3 Top of the Tide

Application of Deep Learning in Recommender Systems

With the introduction of Microsoft's Deep Crossing, Google's Wide&Deep, and a large number of excellent deep learning recommendation models such as Factorization-machine-supported Neural Network (FNN) and Product-based Neural Network (PNN) in 2016, the field of recommender systems and computational advertising has fully entered the era of deep learning. Today, deep learning models have become a well-deserved mainstream in the field of recommender systems and computational advertising. In Chapter 2, we discuss the structural characteristics and evolution of traditional recommendation models. After entering the era of deep learning, the recommendation model has made significant progress mainly in the following two aspects:

(1) Compared with traditional machine learning models, deep learning models have stronger expressivity and can mine more hidden patterns in data.
(2) The model structure of deep learning is very flexible. The model structure can be adjusted according to business use cases and data characteristics, so that the model fits perfectly with the application scenario.

From a technical point of view, the deep learning recommendation model learns from many deep learning techniques in computer vision, and in speech and natural language processing, and has undergone rapid evolution in its model structure.

This chapter summarizes the deep learning recommendation models with great influence in the recommendation field, constructs the evolution map between them, and introduces the technical characteristics of each model. The criteria for selecting a model should follow the following three principles:

(1) Models have great influence in industry and academia.
(2) The model has been successfully applied by well-known IT companies such as Google, Alibaba, and Microsoft.
(3) It plays an important role in the development of deep learning recommender systems.

Now, we will enter the "top of the tide" of recommender systems technology and explore how deep learning is transforming its application.

3.1 Evolution Graph of Deep Learning Recommendation Models

Figure 3.1 shows the evolution graph of the state-of-art deep learning recommendation models. Taking the Multi-Layer Perceptron (MLP) as the core, by changing

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

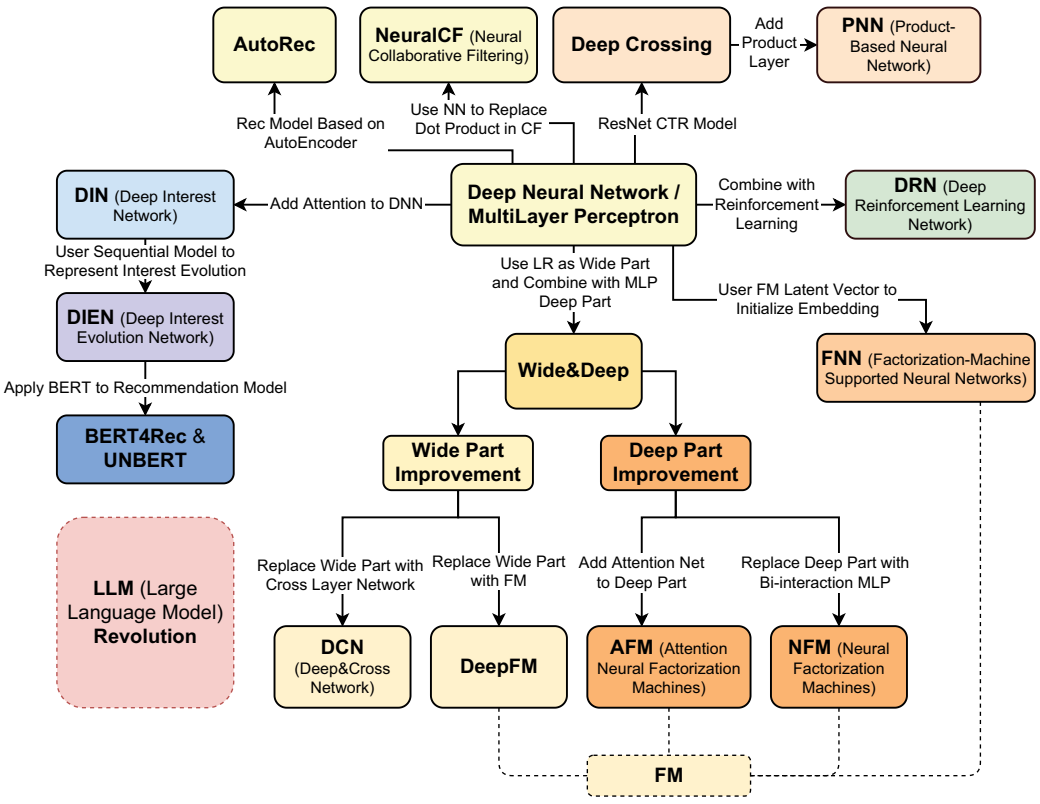

Figure 3.1 Evolution graph of mainstream deep learning recommendation models.

the structure of the neural network, a deep learning recommendation model with different characteristics is constructed. The main evolution directions are as follows:

(1) Changing the complexity of the neural network. From the simplest single-layer neural network model AutoRec (autoencoder recommendation) to the classic deep neural network structure Deep Crossing (deep feature crossing), the main evolutionary method is increasing the depth, that is, the number of layers and the structural complexity of the neural network.

(2) Changing the way features are crossed. The main change in this type of model is to enrich the way features are crossed in deep learning networks. For example, the NeuralCF (Neural Collaborative Filtering) changes the manner of user vector and item vector interoperability, and the PNN (Product-based Neural Network) model defines multiple types of feature vector cross operations.

(3) Ensemble models. This type of model mainly refers to the Wide&Deep model and its subsequent variants, for example, Deep and Cross, DeepFM, and so on. The idea is to improve the model's comprehensive ability by combining

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

two deep learning networks with different characteristics and complementary advantages.

(4) Evolving the FM models based on deep learning framework. The traditional recommendation model FM has many upgraded versions in the deep learning era, including NFM (Neural Factorization Machine), FNN (Factorization-machine-supported Neural Network), AFM (Attention neural Factorization Machine), and so on. These upgraded models improve FM in different directions. For example, NFM mainly uses neural networks to improve the capability of feature interaction on the second-order term. AFM is an FM model that introduces an attention mechanism, and FNN uses the results of FM to initialize the network.

(5) Combining attention mechanism and recommendation models. This type of model mainly applies the "attention mechanism" to the deep learning recommendation model, mainly including AFM, which combines FM and attention mechanism, and DIN (Deep Interest Network), which introduces the attention mechanism for CTR prediction.

(6) Combining sequence models and recommendation models. This type of model is characterized by using a sequence model to simulate the evolving trend of user behavior or user interest. The representative model is DIEN (Deep Interest Evolution Network).

These summaries clearly show the rapid development and broad thinking of deep learning models in recommendation applications. But each model is not a tree without roots, and its appearance is traceable. As with the structure of Chapter 2, we will explore together to learn the details of each model on the evolution graph as shown in Figure 3.1.

3.2 AutoRec: A Single Hidden-Layer Neural Network Recommendation Model

The AutoRec [1] model was proposed by the Australian National University in 2015. It combines the idea of AutoEncoder with collaborative filtering, and proposes a single hidden-layer neural network recommendation model. Because of its concise network structure and easy-to-understand theory, AutoRec is very suitable for learning as an entry model for deep learning recommender systems.

3.2.1 Theories of AutoRec

The AutoRec model is a standard autoencoder, and its basic theory is to use the co-occurrence matrix in collaborative filtering to complete the autoencoding of item vectors or user vectors. Then it uses the result of self-encoding to get the user's estimated rating of the item, and lastly performs recommendation ranking.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

Basics: Autoencoder

As the name suggests, an autoencoder is a model that is capable of "self-encoding" data. Whether it is image, audio, or text data, it can be converted into a vector for expression. Assuming the featured data vector is $\boldsymbol{r}$, the function of the autoencoder is to take the vector $\boldsymbol{r}$ through a reconstruction function. It will keep the obtained output vector as close to itself as possible after it is applied.

Assuming that the reconstruction function of the autoencoder is $h(\boldsymbol{r};\theta)$, then the objective function of the autoencoder is

$$\min_{\theta} \sum_{\boldsymbol{r} \in S} \left\| \boldsymbol{r} - h(\boldsymbol{r};\theta) \right\|_2^2 \tag{3.1}$$

where S is the entire training dataset.

After completing the training of the autoencoder, it is equivalent to storing the "essence" of all data vectors in the reconstruction function $h(\boldsymbol{r};\theta)$. In general, the number of parameters in the reconstruction function is much smaller than the number of dimensions of the input vector, so the autoencoder is functionally equivalent to data compression and dimensionality reduction.

Due to the "generalization" process, the output vector generated by the autoencoder will not be completely equivalent to the input vector, so it has a certain prediction ability for the missing dimensions. This is also the reason why the autoencoder can be used for recommender systems.

Assuming that there are m users and n items, the user will rate one or several of the n items, and the unrated items can be represented by the default value or the average score. Then the ratings of all m users can form a scoring matrix with the dimension of $m \times n$, which is also known as a co-occurrence matrix in collaborative filtering.

For an item i, the ratings of all the m users can form an m-dimensional vector $\boldsymbol{r}^{(i)} = (R_{1i}, \ldots, R_{mi})^T$. As mentioned in *Basics: Autoencoder*, the problem that AutoRec solves is to construct a reconstruction function $h(\boldsymbol{r};\theta)$, so that the sum of the squared residuals between all the score vectors generated by the reconstruction function and the original score vector is minimized (Eq. 3.1).

After obtaining the reconstruction function of the AutoRec model, the final recommendation list can be obtained through the process of score estimation and ranking. The following section will introduce two key points of the AutoRec model: the model architecture of the reconstruction function, and the process of using the reconstruction function to obtain the final recommendation list.

3.2.2 Network Structure of the AutoRec Model

AutoRec uses a single hidden-layer neural network to build a reconstruction function. As shown in Figure 3.2, the input layer of the network is the item's rating vector r, and the output is a multiclassification layer. The blue neurons in Figure 3.2 represent a k-dimensional hidden layer of the model, where $k \quad m$.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

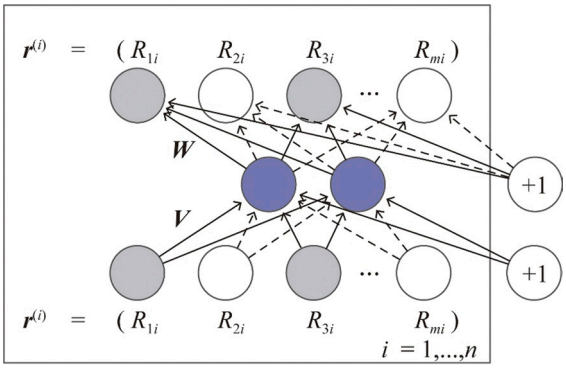

Figure 3.2 Architecture of the AutoRec model.

$\boldsymbol{V}$ and $\boldsymbol{W}$ in Figure 3.2 represent a parameter matrix from the input layer to the hidden layer and the hidden layer to the output layer, respectively. The reconstruction function is defined as follows,

$$h(\boldsymbol{r};\theta) = f\left(\boldsymbol{W} \cdot g\left(\boldsymbol{V}\boldsymbol{r} + \mu\right) + b\right) \tag{3.2}$$

where $f(\cdot)$ and $g(\cdot)$ are the activation functions of the output layer and the hidden layer, respectively.

In order to prevent overfitting of the reconstruction function, the L2 regularization term is added. Then, the AutoRec objective function becomes,

$$\min_{\theta} \sum_{i=1}^{n} \left\| \boldsymbol{r}^{(i)} - h\left(\boldsymbol{r}^{(i)};\theta\right) \right\|_{\mathcal{O}}^{2} + \frac{\lambda}{2} \cdot \left(\|\boldsymbol{W}\|_F^2 + \|\boldsymbol{V}\|_F^2 \right) \tag{3.3}$$

Since the AutoRec model is a standard three-layer neural network, the model can be trained using a gradient backpropagation approach.

Basics: Neuron, Neural Network, and Backpropagation

In this section, the basic concepts related to deep learning are mentioned many times, such as neurons, neural networks, and gradient backpropagation – the main training method of neural networks. We will briefly walk through these concepts.

Neuron, also known as Perceptron, is the same as a logistic regression unit from the model structure perspective. Here, we will use an example of a two-dimensional input vector to elaborate it. Assuming that the input vector of the model is a two-dimensional feature vector (x_1, x_2), the model structure of a single neuron is depicted in Figure 3.3.

In Figure 3.3, the elements in the blue circle can be viewed as a linear weighted summation, plus a constant bias b, and the final input can be expressed as follows

$$(x_1 \cdot w_1) + (x_2 \cdot w_2) + b$$

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

The entire blue circle in Figure 3.3 is a representation of the activation function. Its main role is to map an unbounded input variable to a normalized, bounded range of values. In addition to the sigmoid function introduced in Section 2.4, the other common activation functions are tanh, ReLU, and so on. Due to the limitation of the simple structure, the single neuron has poor fitting ability. Therefore, when solving complex problems, multiple neurons are often linked as a network, so that it can have the ability to fit any complex function. Such a network is what we often call a *Neural Network*. Figure 3.4 illustrates a simple neural network consisting of an input layer, a two-neuron hidden layer, and a single-neuron output layer.

