pipelines, obviating the need for complex feature engineering.



The architectures of the two components in DeepFM. Source: https://arxiv.org/pdf/1703.04247.pdf

The FM component is a factorization mechanism to derive feature interactions for recommendations, and operates well on sparse datasets, while the deep component is a feed-forward network that models higher-order interactions.



Model Input into Embedding Layers

The latent feature vectors derived from the FM data flow effectively operate as network weights which, once learned, are used to compress input field vectors to the embedding layers.

When tested against other eight other hybrid models performing click-through rate (CTR) prediction, the original DeepFM authors were able to improve on all models.

Table 2: Performance on CTR prediction.

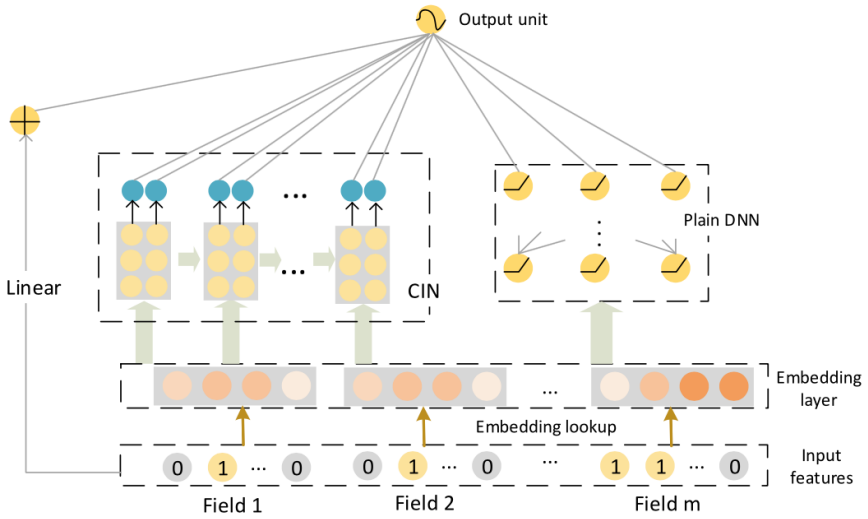
	Company∗		Criteo	
	AUC	LogLoss	AUC	LogLoss
LR	0.8640	0.02648	0.7686	0.47762
FM	0.8678	0.02633	0.7892	0.46077
FNN	0.8683	0.02629	0.7963	0.45738
IPNN	0.8664	0.02637	0.7972	0.45323
OPNN	0.8658	0.02641	0.7982	0.45256
PNN∗	0.8672	0.02636	0.7987	0.45214
LR & DNN	0.8673	0.02634	0.7981	0.46772
FM & DNN	0.8661	0.02640	0.7850	0.45382
DeepFM	**0.8715**	**0.02618**	**0.8007**	**0.45083**

Model Performance

The test was performed against the Criteo dataset of 45 million users' click records, split randomly into 90/10 training/test split; and from a 7-day subset of user click data from the Kaggle Company Dataset. Metrics used were AUC (Area Under ROC) and Logloss.

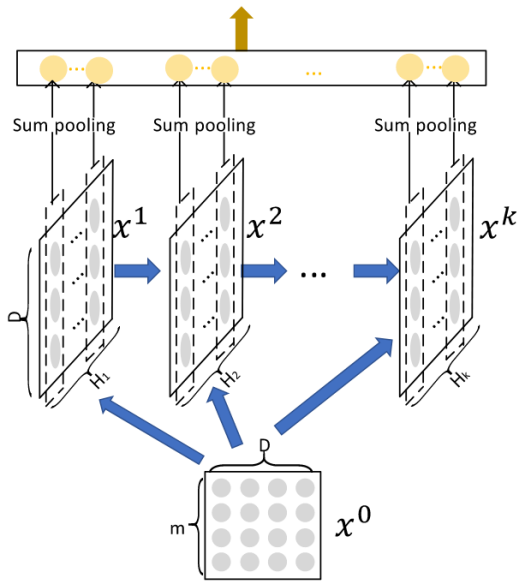
xDeepFM and CIN

A collaboration between Microsoft and the University of Science and Technology of China led in 2018 to the development of a new approach to factorization entitled Compressed Interaction Network (CIN).



xDeepFM Architecture With Compressed Interaction Network

CIN operates as a third layer, on top of the linear model (which extracts raw input features) and the neural network (which operates on dense feature embeddings).