In Figure 3.4, the neurons (blue circles) have the same structure as that of the perceptron described here. The inputs to neurons h_1 and h_2 are the feature vector (x_1, x_2) and the inputs to neuron o_1 are h_1 and h_2. Here, we show the simplest form of neural network. As the development of deep learning continues, researchers' exploration of different connection methods of neurons leads to different generations of deep learning networks with different characteristics.

After introducing the structure of a basic neural network, the next important question is how to train a neural network. We will start with two important concepts in the neural network training – *Forward Propagation* and Backpropagation.

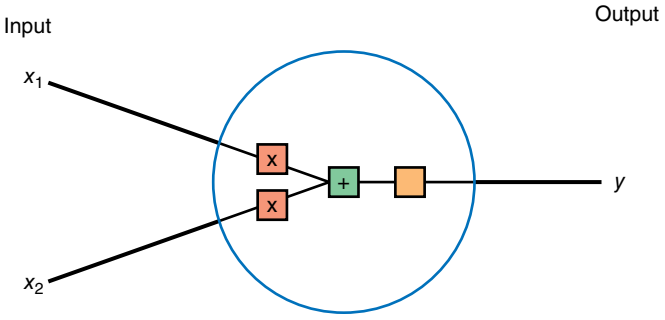

Figure 3.3 The model structure of a single neuron.

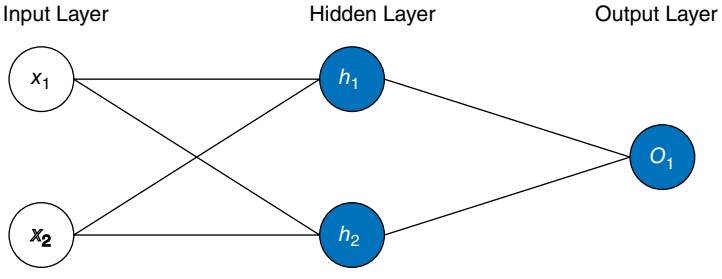

Figure 3.4 A simple neural network.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

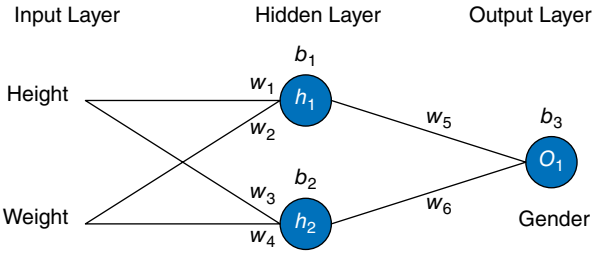

Figure 3.5 A schematic diagram of neural network structure and its weights.

The purpose of forward propagation is to obtain the predicted value of the model's input based on the current network parameters. This process is also often referred to as the model inference. After getting the predicted value, you can use the definition of the *loss function* to calculate the loss of the model. For the output layer neurons (o_1 in Figure 3.4), the gradient descent method can be used directly to calculate the gradient of the associated weights (that is, the weights w_5 and w_6 in Figure 3.5), so as to update the weights. But for the hidden layer, how could we use the gradient descent to update the parameters for the neurons in the hidden layer (for example, w_1 in Figure 3.5) based on the loss from the output layer?

It can be solved through the gradient backpropagation. The gradient backpropagation is used to derive model weights based on model loss utilizing the *chain rule*. As shown in the following equation, the gradient of the final loss function to the weight w_1 is obtained by multiplying the partial derivative of the loss function to the output of the neuron h_1 and the partial derivative of the output of the neuron h_1 to the weight w_1. That is, the final gradient is propagated back layer by layer, leading to the update of the weight w_1.

$$\frac{\partial L_{o_1}}{\partial w_1} = \frac{\partial L_{o_1}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$$

In the specific calculation, it is necessary to clarify the form of the final loss function and the form of the activation function of each layer of neurons, and then calculate the partial derivative according to the specific function.

To summarize, a neuron is the basic structure in neural networks. The specific implementation, mathematical expression, and training methods are consistent with logistic regression models. A neural network is a network formed by connecting multiple neurons in a certain way. The training method of a neural network is gradient backpropagation based on the chain rule.

3.2.3 Recommendation Process Based on the AutoRec Model

The recommendation process based on the AutoRec model is not complicated. Given the rating vector of the input item i is $\boldsymbol{r}^{(i)}$, the output vector of the model $h(\boldsymbol{r}^{(\mathbf{i})};\theta)$ is

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

the prediction of ratings for the item i by all users. Then, $\hat{R}_{\text{ui}}$ represents the rating prediction of user u for item i, as shown in Eq. 3.4.

$$\hat{R}_{\text{ui}} = \left(h\left(\boldsymbol{r}^{(i)}; \hat{\theta}\right)\right)_u \tag{3.4}$$

By traversing the input item vector, the rating predictions of all items from user u can be obtained. Then the recommendation list can be generated based on the rating predictions.

Like the collaborative filtering algorithm introduced in Section 2.2, AutoRec is also divided into item-based AutoRec and user-based AutoRec. The input vector in the formula introduced here is the rating vector of the item, so it can be called I-AutoRec (Item-based AutoRec). If the user's rating vector is used as the input vector, then we will get U-AutoRec (User-based AutoRec). In the process of recommendation list generation, the advantage of U-AutoRec over I-AutoRec is that it only needs to input the user vector of the target user once, and then the user's rating vector for all items can be constructed. That is to say, only one model inference process is needed to obtain the user's recommendation list; the disadvantage is that the sparsity of the user vector may affect the model's effectiveness.

3.2.4 Strengths and Limitations of the AutoRec Model

The AutoRec model uses a single hidden layer autoencoder to generalize user or item ratings, so that the model has a certain level of generalization and expressivity. Because the structure of the AutoRec model is relatively simple, it has a certain problem of insufficient expressivity.

In terms of model structure, the AutoRec model is exactly the same as the later word-to-vector model (Word2vec), but with different optimization targets and training methods. After learning Word2vec, interested readers can compare the similarities and differences between these two models.

From the perspective of deep learning, the proposal of the AutoRec model opened the prelude to the use of deep learning to solve the recommendation problem, and provided ideas for the construction of complex deep learning networks.

3.3 Deep Crossing Model: A Classic Deep Learning Architecture

If the AutoRec model is an initial attempt to apply deep learning to the recommender system, then the Deep Crossing model [2] proposed by Microsoft in 2016 is a complete application of the deep learning architecture in the recommender system. Although companies have claimed that they have applied deep learning models in their recommender systems since 2014, it was not until the year when the Deep Crossing model was released that there were official papers sharing the technical details of the complete deep learning recommender system. Compared with some problems of poor expressivity caused by the simple network structure of the AutoRec model, the Deep Crossing model completely solves a series of deep learning implementation issues from feature

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

engineering, sparse vector densification, and multilayer neural network optimization target fitting. The solutions provided in this model have laid a good foundation for much subsequent research.

3.3.1 Application Scenarios of the Deep Crossing Model

The application scenario of the Deep Crossing model is the search advertisement recommendations in the Microsoft search engine Bing. After a user enters a search term in the search box, the search engine will not only return relevant results but also return advertisements related to the search term, which is also the main profit source of most search engines. Based on the business model, the most important module of an ads system is to build a CTR model to accurately predict click-through rate and further lift performance of ads recommendation. Therefore, CTR naturally become the optimization objective of the Deep Crossing model.

The features used by Microsoft under this use case are shown in Table 3.1. These features can be divided into three categories – the categorical features that can be processed into one-hot or multi-hot vectors, including user search terms (that is, query), ad keyword, ad title, landing page, match type; the numeric features, which Microsoft calls counting features, including CTR and click prediction; the other one is the features that need further processing, including advertising campaign, impression, click, and so on. Strictly speaking, these are not independent features but rather a group of features that need further processing. For example, the budget in the advertising campaign can be used as a numerical feature, and the ID of the advertising plan can be used as a categorical feature.

Categorical features can be processed into feature vectors through one-hot or multi-hot encoding, and numerical features can be directly concatenated into

Table 3.1 Features in the Deep Crossing model

Feature	Feature meaning
Search term	The search term entered by the user in the search box
Ad keyword	Keywords that the advertiser adds to the ad to describe their product
Ad title	The titles of the ads
Landing page	The first page after ad is clicked
Match type	Advertiser-selected ad-search term match type (including exact match, phrase match, semantic match, and so on)
CTR	Ad's historical CTR
Click prediction	CTR prediction from another CTR model
Ad campaign	The ad delivery plan created by the advertiser, including budget, targeting conditions, and so on
Impression Sample	An example of an ad "impression" that records the contextual information about the ad in the actual impression scene
Click Sample	An example of an ad "click" that records the contextual information about the ad in the actual click scenario

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

feature vectors. After generating the vector representation of all input features, the Deep Crossing model uses the feature vectors to predict CTR. The characteristic of a deep learning network is that the network structure can be flexibly adjusted according to the business and engineering needs, so as to achieve the end-to-end training from the original input features to the final optimization target. Next, by analyzing the network structure of the Deep Crossing model, we can explore how deep learning can accurately predict the CTR through the layer-by-layer processing of features.

3.3.2 Network Structure of Deep Crossing Model

In order to achieve end-to-end training, the Deep Crossing model needs to solve the following problems in its network:

(1) How to solve the problem of densification of sparse feature vectors since one-hot encoding feature is too sparse, which is not in favor of direct training;
(2) How to solve the problem of automatic feature crossovers;
(3) How to achieve the optimization target set by the problem in the output layer.

The Deep Crossing model sets up different neural network layers to solve these problems. As shown in Figure 3.6, the network structure mainly includes four layers: the embedding layer, the stacking layer, the multiple residual units layer, and the scoring

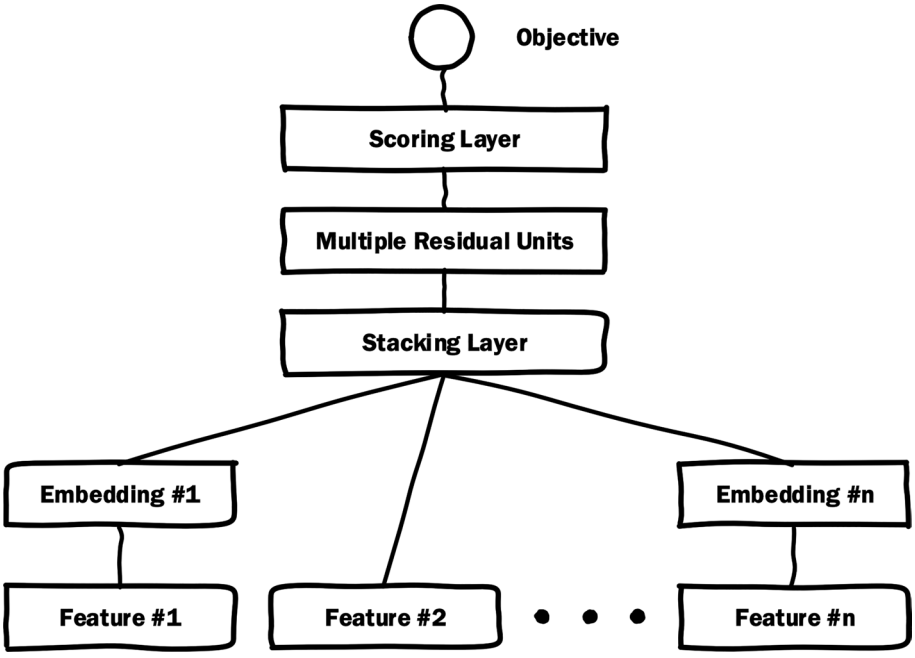

Figure 3.6 Structure diagram of the Deep Crossing model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

layer. Next, the functions and implementations of each layer will be introduced from bottom to top.

- *Embedding layer*: The role of the embedding layer is to convert sparse categorical features into dense embedding vectors. As can be seen from Figure 3.6, each feature (such as Feature#1, here refers to the one-hot encoded sparse feature vector) will be converted into the corresponding embedding vector (such as Embedding#1) after passing through the embedding layer.

  The structure of the embedding layer is mainly based on the classic fully connected layer structure, but the embedding technology itself, as a very widely studied topic in the deep learning domain, has derived many other different embedding methodologies such as Word2vec, Graph Embedding, and so on. Chapter 4 will give a more detailed introduction to the state-of-the-art embedding models.

  Generally speaking, the dimension of the embedding vectors should be much smaller than the original sparse feature vector, and tens to over one hundred dimensions can generally meet the requirements. It should be noted here that Feature#2 in Figure 3.6 actually represents a numerical feature. The numerical feature does not need to go through the embedding layer, but directly enters the stacking layer.

- *Stacking layer*: The function of the stacking layer is relatively simple. It is to concatenate different embedding features and numerical features to form a new feature vector containing all features. This layer is also usually referred as the concatenate layer.
- *Multiple Residual Units layer*: The main structure of this layer is a MLP. Compared with the standard neural network with a perceptron as the basic unit, the Deep Crossing model uses a multilayer residual network as the MLP implementation. The most famous residual network is the 152-layer residual network proposed by Microsoft researcher Yuming He in the ImageNet competition [3]. The application of residual networks in the Deep Crossing model is also the first successful extension of residual networks outside the field of computer vision.

  Through the multilayer residual network, the various dimensions of the feature vector are fully crossed, so the model can capture more information about nonlinear features and combined features. As a result, the deep learning model is more expressive than traditional machine learning models.

Basics: Residual Neural Networks and Its Characteristics

Residual neural network is a neural network composed of residual units. The specific structure of the residual unit is depicted in Figure 3.7.

Different from the traditional perceptron, the residual unit has two main characteristics:

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

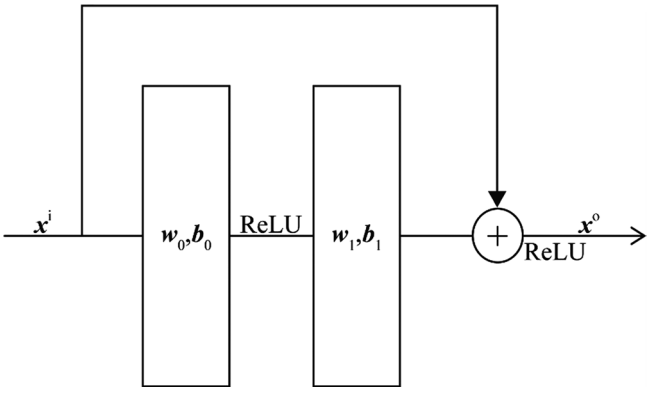

Figure 3.7 The specific structure of a residual unit.

(1) The residual unit contains a fully connected layer with ReLU as the activation function.
(2) The input is directly connected with ReLU output through a shortcut path.

Under such a structure, what the residual unit is actually fitting is the "residual difference" ($\boldsymbol{x}^0 - \boldsymbol{x}^1$) between the output and the input, which is the origin of the name of the residual neural network.

The birth of the residual neural network is mainly to solve two problems:

(1) For traditional perceptron-based neural networks, when the network is deepened, there is often an overfitting problem; that is, the deeper the network, the worse the performance on the test set. In the residual neural network, due to the existence of short-circuit of the input vector, the two-layer ReLU network can be skipped in many cases to reduce the occurrence of overfitting.
(2) When the neural network is deep enough, there is often a serious gradient vanishing phenomenon. The vanishing gradient phenomenon means that in the process of gradient backpropagation, the closer to the input end, the smaller the magnitude of the gradient, and then the slower the parameter convergence speed. To solve this problem, the residual unit uses the ReLU activation function to replace the original sigmoid activation function. In addition, short-circuiting the input vector is equivalent to directly passing the gradient to the next layer without modification, which also makes the residual network converge faster.

- *Scoring layer*: The scoring layer, as the output layer, is to fit the optimization objective. For binary classification problems such as CTR prediction, the scoring layer often uses a logistic regression model, while for multiclassification problems such as image classification, the scoring layer often uses a softmax model.

This is the model structure of Deep Crossing. On this basis, the gradient backpropagation method is used for training, and finally the CTR prediction model based on Deep Crossing is obtained.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.3.3 The Revolution to Feature Crossing Method by Deep Crossing Model

From the view of the current deep learning world, the Deep Crossing model is unremarkable, because it does not introduce any special model structure such as attention mechanism, sequence model, and so on. It just uses the typical deep learning architecture with embeddings and a multilayer neural network. But from a historical perspective, the emergence of the Deep Crossing model is revolutionary. There is no manual feature engineering involved in the Deep Crossing model. The original features are fed into the neural network layer after the embedding layer, and the task of feature crossing is all handed over to the model. Compared with the previously introduced FM and FFM models, which only have the ability to cross second-order features, the Deep Crossing model can perform "deep crossover" among features by adjusting the depth of the neural network, which is the origin of the name Deep Crossing.

3.4 NeuralCF Model: Combination of CF and Deep Learning

In Section 2.2, we introduce the classic algorithm of a recommender system – collaborative filtering. The Matrix Decomposition technique is then developed along the idea of collaborative filtering (Section 2.3), which decomposes the co-occurrence matrix in collaborative filtering into the user vector matrix and item vector matrix. In this model, the inner product of the hidden vector of user u and the hidden vector of item i is the prediction of the rating of item i by user u. Following the development path of Matrix Decomposition and combining with deep learning knowledge, researchers from the National University of Singapore proposed a deep-learning-based collaborative filtering model NeuralCF [4] in 2017.

3.4.1 Revisiting Matrix Factorization Models from the Perspective of Deep Learning

As mentioned in the introduction to the Deep Crossing model in Section 3.3, the main function of the embedding layer is to convert sparse vectors into dense vectors. In fact, if we view the Matrix Decomposition model from the perspective of deep learning, the user-hidden vector and item-hidden vector of the matrix decomposition layer can be treated as one kind of embedding method. The final "scoring layer" is to obtain the "similarity" after the inner product of the user's latent vector and the item's latent vector. The "similarity" here is the prediction of the rating. In summary, the architecture of the matrix factorization model can be described by a deep learning network like structure, as shown in Figure 3.8.

In the process of training and evaluating models using Matrix Decomposition, it is often found that the model is prone to underfitting. The reason is that the model structure of matrix decomposition is relatively simple, especially the output layer (that is, the scoring layer), which cannot effectively fit the optimization objective.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

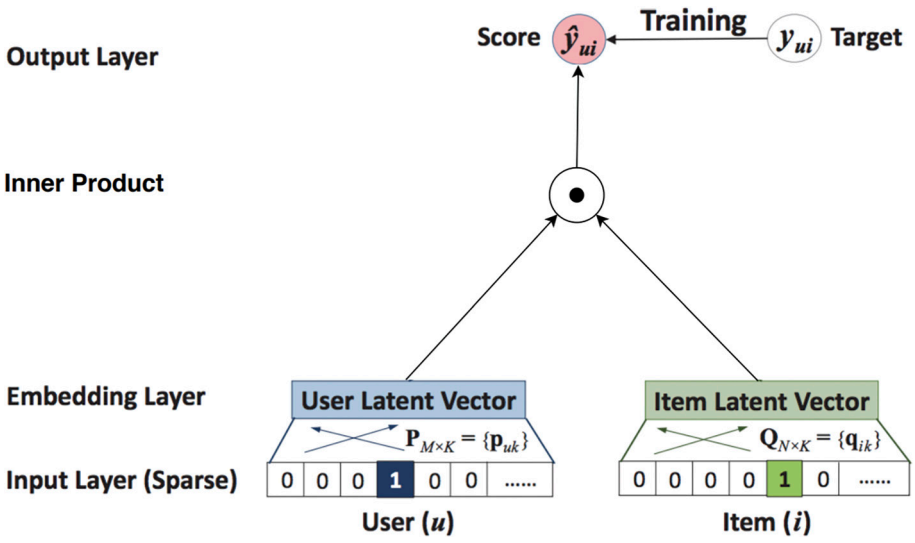

Figure 3.8 Representation of matrix factorization in a deep learning network-like structure.

This requires the model to have stronger expressivity. Inspired by this motivation, researchers from the National University of Singapore proposed the NeuralCF model.

3.4.2 Network Structure of the NeuralCF Model

As shown in Figure 3.9, NeuralCF replaces the simple inner product operation in the matrix factorization model with the structure of the multilayer neural network and output layer. The benefits of doing so are intuitive. First, the user vector and the item vector can be more effectively crossed to obtain more valuable feature combination information; the second is to introduce more nonlinear features to make the model more expressive.

In fact, the interaction layer of user and item vectors can be replaced by any other form of manipulations. Such type of model is the so-called Generalized Matrix Factorization model.

The original matrix decomposition uses the "inner product" method to allow the user to interact with the item vector. In order to further allow the vectors to fully cross in each dimension, the element-wise product (that is, multiplying the corresponding elements from two vectors with the same dimension) is used for interoperability. Then the final prediction target is fitted through the output layer, such as logistic regression. The use of neural networks to fit interaction functions in NeuralCF is a generalized form of feature crossing. In the chapters that introduce the PNN model and the Deep and Cross model, more feasible forms of interaction functions will be introduced.

Further, the feature vectors obtained through different interaction networks can be concatenated and passed to the output layer for fitting. An example of integrating two

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

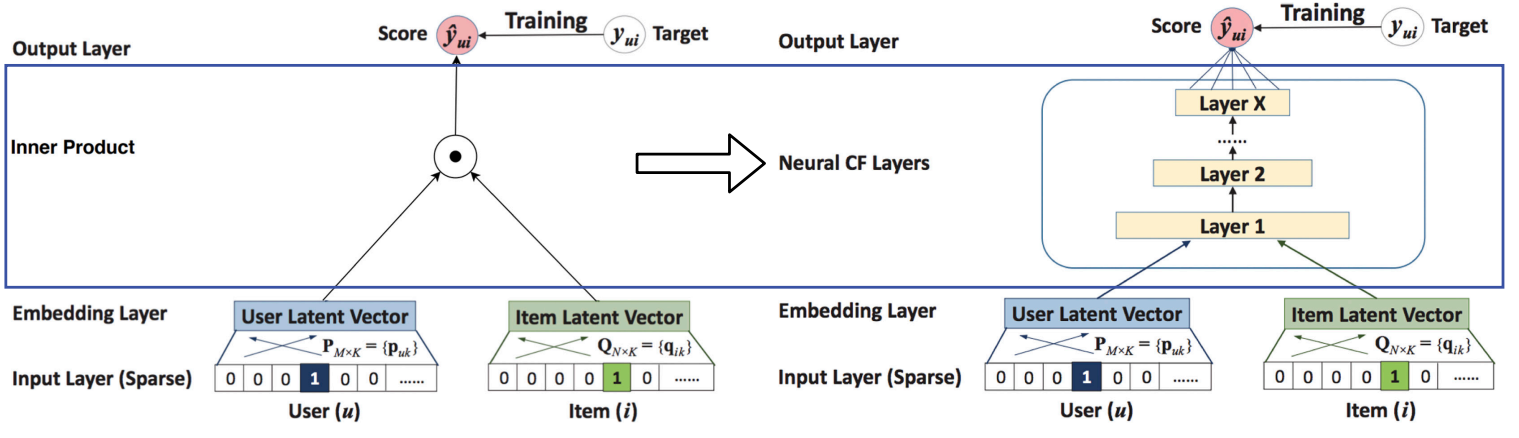

Figure 3.9 From traditional matrix factorization to NeuralCF.

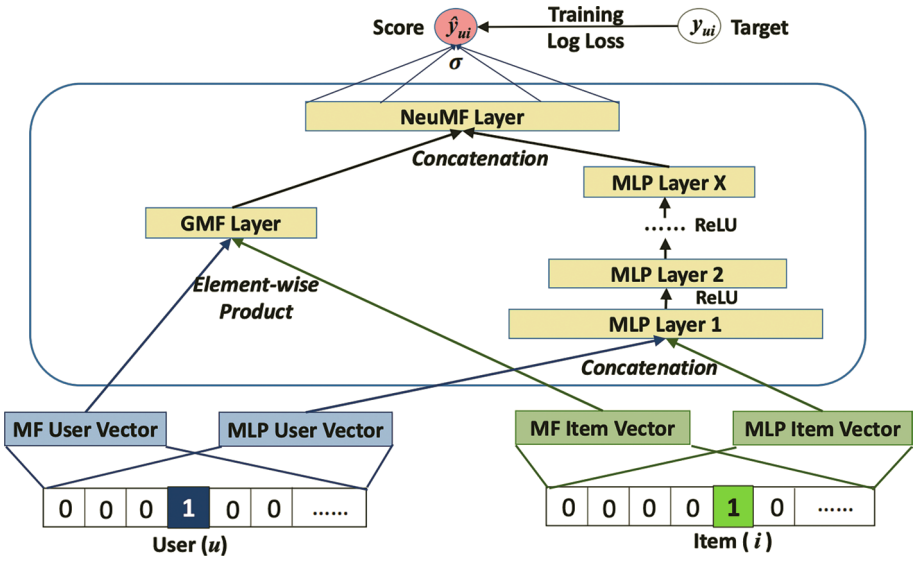

Figure 3.10 The hybrid NeuralCF model.

networks is given in the NeuralCF paper [4] (shown in Figure 3.10). Such a model is called the hybrid NeuralCF. It can be seen that the hybrid NeuralCF model integrates with the original NeuralCF model mentioned earlier and the generalized matrix factorization model with element-wise results, such as interoperability. This allows the model to have stronger feature crossing and nonlinearity.

Basics: What Is the Softmax Function?