The Compressed Interaction Network in xDeepFM generates interactions at a vector-wise level, rather than bit-wise, specifically evaluating high-order feature interactions. It also provides this functionality without ballooning the size of the network. Source: https://arxiv.org/pdf/1803.05170.pdf

xDeepFM with the CIN has a number of advantages over DeepFM:

- It operates at a vector level, where it can navigate data relationships, rather than a bitwise level, which lacks context.
- It scales transparently, without adding complexity.
- It evaluates high-level features explicitly.

$$\mathbf{X}_{h,*}^{k} = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} \mathbf{W}_{ij}^{k,h} (\mathbf{X}_{i,*}^{k-1} \circ \mathbf{X}_{j,*}^{0})$$

The hidden layers of CIN (every 'x' in the image) facilitate the learning of explicit feature interactions in xDeepFM. Source: https://towardsdatascience.com/extreme-deep-factorization-machine-xdeepfm-1ba180a6de78

CIN's relatively simple architecture is close in form to that of a recurrent neural network (RNN), and in other ways to a convolutional neural network (CNN). The authors of the xDeepFM paper have noted that network structures need not be that deep in order to obtain optimal results from a neural-based model. They established that the ideal depth setting for xDeepFM is level 3, indicating that the interactions and relationships learned and developed in training do not burrow any deeper than level four.

Source code for xDeepFM is available at GitHub. Microsoft has also included xDeepFM in its gathered repository of recommender systems at GitHub. The deep learning package DeepCTR has incorporated xDeepFM (also available in a Torch version). The code has also been exploited in a number of CTR prediction repositories.

TF-IDF

TF-IDF (*Term Frequency–Inverse Document Frequency*) is an older content based approach designed to stop low-frequency data points from becoming demoted (or even discarded) as outlier information.

Rescuing Infrequent Words From Obscurity With TF and IDF

The problem with assigning high rank to frequency is that some of the least valuable words occur most frequently (i.e. 'the', 'was', 'it', and other function words). Engineering manual features to filter these out or put them in a lower-ranked context is arduous and often counterproductive.

Additionally, manual feature intervention makes it difficult to port a recommender algorithm to other languages than the one used in initial development (usually English), because the 'stop words' identified during preprocessing will relate specifically to the original language of the system.

Therefore TF-IDF evaluates the data through two separate methodologies, and multiplies the results from each:

Term Frequency (TF)
Term Frequency counts the number of times a word appears in a document. For example, if the word 'plate' appears 8 times in a 100-word document, the Term Frequency is expressed as:

```
TFplate=8/100 [0.8]
```

The result is representative rather than absolute. Under LogBase2, doubling the number of word occurrences will increase but not double the TF value. Nonetheless, this is a quantitative result that tells us little about the qualitative value of the terms identified.

Inverse Document Frequency (IDF)
To evaluate the *information value* rather than the mere frequency of a term, IDF divides the total number of documents by the number of documents that actually contain a specific term. As 'inverse' suggests, this reverses the perception of term infrequency as a negative indicator, and, by analogy, considers infrequent terms to be the lexical equivalents of 'precious metals' – rare but valuable.

In high volume datasets, the IDF will balloon beyond useful measurement, but this can be remediated by taking a log of the output rather than the direct result. The IDF model assigns higher weight to informative words, so that the result will be somewhere between 0 (very common words) and 1 (highly informative words).

These values can be transformed into word vectors, and subsequently used in common NLP systems such as Support Vector Machine (SVM) and Naïve Bayes. Documents with similar vector features will have similar frequencies of a word, enabling pattern identification.

The Value Of TF-IDF

TF-IDF is a useful architecture for keyword extraction, but is also a natural contender for keyword-driven information retrieval systems. To an extent, the SEO industry's determination to infer insights into proprietary and well-guarded search engine methodologies has dominated online discussion of this approach.

Since Google has used TF-IDF at least to generate stop-words for its search algorithms over the last 10-15 years, it remains of great interest to the SEO optimization community – even if its relevance as a text-optimization measure is highly questionable.

Though Google's senior webmaster analyst John Mueller has discounted TF-IDF as a valid current SEO strategy, and instead recommends that users concentrate on improving the general quality of their content, Mueller tends to respond this way to all and any attempts to evaluate the influence of any single approach in Google's algorithm, leaving the exact value of TF-IDF strategies unknown, in terms of 'gaming' Google search.

For a local recommender system, however, TF-IDF remains a viable model, and one that is capable of inserting relevant but little-discovered content into the otherwise impenetrable content bubbles to which single-approach architectures are prone.

Using TF-IDF For Article Recommendations

TF-IDF is a great recommendation system for content focused recommendations, where the contents of an item are the only data used to find similarities and make recommendations. TF-IDFs use of tokenization methods like NLTK and Bert allow the model to have a very strong understanding of the text contents, which

with can be used in a model like cosine similarity to make recommendations on articles. This code example by Microsoft shows the ability to recommend articles based solely on information learned in the text and meta data in a huge article corpus.