While introducing the Deep Crossing and NeuralCF models, it has been mentioned many times that the softmax function is used as the final output layer of the model to solve the fitting of multiclassification problems. So what is the softmax function and why is the softmax function able to solve multiclassification problems?

Mathematical Definition of Softmax Function

Given an n-dimensional vector, the softmax function maps it to a probability distribution. The standard softmax function $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ is defined by the following formula,

$$\sigma(\boldsymbol{X})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \quad \text{where } i = 1, \cdots, n \text{ and } \boldsymbol{X} = [x_1, \cdots, x_n]^T \in \mathbb{R}^n$$

It can be seen that the softmax function solves the problem of mapping from an original n-dimensional vector to an n-dimensional probability distribution. Then in the multiclass classification problem, assuming that the number of classes is n, what the model wants to predict is the probability distribution of a sample on n classes. If a

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

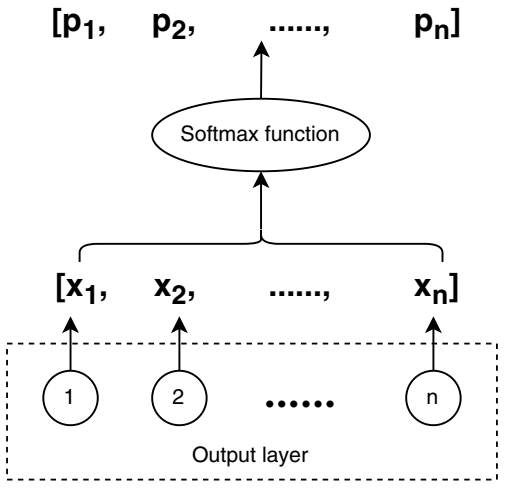

Figure 3.11 The structure of the softmax output layer.

deep learning model is used for prediction, then the final output layer is composed of n neurons. The output of the n neurons then become the input (a n-dimensional vector) to the final softmax function. Eventually, the final multiclass probability distribution can be obtained from the output of the softmax function. In a neural network, the structure of the softmax output layer can be presented as shown in Figure 3.11.

In multiclass classification problems, the softmax function is often used together with the cross-entropy loss function,

$$\text{Loss}_{\text{Cross Entropy}} = -\sum_i y_i \ln\left(\sigma(\boldsymbol{x})_i\right)$$

where y_i is the ground truth label value of the ith category, and $\sigma(\boldsymbol{x})_i$ represents the predicted value of the ith category by the softmax function. Because the softmax function normalizes the classification output into the probability distribution of multiple classifications, and the cross entropy describes the similarity between the predicted classification and the actual result, the softmax function is often used in conjunction with the cross entropy. When using cross-entropy as the loss function, the gradient descent form of the entire output layer becomes extremely simple. The derivative of the softmax function turns,

$$\frac{\partial \sigma(\boldsymbol{x})_i}{\partial x_j} = \begin{cases} \sigma(\boldsymbol{x})_i \left(1 - \sigma(\boldsymbol{x})_j\right), & i = j \\ -\sigma(\boldsymbol{x})_i \cdot \sigma(\boldsymbol{x})_j, & i \neq j \end{cases}$$

Based on the chain rule, the derivative of the cross-entropy function to the jth-dimensional input x_j of the softmax function can be expressed as,

$$\frac{\partial \text{Loss}}{\partial x_j} = \frac{\partial \text{Loss}}{\partial \sigma(\boldsymbol{x})} \cdot \frac{\partial \sigma(\boldsymbol{x})}{\partial x_j}$$

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

In a multiclass classification problem, only one dimension of 1 is in the ground truth label, and the rest of the dimensions are all 0. Assuming that the kth dimension is 1, that is, $y_k = 1$, then the cross-entropy loss function can be simplified into the following form,

$$\text{Loss}_{\text{Cross Entropy}} = -\sum_i y_i \ln\left(\sigma(\boldsymbol{x})_i\right) = -y_k \cdot \ln\left(\sigma(\boldsymbol{x})_k\right) = -\ln\left(\sigma(\boldsymbol{x})_k\right)$$

Then,

$$\frac{\partial \text{Loss}}{\partial x_j} = \frac{\partial\left(-\ln\left(\sigma(\boldsymbol{x})_k\right)\right)}{\partial \sigma(\boldsymbol{x})_k} \cdot \frac{\partial \sigma(\boldsymbol{x})_k}{\partial x_j} = -\frac{1}{\sigma(\boldsymbol{x})_k} \cdot \frac{\partial \sigma(\boldsymbol{x})_k}{\partial x_j} = \begin{cases} \sigma(\boldsymbol{x})_j - 1, j = k \\ \sigma(\boldsymbol{x})_j, j \neq k \end{cases}$$

From this, it can be seen that the combination of softmax function and cross entropy is not only perfectly aligned in mathematical meaning, but also makes the gradient formula concise. Based on this gradient equation, the update of the weight of the entire neural network can be completed by the method of gradient backpropagation.

3.4.3 Strengths and Limitations of NeuralCF Models

The NeuralCF model actually proposes a model framework – it is based on the two embedding layers of the user vector and the item vector, uses different interaction layers to cross the features, and can flexibly concatenate different interaction layers. From this, we can see the advantages of deep learning in building a recommendation model – using the ability of neural networks to fit arbitrary functions in theory, flexibly combining different features, and increasing/decreasing the complexity of the model as needed.

In practice, it should be noted that it is not always true that the more complex the model structure and the more features, the better. We need to understand the consequence induced by adding more complexities to the model: (1) risk of overfitting; (2) demand of a larger amount of training data; and (3) longer training time. These aforementioned aspects are what algorithm engineers need to consider while making trade-off decisions between model practicability, real-time performance, and effectiveness.

The NeuralCF model also has its own limitations. Since it is developed on the basis of collaborative filtering, the NeuralCF model does not introduce other types of features, which undoubtedly wastes other valuable information in practical applications. In addition, there is no further exploration and categorization of feature interaction types in the model. It requires deeper dives in the follow-up research.

3.5 PNN Model: A Way of Enhancing Feature Cross Capabilities

The main idea of the NeuralCF model introduced in Section 3.4 is to use a multilayer neural network to replace the dot product operation of classical collaborative filtering to enhance the expressiveness of the model. In a broader sense, any manipulation

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

method between vectors can be used to replace the inner product operation of collaborative filtering, and the corresponding model can be called a generalized matrix factorization model. However, the NeuralCF model only mentions two fields of feature vectors, the user vector and the item vector. How to design the feature crossing method if multiple sets of feature vectors are added? In 2016, the PNN (Product-based Neural Networks) model proposed by researchers from Shanghai Jiao Tong University [5] gave several design ideas for feature interaction.

3.5.1 Network Structure of the PNN Model

The purpose of the PNN model proposal is also to solve the problem of CTR prediction in the recommender system, so the application scenarios of the model will be omitted here. Figure 3.12 shows the model structure diagram. Compared with the Deep Crossing model (as shown in Figure 3.6), the PNN model is similar in most parts of the overall structure, including the input layer, embedding layer, MLP layer, and final output layer. The only difference is that the PNN model replaces the stacking layer in the Deep Crossing model with a product layer. In other words, the embedding vectors of different features are no longer simply concatenated; instead, a product operation is applied to each pair of embedding vectors to capture cross-feature information in a more structured manner.

In addition, compared with NeuralCF, the input of the PNN model not only includes user and item information but can also have more features in different forms

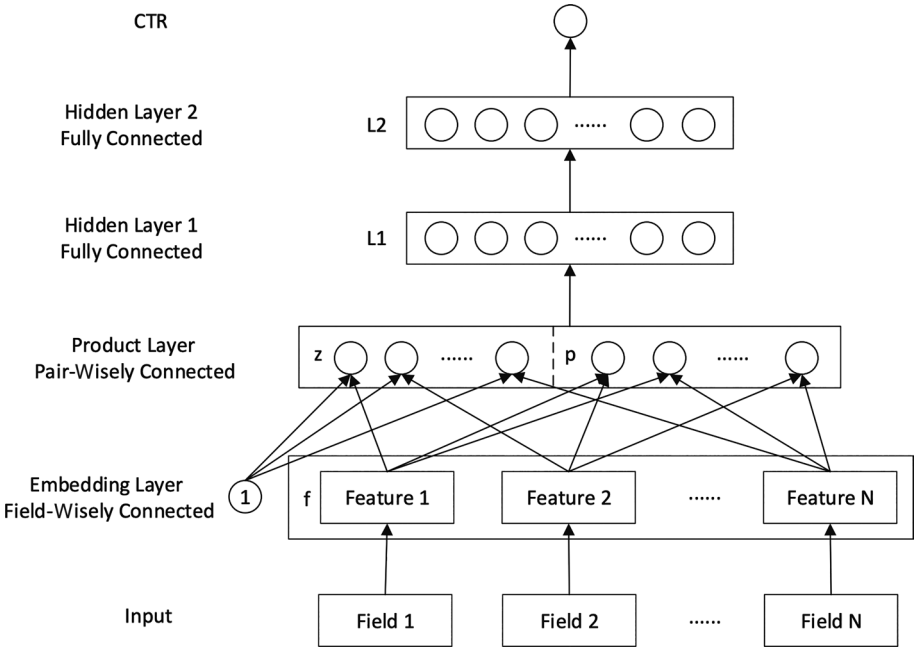

Figure 3.12 Structure diagram of the PNN model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

and sources, followed by generating the dense embedding feature vectors of the same length through the encoding of the embedding layer. To model feature crossing, the PNN model also provides more specific interaction methods.

3.5.2 Multiple Feature Intersection Forms in the Product Layer

The main innovation of the PNN model for the deep learning structure is the introduction of the product layer. Specifically, the product layer of the PNN model consists of a linear operation part (block z of the product layer in Figure 3.12) and a product operation part (block p of the product layer in Figure 3.12). Among them, the product feature interaction part can be divided into inner product type and outer product type. The PNN model using inner product operation is called Inner Product-based Neural Network (IPNN), and the PNN model using outer product operation is called Outer Product-based Neural Network (OPNN).

Whether it is an inner product type or an outer product type, it is a form of pairwise combination of different feature embedding vectors. In order to ensure the smooth operation of the product, the dimensions of each embedding vector must be the same.

The inner product is a classic vector manipulation method. Assuming that the input feature vectors are $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$ respectively, the inner product equation $g_{\text{inner}}(\boldsymbol{f}_i, \boldsymbol{f}_j)$ can be defined as,

$$g_{\text{inner}}\left(\boldsymbol{f}_i, \boldsymbol{f}_j\right) = \left\langle \boldsymbol{f}_i, \boldsymbol{f}_j \right\rangle \tag{3.5}$$

The outer product operation is to cross each dimension of the input feature vectors $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$ for each pair of elements to generate a feature cross matrix. The outer product equation $g_{\text{outer}}(\boldsymbol{f}_i, \boldsymbol{f}_j)$ can be defined as,

$$g_{\text{outer}}\left(\boldsymbol{f}_i, \boldsymbol{f}_j\right) = \boldsymbol{f}_i \boldsymbol{f}_j^{\mathrm{T}} \tag{3.6}$$

The outer product operation generates a square matrix with the dimension of $M \times M$, where M is the dimension of the input vector. It is clear that such an operation will directly increase the complexity of the algorithm from the order of M originally to M^2. In order to reduce the burden of model training, a dimensionality reduction method was introduced in the PNN model paper. The results of the outer product of the feature embedding vectors are super-positioned to form a combined outer product matrix $\boldsymbol{p}$, as shown,

$$\boldsymbol{p} = \sum_{i=1}^{N}\sum_{j=1}^{N} g_{\text{outer}}\left(\boldsymbol{f}_i, \boldsymbol{f}_j\right) = \sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{f}_i \boldsymbol{f}_j^{\mathrm{T}} = \boldsymbol{f}_{\Sigma} \boldsymbol{f}_{\Sigma}^{\mathrm{T}}, \boldsymbol{f}_{\Sigma} = \sum_{i=1}^{N} \boldsymbol{f}_i \tag{3.7}$$

From the final form of Equation 3.7, the final superposition matrix $\boldsymbol{p}$ is similar to applying an average pooling on all the feature embeddings and then performing the outer product operation.

In practical applications, the operation of average pooling should also be treated with caution. Because the corresponding dimensions of different features are averaged, it is actually assumed that the corresponding dimensions of different features have similar physical meanings. But obviously, if one feature is "age" and the other

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

is "region," then after these two features have passed through their respective embedding layers, the embedding vectors of the two are not in the same vector space, which is obviously not comparable. At this time, averaging the two will obscure a lot of valuable information. The average pooling often occurs in the embeddings in the same domain; for example, the embedding of multiple items browsed by the user is averaged. Therefore, the outer pooling operation of the PNN model needs to be cautious, and carefully balanced between training efficiency and model performance.

In fact, after the linear and product operations of the features, the PNN model does not directly send the results to the upper L_1 fully connected layer (as shown in Figure 3.12), but performs a local fully connected layer conversion inside the product layer. It maps the linear portion z, the product portion p into D_1-dimensional input vectors $\boldsymbol{l}_z$ and $\boldsymbol{l}_p$ respectively. D_1 is the number of hidden units in the L_1 hidden layer. The mapped vectors $\boldsymbol{l}_z$ and $\boldsymbol{l}_p$ are superimposed and passed into the hidden layer. This part of the operation is commonly seen and can be replaced by other types of transformation operations, so it will not be described in detail here.

3.5.3 Strengths and Limitations of the PNN Model

The highlight of the PNN model is that it emphasizes the versatility of interaction methods between feature embedding vectors. Compared with the simple, undifferentiated processing in the fully connected layer, the inner product and outer product operations adopted by the PNN model obviously focus more on the interaction between different features, which makes it easier for the model to capture the interacting relationship of the features.

However, the PNN model also has some limitations. For example, in the practical application of the outer product operation, a lot of simplification operations have to be performed to optimize the training efficiency. Furthermore, performing an indiscriminate crossover of all features, to some extent, ignores the valuable information contained in the original feature vector. It then comes down to questions such as how to integrate original features and crossed features to make feature crossing more efficient. The Wide&Deep model and various deep learning models based on FM introduced in the later sections will give their solutions.

3.6 Wide&Deep Model: Combining Memorization and Generalization

This section introduces a model that has had great influence in the industry since it was proposed: the Wide&Deep model, presented by Google in 2016 [6]. The main idea of the Wide&Deep model, as its name suggests, is a hybrid model consisting of a single-layer "wide" substructure and a multilayered "deep" substructure. Among them, the main function of the wide part is to make the model have strong memorization ability, while the main responsibility of the deep part is to make the model have more generalization ability. It has the advantages of logistic regression as well

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

as deep neural network. That is, it can quickly process and memorize a large number of historical behavioral characteristics, and also has strong expressivity. It not only quickly became the state-of-art model in the industry at that time, but also derived a large number of hybrid models based on the foundations of the Wide&Deep model. Its influence still continues today.

3.6.1 Memorization and Generalization of the Wide&Deep Model

The original intention of the Wide&Deep model and its greatest value are from strong memorization ability and generalization ability at the same time. This is the first time we have mentioned the Memorization concept in this book. Although generalization has been mentioned many times in previous chapters, it has never given a detailed explanation. In this section, we will give a detailed explanation of both these two concepts.

Memorization can be understood as the ability of the model to directly learn and utilize the "co-occurrence frequency" of items or features in the historical data. Generally speaking, simple models such as collaborative filtering and logistic regression have strong "memorization capabilities." Due to the simple structure of this type of model, the original data can often directly affect the recommendation results, resulting in inductive recommendations like "if you have clicked on A, recommend B." This is equivalent to the model directly remembering the distribution of historical data characteristics, and use these memories to make recommendations.

Since the Wide&Deep model was originally proposed by the Google Play recommendation team, here we take the scenario of app recommendation as an example to explain the model's memorization capability.

Suppose that the following combined features are adopted during the training process of the Google Play recommendation model, $\text{AND}\left(\text{user}_{\text{installed\_app}} = \text{netflix}; \text{impression}_{\text{app}} = \text{pandora}\right)$, or (netflix & pandora). This feature means that the user has installed the Netflix app and sees the Pandora app recommended in the Google Play App store. If we use a successful Pandora installation as a positive label, it is easy to count the co-occurrence frequency between the feature of (Netflix and Pandora) and the positive labels of Pandora installation. Assuming that the co-occurrence frequency of the two is as high as 10% (the global average application installation rate is 1%), this feature is so strong that when designing a model, we expect that the model will recommend Pandora as soon as it finds this feature. It is like a memorable point imprinted in people's minds. This is the so-called memorization of the model. For simple models such as logistic regression, if such a "strong feature" is found, its corresponding weight will be greatly adjusted during the model training process, thus reflecting the direct memory of this feature. On the other hand, for a multilayer neural network, the feature will be processed through multiple layers and continuously crossed with other features, so the model's memory of any strong feature is not as prominent as that of a simple model.

The generalization ability can be understood as the relevancy in the feature transfer, and the ability to discover the latent correlation between the features and ground truth label, especially when the features are sparse or never appeared. Matrix

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

decomposition has a stronger generalization ability than collaborative filtering, since matrix decomposition introduces a structure such as a hidden vector, which lets users or items with sparse data generate hidden vectors to obtain data-driven recommendation scores. This is a typical example of passing global data to the sparse items to improve generalization. For another example, deep neural networks can deeply explore latent patterns in data through multiple automatic crossings of the features. Even with very sparse input feature vectors, we can still obtain a relatively stable and smooth recommendation probability through a deep neural network structure. This is the generalization ability that simple models lack.

3.6.2 Network Structure of the Wide&Deep Model

Given the strong memorization ability of the simple model and the strong generalization ability of the deep neural network, the direct motivation for designing the Wide&Deep model is to combine these two structures. The specific model structure is shown in Figure 3.13.

The Wide&Deep model combines a wide part with a single input layer, and a deep part, which consists of the embedding layer and multiple hidden layers. Then both parts are fed to the final output layer to generate the prediction. The single-layer side (wide side) is good at dealing with a large number of sparse ID features while the deep side uses the strong expressive ability of the neural network to perform deep feature crossing and mine the data patterns hidden behind the features. Finally, using the logistic regression model, the output layer combines the outputs from the wide part and the deep part to generate the final prediction.

The specific feature engineering and input layer design present a deep understanding of the business use cases from the Google Play Recommendation team. From Figure 3.14, we can learn in detail which features the Wide&Deep model uses as the input of the deep part and which features are used as the input of the wide part.

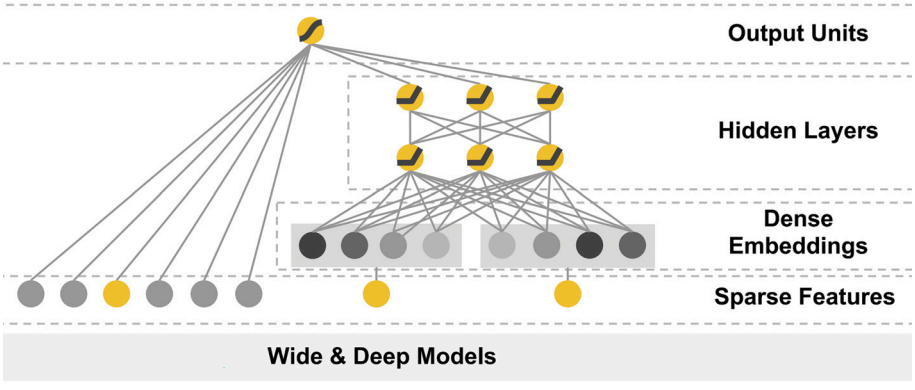

Figure 3.13 The structure of the Wide&Deep model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

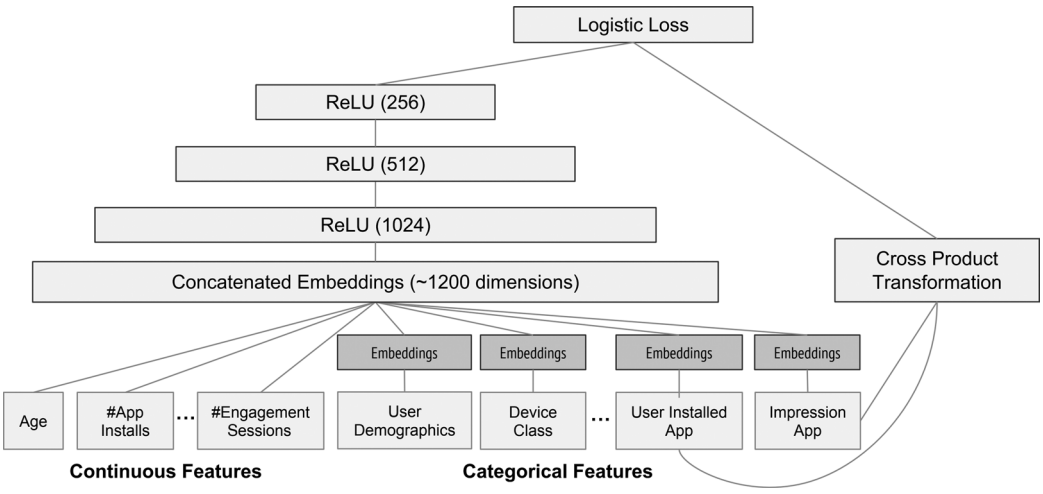

Figure 3.14 The structure of the Wide&Deep model with more feature details.

The input of the deep part is the full set of feature vectors, including user age, number of installed applications, device type, installed applications, impression applications, and so on. The category features, such as installed applications and impression applications, need to go through the embedding layer before entering the connection layer, where embeddings are concatenated into a 1200-dimensional vector. Then this vector is passed through three layers of ReLU fully connected layers, and finally fed to the output layer with the log-loss function.

The input of the wide part only includes two types of features – installed applications and impressed applications, where the installed applications represent the user's historical behavior, and the impressed applications represent the current application candidate to be recommended. The reason for choosing these two types of features is to take full advantage of the memorization ability of the wide part. As mentioned in the memorization example in Section 3.6.1, simple models are good at memorizing information in user behavior characteristics, and can directly influence recommendation results.

The function of combining the feature installed application and impressed application in the wide part is called the *Cross Product Transformation* function, and its definition is shown as follows,

$$\varnothing_{\kappa}\left(X\right)=\prod_{i=1}^{d}x_i^{c_{ki}} \qquad c_{ki}\in\left\{0,1\right\} \tag{3.8}$$

where c_{ki} is a Boolean variable and x_i is the ith feature. When the ith feature belongs to the kth crossed feature, the value of c_{ki} is 1, otherwise it is 0. For example, for the crossed feature AND(user$_{\text{installed_app}}$ = netflix; impression$_{\text{app}}$ = pandora), the corresponding cross-product transform function output is 1 only when both two individual features user$_{\text{installed_app}}$ = netflix and impression$_{\text{app}}$ = pandora are positive, otherwise it is 0.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

After the features are crossed through the cross-product transformation layer operation, the wide part feeds the combined features into the final log-loss output layer, and participates in the final objective fitting together with the output from the deep part.

3.6.3 Evolution of the Wide&Deep Model: The Deep and Cross Model

The development of the Wide&Deep model not only integrates memorization and generalization, but also opens up a new idea for the integration of different network structures. After the Wide&Deep model, more and more works focus on improving the Wide&Deep parts, respectively. A representative model is the Deep and Cross model (DCN) proposed by researchers from Stanford University and Google in 2017 [7].

The structure diagram of the Deep and Cross model is shown in Figure 3.15. The main idea is to use the cross network to replace the original wide part. Since the design idea of the deep part has not changed substantially, this section focuses on the design idea and specific implementation of the cross part.

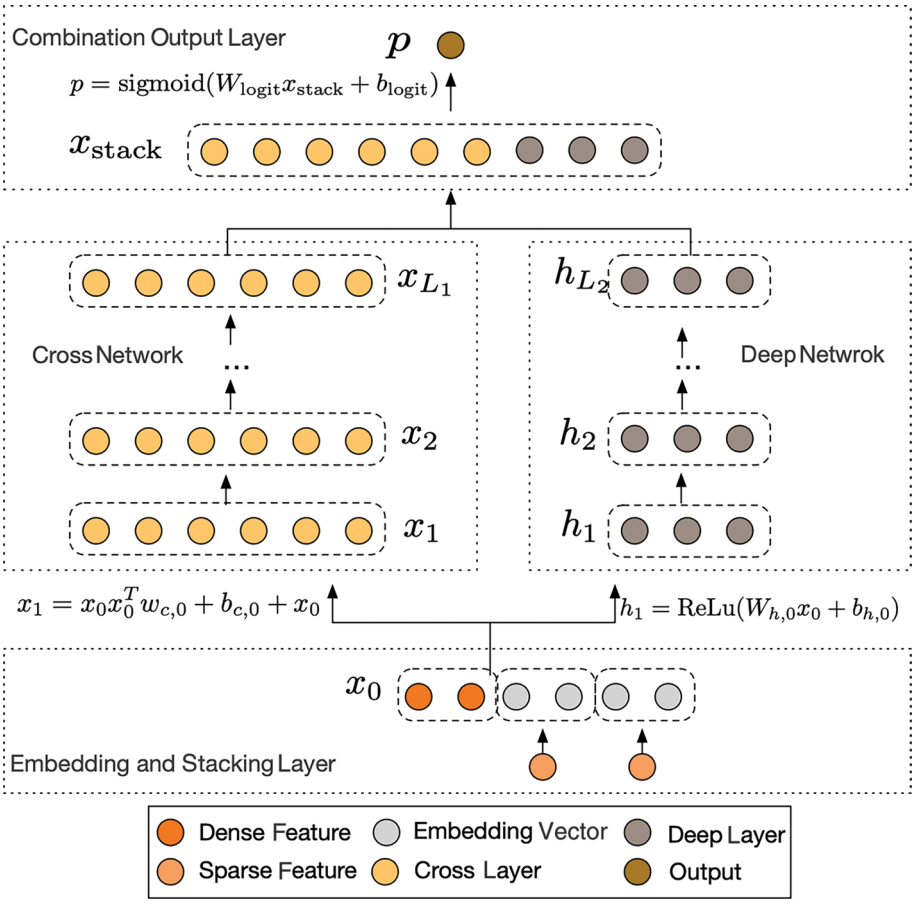

Figure 3.15 The structure of the Deep and Cross model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

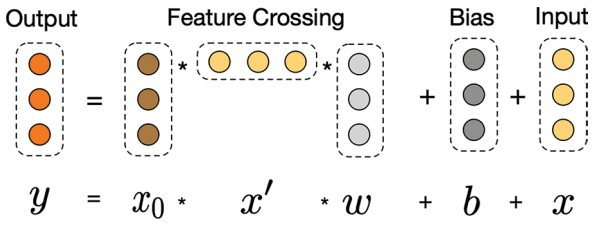

Figure 3.16 The operation of the cross-layer.

The purpose of designing the cross network is to increase the interaction strength between the features, and use a multilayer cross layer to perform feature crossover on the input vector. Assuming that the output vector of the lth cross layer is $\boldsymbol{x}_l$, then the output vector of the $(l+1)$th layer can be expressed as,

$$\boldsymbol{x}_{l+1} = \boldsymbol{x}_0 \boldsymbol{x}_l^{\mathrm{T}} \boldsymbol{W}_l + \boldsymbol{b}_l + \boldsymbol{x}_l \tag{3.9}$$

It can be seen that the second-order term of the cross-layer operation is very similar to the outer product operation mentioned in the PNN model in Section 3.5. On this basis, the weight vector $\boldsymbol{w}_l$ of the outer product operation, as well as the original input vector $\boldsymbol{x}_l$ and bias vector $\boldsymbol{b}_l$ are added. The operation of the cross layer is shown in Figure 3.16.

It can be seen that the cross layer is relatively "conservative" in increasing parameters. Each layer only adds an n dimensional weight vector $\boldsymbol{w}_l$, where n is the input vector dimension, and the input vector is retained in each layer. So the change between output and input will not be particularly noticeable. The cross network composed of multiple interaction layers performs automatic feature crossover, which is more advanced than the wide part in the Wide&Deep model. This can help reduce the efforts on feature crossing based on human business understanding. Similar to that in the Wide&Deep model, the deep part of the Deep and Cross model is more expressive than the cross part, which gives the model a stronger learning ability on nonlinear relationships.

3.6.4 Influence of the Wide&Deep Model

The influence of the Wide&Deep model is undoubtedly significant. Not only has it been successfully applied to many first-tier IT companies, but its subsequent improvement and innovation work has continued to this day. In fact, DeepFM, NFM, and other models can be viewed as extensions of the Wide&Deep model. The key to the success of the Wide&Deep model is that:

(1) It grasps the essential characteristics of business problems, and can integrate the advantages of memorization ability from the traditional models and generalization ability from the deep learning models;
(2) The structure of the model is not complicated, and it is relatively easy to implement, train and productionize, which accelerates its popularization and application in the industry.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

It is also from the Wide&Deep model that more and more model structures are added to the recommendation model, and the structure of the deep learning model begins to develop in a diversified and complex direction.

3.7 Integration of FM and Deep Learning Models

The evolution of the FM model family has been presented in detail in Section 2.5. After entering the era of deep learning, the evolution of FM has never stopped. The FNN, DeepFM, and NFM models introduced in this section use different methods to apply or improve the FM model, and integrate them into the deep learning model, continuing the advantages in an easy feature combination.

3.7.1 FNN: Embedding Layer Initialization with the Hidden Vector of FM

The FNN model was proposed by researchers at University College London in 2016 [8]. The structure of this model (as shown in Figure 3.17) is a classic deep neural network similar to the Deep and Cross model. It also includes a typical embedding layer to map the sparse input vector to dense vector. So how exactly is the FNN model combined with the FM model?

The key to the problem is the improvement of the embedding layer. In the parameter initialization process of the neural network, random initialization is often used,

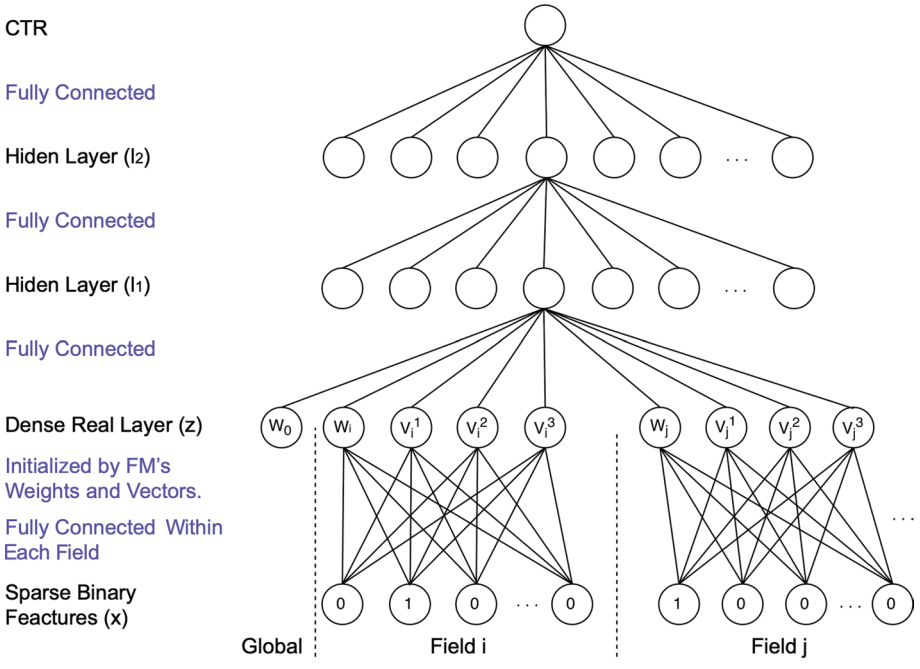

Figure 3.17 Structure of the FNN model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

which does not contain any prior information. Because the input of the embedding layer is extremely sparse, the convergence rate of the embedding layer is very slow. In addition, the number of parameters of the embedding layer often accounts for more than half of the parameters of the entire neural network, so the convergence speed of the entire model is often limited by the embedding layer.

Basics: Why the Convergence Rate of the Embedding Layer Tends to Be Slow

In a deep learning network, the role of the embedding layer is to convert the sparse input vector into a dense vector, but the existence of the embedding layer often slows down the convergence speed of the entire neural network for the following two reasons:

(1) The number of parameters in the embedding layer is huge. A simple calculation can be done here. Assuming that the dimension of the input layer is 100 000, the output dimension of the embedding layer is 32. There are five layers of 32-dimensional fully connected layers added above the embedding layer, and the final output layer dimension is 10. Then, the number of parameters from the input layer to the embedding layer is $32 \times 100\,000 = 3\,200\,000$. The total number of parameters for all remaining layers is $(32 \times 32) \times 4 + 32 \times 10 = 4416$. As a result, the total weight of the embedding layer is $3\,200\,000/(3\,200\,000 + 4416) = 99.86\%$. That is to say, the weight of the embedding layer accounts for the vast majority of the weight of the entire network. It is not hard to understand that most of the training time and computational overhead are attributed to the embedding layer.

(2) Since the input vector is too sparse, in the process of stochastic gradient descent, only the weight of the embedding layer connected to the nonzero feature will be updated (please refer to the parameter update formula in the stochastic gradient descent for understanding), which further reduces the embedding layer convergence speed.

Aiming at the problem of the convergence speed of the embedding layer, the solution of the FNN model is to initialize the parameters of the Embedding layer with each feature latent vector trained by the FM model, which is equivalent to introducing valuable prior information when initializing the neural network parameters. That is to say, the starting point of neural network training is closer to the target optimal point, which naturally accelerates the convergence process of the entire neural network.

Let's review the mathematical form of FM again, as shown in (Eq. 3.10).

$$\boldsymbol{y}_{\mathrm{FM}}(x) := \mathrm{sigmoid}\left(w_0 + \sum_{i=1}^{N} w_i x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \left\langle \boldsymbol{v}_i, \boldsymbol{v}_j \right\rangle x_i x_j \right) \tag{3.10}$$

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

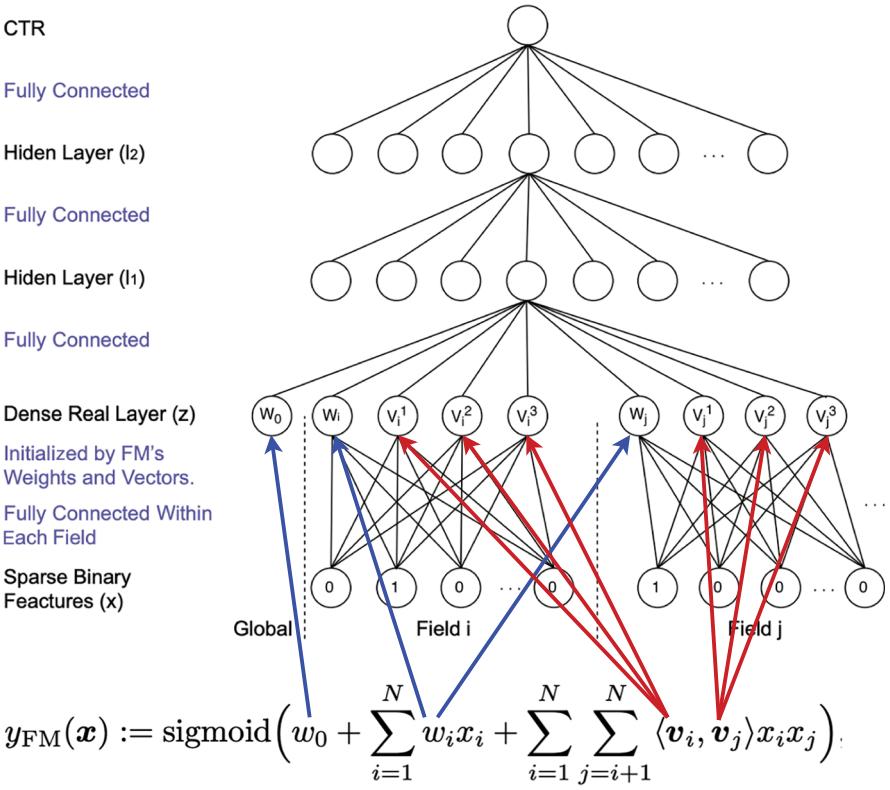

Figure 3.18 The process of using FM to initialize the embedding layer.

The parameters mainly include constant bias w_0, the first-order term parameter w_i and second-order hidden vector $\boldsymbol{v}_i$. The corresponding relationship between the parameters of FM and the parameters of the embedding layer in FNN is depicted in Figure 3.18.

It should be noted that although the parameters in FM are pointed to each neuron in the embedding layer in Figure 3.18, its specific meaning is to the connection weight between the embedding neuron and the input neuron. Assuming that the dimension of the FM hidden vector is m, the hidden vector of the k-th dimension feature of the i-th feature field is $\boldsymbol{v}_{i,k} = (v_{i,k}^1, v_{i,k}^2, \ldots, v_{i,k}^l, \ldots, v_{i,k}^m)$, then the l-th dimension $v_{i,k}^l$ of the hidden vector will become the initial value of the connection weight between the input neuron k and embedding neuron l.

In the process of the FM model training, the feature fields are not distinguished. However, in the FNN model, the features are divided into different feature fields. Each feature field has a corresponding embedding layer, and the dimension of embedding in each feature field should be consistent with the dimension of the FM hidden vector.

In addition to using FM parameters to initialize the weights of the embedding layer, the FNN model also introduces another processing method for the embedding layer in the real application – pre-training. More details are introduced in Chapter 4.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.7.2 DeepFM: Replacing the Wide Part with FM

FNN uses the training result of FM as the initialization weight, and does not adjust the structure of the neural network, while DeepFM [9] jointly proposed by Harbin Institute of Technology and Huawei in 2017 integrates the model structure of FM with the Wide&Deep model. Its model structure diagram is shown in Figure 3.19.

As mentioned in Section 3.6, after the Wide&Deep model, many other models follow the structure of the dual-model combination, and DeepFM is one of them. The improvement of DeepFM on top of the Wide&Deep model is that it replaces the original wide part with FM, which strengthens the ability of partial feature combination of the shallow network. As shown in Figure 3.19, the FM part on the left shares the same embedding layer with the deep neural network part on the right. The FM part on the left crosses the embeddings of different feature fields in pairs, that is, the embedding vector is treated as the feature hidden vector in the original FM. Finally, the output of the FM and the output of the deep part are input into the final output layer to participate in the final prediction.

Compared with the Wide&Deep model, the improvement of the DeepFM model is mainly aimed at mitigating the shortage that the wide part of the Wide&Deep model does not have the ability to automatically cross features. The motivation for improvement here is exactly the same as that of the Deep and Cross model. The only difference is that the Deep and Cross model uses a multilayer cross network for feature

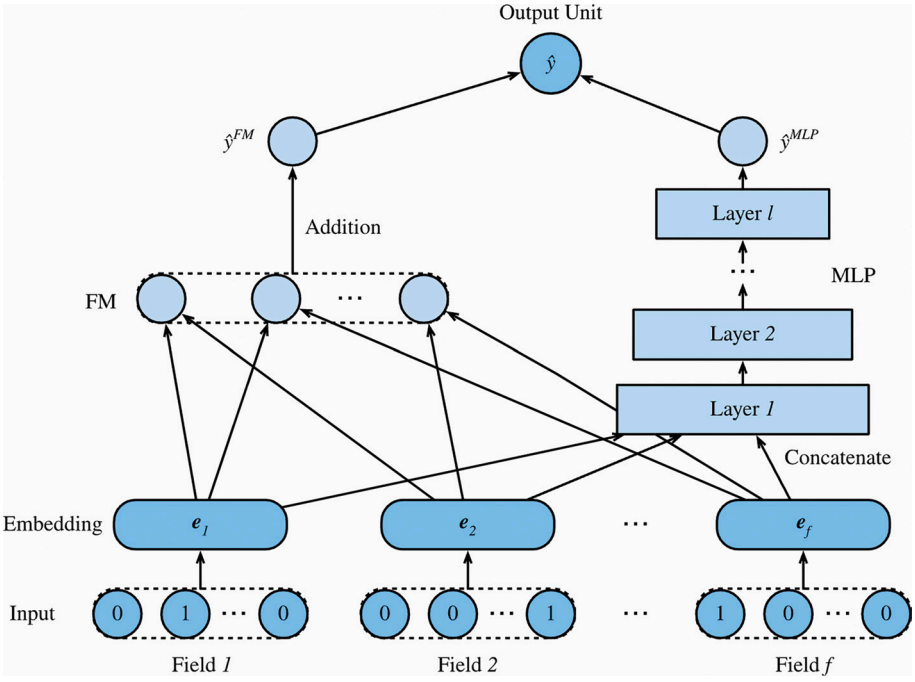

Figure 3.19 The structure diagram of the DeepFM model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

combination, while the DeepFM model uses FM structure for feature combination. Of course, the specific application effects still need to be compared through experiments.

3.7.3 NFM: FM Model's Neural Network Attempt

When we introduced the limitations of FM in Section 2.5, it mentions that whether it is FM or its improved model FFM, it is still basically a simple model with the second-order feature intersection. Affected by the "Curse of Dimensionality" issue, it is almost impossible for FM to extend the feature crossing beyond the third order, which inevitably limits the expressivity of the FM model. So is it feasible to use the stronger expressive power of deep neural networks to improve the FM model? In 2017, researchers from the National University of Singapore made an attempt to explore this and proposed the NFM [10] model.

The mathematical form of the classical FM is presented in Eq. 3.10. The main idea of the NFM model is to replace the part of the inner product of the second-order latent vector in the original FM with a function with stronger expressivity, as shown in Figure 3.20.

If the traditional machine learning idea is used to design the function $f(x)$ in the NFM model, it usually leads to a more expressive function through a series of mathematical derivations. But after entering the era of deep learning, since theoretically the deep learning network has the ability to fit any complex function, the construction of $f(x)$ can be completed by a deep learning network and learned through gradient backpropagation. In the NFM model, the neural network structure used to replace the second-order part of the FM is shown in Figure 3.21.

The characteristic is to add a feature cross-pooling layer (Bi-Interaction Pooling Layer) between the embedding layer and the multilayer neural network. Assuming that V_x is the embedding set of all feature domains, the specific operation of the feature cross-pooling layer is shown in Eq. 3.11,

$$f_{\mathrm{BI}}\left(V_{\boldsymbol{x}}\right)=\sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(x_i v_i\right)\odot\left(x_j\, v_j\right) \tag{3.11}$$

where $\odot$ represents the element-wise product operation of two vectors, that is, the corresponding dimension of two vectors with the same length is multiplied to obtain an element-wise product vector. The element-wise product operation on the kth dimension is as follows,

$$\left(v_i \odot v_j\right)_k = \boldsymbol{v}_{ik}\boldsymbol{v}_{jk} \tag{3.12}$$

After performing the element-wise product operation of the two embedding vectors, the crossed feature vectors are summed to obtain the output vector of the pooling layer. The vector is then input into the upper multilayer fully connected neural network for further interaction.

The first-order structure has been omitted in the NFM architecture diagram shown in Figure 3.21. If the first-order part of NFM is viewed as a linear model, then the architecture of NFM is equivalent to the evolution of the Wide&Deep model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

$$\hat{y}_{\mathrm{FM}}(x) = w_0 + \sum_{i=1}^{N} w_i x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \boldsymbol{v}_i^{\mathrm{T}} \boldsymbol{v}_j \cdot x_i x_j$$

$$\hat{y}_{\mathrm{NFM}}(x) = w_0 + \sum_{i=1}^{N} w_i x_i + f(x)$$

Figure 3.20 Improvement of NFM to the second-order term of FM.

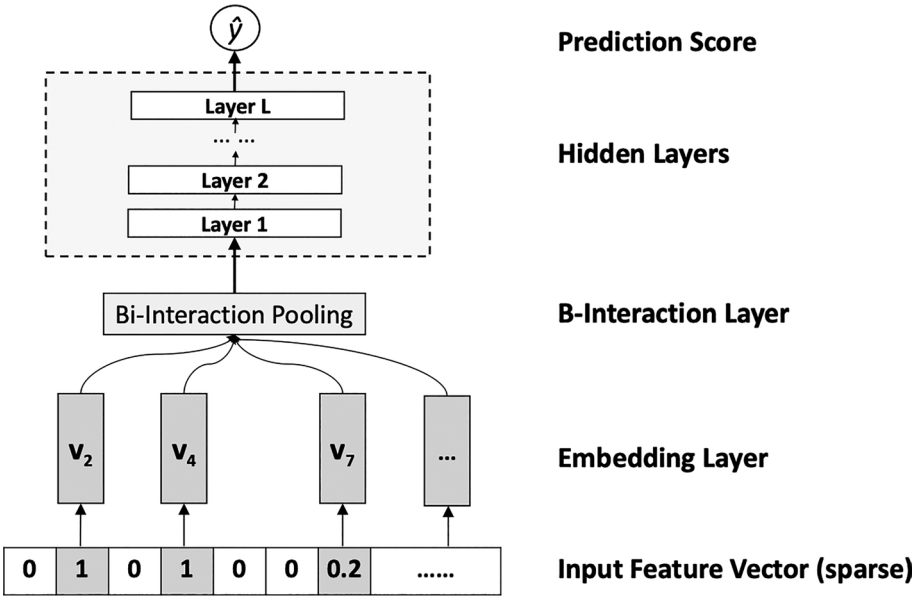

Figure 3.21 The model structure (partial) of the NFM model.

Compared with the original Wide&Deep model, the NFM model adds a cross-pooling layer to its deep part, which strengthens the feature interaction. This is another aspect to understand with the NFM model.

3.7.4 Strengths and Limitations of FM-Based Deep Learning Models

This section introduces three deep learning models (FNN, DeepFM, NFM) that were developed on top of FM approach. They are all characterized by adding targeted feature crossover operations to classic multilayer neural networks, so that the model has stronger nonlinear expressivities.

Following the idea of feature engineering automation, the deep learning model has come all the way from PNN, through Wide&Deep, Deep and Cross, FNN, DeepFM,

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

NFM and other models, and has made a lot of attempts based on different feature interaction ideas. However, the idea of feature engineering has almost exhausted all possible attempts, and the room for further improvement of the model is quite small, which is also one limitation of such type of model.

Since then, more and more deep learning recommendation models have started to explore some "structural" modifications. For example, attention mechanism, sequence model, reinforcement learning, and other model structures that shine in other fields have gradually entered the field of recommendation world. These attempts have achieved remarkable results in the improvement of the recommendation model.

3.8 Application of Attention Mechanism in the Recommendation Model

The "attention mechanism" comes from the natural human habit of attention. The most typical example is that when users browse the web, they will selectively pay attention to specific areas of the page and ignore other areas. Figure 3.22 is the heat map of page attention from research by the Google Search team by conducting eye-tracking experiments on a large number of users. It can be seen that the distribution of users' attention to the areas of the page is very different. Based on this observation, it indicates that considering the attention mechanism on the prediction in the modeling process may result in some good benefits.

In recent years, the attention mechanisms have been widely used in various fields of deep learning studies, and attention models have achieved great success in the fields of natural language processing, speech recognition, or computer vision. Since 2017, the recommendation field has also begun to try to introduce the attention mechanism into the model, among which the most influential works are AFM [11], proposed by Zhejiang University, and DIN [12], proposed by Alibaba.

3.8.1 AFM: FM Model with Attention Mechanism

The AFM model can be considered as a continuation of the NFM model introduced in Section 3.7. In the NFM model, the feature embedding vectors of different domains are crossed by the feature cross-pooling layer, and the crossed feature vectors are "summed" and input to the output layer through a multilayer neural network. The crux of the problem lies in the operation of sum pooling, which is equivalent to treating all intersecting features "equally," regardless of the degree of influence of different features on the result. In fact, this sum operation eliminates a lot of valuable information.

Here, the "attention mechanism" comes in handy. It is based on the assumption that different crossed features have different effects on the results. Take a more intuitive business scenario as an example to illustrate how users may pay different attention to different cross-features. If the application scenario is to predict the likelihood of a male user buying a keyboard, the cross-feature of "gender=male & purchase history

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

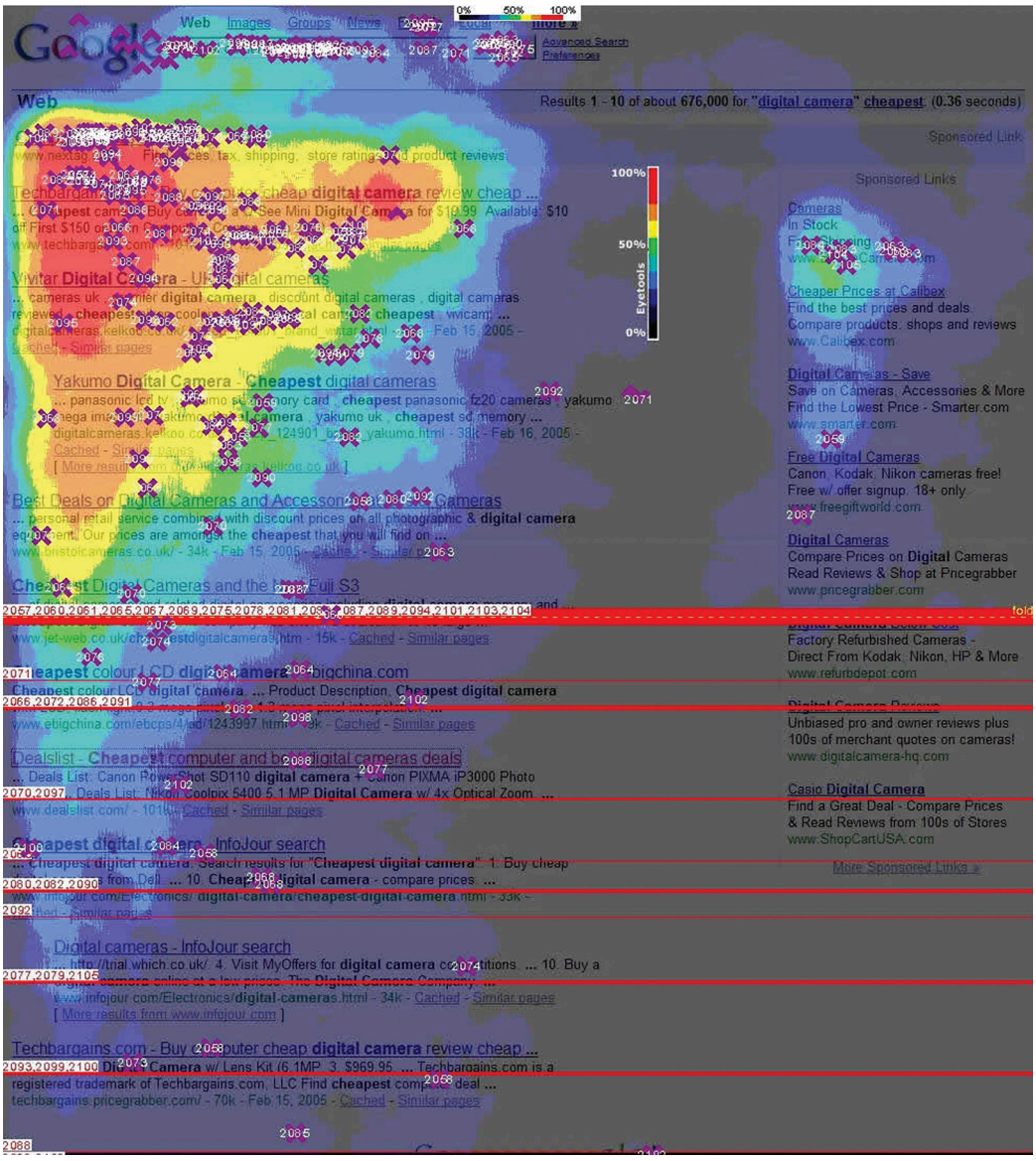

Figure 3.22 Google search engine page attention heatmap.

includes a mouse" is likely more important than the feature "gender=male & user age=30." Thus, the model pays more "attention" to the preceding features. Because of this, it makes sense to combine the attention mechanism with the NFM model.

Specifically, the AFM model introduces an attention mechanism by adding an attention net between the feature intersection layer and the final output layer. The model structure of AFM is shown in Figure 3.23. The role of the attention network is to provide weights for each cross feature, that is, the attention score.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

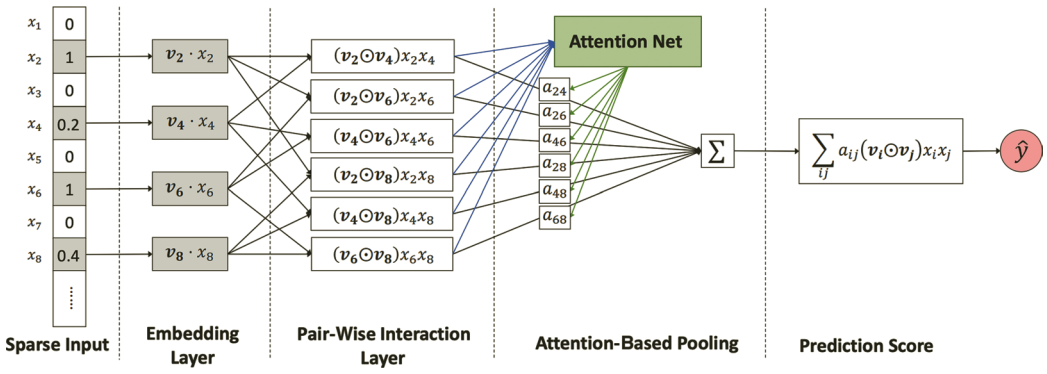

Figure 3.23 The structure diagram of the AFM model.

Like NFM, the feature intersection process of AFM also uses the element-wise product operation, as shown next,

$$f_{\mathrm{PI}}(\varepsilon)=\left\{\left(\boldsymbol{v}_i \odot \boldsymbol{v}_j\right) x_i x_j\right\}_{(i,j)\in\mathcal{R}_x} \tag{3.13}$$

The pooling process after AFM is added to the attention score is expressed as,

$$f_{\mathrm{Att}}\left(f_{\mathrm{PI}}(\varepsilon)\right)=\sum_{(i,j)\in\mathcal{R}_x} a_{ij}\left(\boldsymbol{v}_i \odot \boldsymbol{v}_j\right) x_i x_j \tag{3.14}$$

For the attention score a_{ij}, the easiest presentation method is to use a weight parameter. But in order to prevent the weight parameter from convergence issues due to the sparse problem of crossed feature, the AFM model uses a pairwise feature interaction and the attention network between the pooling layer to generate the attention score.

The structure of the attention network is a simple structure of a single fully connected layer plus a softmax output layer, and its mathematical form can be expressed as,

$$\begin{aligned} a'_{ij=}&\boldsymbol{h}^T \mathrm{ReLU}\left(\boldsymbol{W}\left(\boldsymbol{v}_i \odot \boldsymbol{v}_j\right) x_i x_j + \boldsymbol{b}\right) \\ a_{ij} &= \frac{\exp\left(a'_{ij}\right)}{\sum_{(i,j)\in\mathcal{R}_x} \exp\left(a'_{ij}\right)} \end{aligned} \tag{3.15}$$

The model parameters to be learned are the weight matrix $\boldsymbol{W}$ from the feature intersection layer to the fully connected layer of the attention network, the bias vector $\boldsymbol{b}$, and the weight vector $\boldsymbol{h}$ from the fully connected layer to the softmax output layer. Together with the other components in the model, the attention network is also trained through backpropagation to obtain the final weight parameters.

AFM is a positive attempt by researchers to improve the model structure. It has nothing to do with specific application scenarios. However, Alibaba's introduction of the attention mechanism into its deep learning recommendation model is a model improvement based on business observation. Next we will introduce Alibaba's well-known recommendation model in the industry: the DIN model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.8.2 DIN: Deep Learning Network with Attention Mechanism

Compared with many previous deep learning models with academic style, the DIN model proposed by Alibaba is obviously more business-centric. Its application scenario is Alibaba's e-commerce advertisement recommendation. When predicting the probability of a user u clicking on an advertisement a, the input features of the model are naturally divided into two parts. One part is the feature group of user u, as shown in Figure 3.24, and the other part is the feature group of candidate advertisement a, as shown in the advertisement feature group in Figure 3.24. Both users and advertisements contain two very important features – product ID (good_id) and shop ID (shop_id). The product ID in the user feature is a sequence, representing the set of products that the user has clicked on, and the same is true for the store ID. The product ID and store ID in the advertisement feature set are the IDs corresponding to the advertisement (the advertisement on the Alibaba platform). Most of them are products that participate in some promotional program).

In the original basic model (the base model in Figure 3.24), the product sequence and store sequence in the user feature group enter the upper neural network for further training after a simple average pooling operation. The product and store sequences have not distinguished the level of importance, and have no explicit relationship with the product ID in the advertisement features.

However, in fact, the degree of correlation between advertising features and user features is very strong, and the use case introduced in Section 3.7 can illustrate the strong correlations. Assuming that the product in the advertisement is a keyboard, there are several different product IDs in the user's click history, for example, mouse, T-shirt, and facial cleanser. Based on common sense, the historical commodity ID of "mouse" should be more important for predicting the click-through rate of "keyboard" ads than the latter two. From the model's point of view, the "attention" given to different features in the modeling process should be different, and the calculation of the "attention score" should be related to the advertising features.

It is also intuitive to reflect the aforementioned idea of "attention" into the model. A weight is calculated by using the correlation between candidate products and historically interacted products. This weight represents the strength of "attention." The DIN model adds the attention weight in the network structure, in which the attention part is formularized as,

$$\boldsymbol{V}_{\mathrm{u}} = f\left(\boldsymbol{V}_{\mathrm{a}}\right) = \sum_{i=1}^{N} w_i \cdot \boldsymbol{V}_i = \sum_{i=1}^{N} g\left(\boldsymbol{V}_i, \boldsymbol{V}_a\right) \cdot \boldsymbol{V}_i \tag{3.16}$$

where $\boldsymbol{V}_{\mathrm{u}}$ is the embedding vector of the user u, $\boldsymbol{V}_{\mathrm{a}}$ is the embedding vector of the candidate advertisement product, and $\boldsymbol{V}_i$ is the embedding vector of the ith action of the user u. Here, the user's action is to browse the product or store, so the embedding vector of the action is the embedding vector of the browsed product or store.

Because the attention mechanism is added, $\boldsymbol{V}_{\mathrm{u}}$ has changed from the simple sum of $\boldsymbol{V}_i$ in the past to the weighted sum of $\boldsymbol{V}_i$ and the weight w_i of $\boldsymbol{V}_i$ is determined by the relationship between $\boldsymbol{V}_i$ and $\boldsymbol{V}_a$, which is $g(\boldsymbol{V}_i, \boldsymbol{V}_{\mathrm{a}})$ in Eq. 3.16. This term is also known as the Attention Score.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

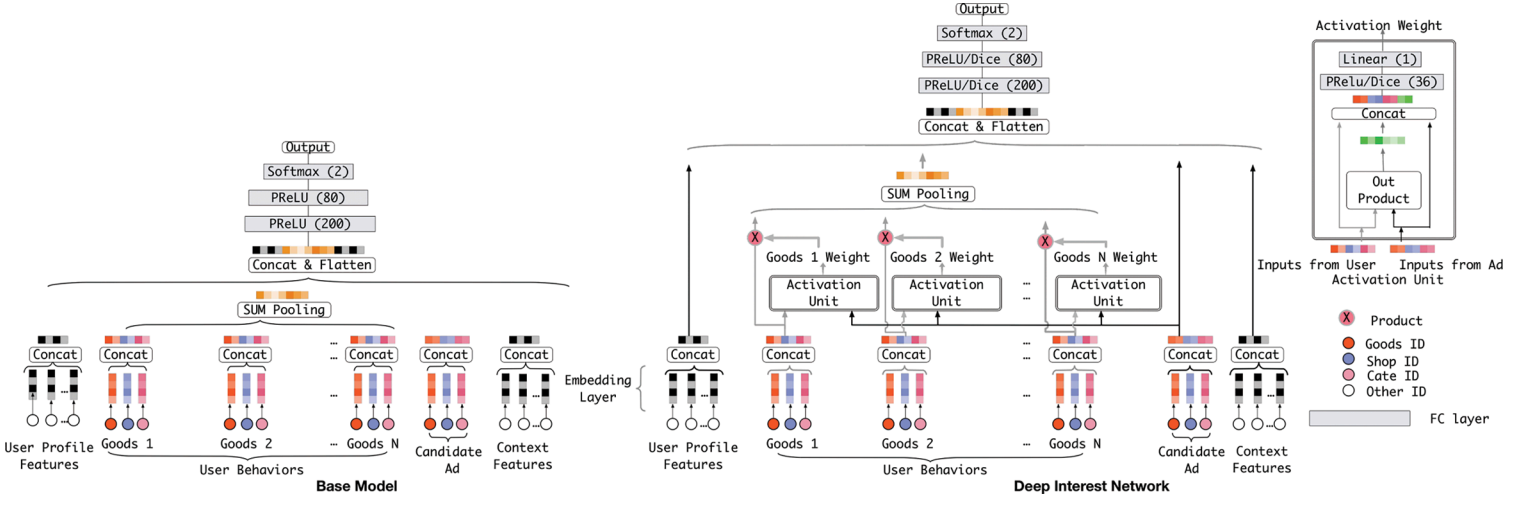

Figure 3.24 The structure diagrams of the base model and DIN model.

Then, what is the good representation for the $g(\boldsymbol{V}_i, \boldsymbol{V}_a)$ function? The answer is to use an attention activation unit to generate the attention score. This attention activation unit is essentially a small neural network, and its specific structure is shown in the activation unit at the upper right corner of Figure 3.24.

It can be seen that the input layer of the activation unit is two embedding vectors. After the element-wise minus operation, they are connected with the original embedding vector to form the input of the fully connected layer. Finally, the attention score is generated through the single neuron output layer.

If you pay attention to the red line in Figure 3.24, you can find that the store ID from the advertisement feature only interacts with the store ID sequence in the user's historical behavior, and the product ID of advertisement only works with the user's product ID sequence, as the weight of attention should be determined more by the correlation of same category of information.

Compared with the FM-based AFM model, the DIN model is a more typical attempt to improve the deep learning network structure. Since the introduction of the DIN model starts from an actual business scenario, it also gives recommendation engineers more substantial inspiration.

3.8.3 Inspiration of Attention Mechanism to Recommender Systems

From the perspective of the mathematical formula, the attention mechanism just replaces the past average or sum operation with a weighted sum or weighted average operation. However, the inspiration of this mechanism for deep learning recommender systems is significant, because the introduction of "attention score" reflects the innate "attention mechanism" characteristics of human beings. The simulation of this mechanism makes the recommender system's logic closer to the user's real thinking process, so as to achieve the purpose of improving the recommendation effect.

Starting from the "attention mechanism," more and more improvements to the structure of deep learning models are based on deep observations of user behavior. Compared with academia, which pays more attention to theoretical innovation, recommendation engineers in the industry need to focus more on their understanding of the actual business problem while developing new recommendation models.

3.9 DIEN: Combination of Sequence Model and Recommender Systems

After Alibaba proposed the DIN model, it did not stop the evolution of its recommendation model, and formally introduced an updated version of the DIN model, DIEN [13], in 2019. The application scenario of the DIEN model is exactly the same as that of DIN. So we will not repeat it in this section. The innovation lies in simulating the evolution process of user interest with the sequence model. The main ideas of DIEN and the design of the interest evolution part are introduced in detail next.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.9.1 Motivation of the DIEN Model

No matter whether it is e-commerce purchase history, video website viewing history, or news application reading history, the historical behavior of a specific user can be always considered as a time sequence. Since it is a time series problem, there must be some level of dependency on the chronological order among the history items. Such chronological information is undoubtedly valuable for the recommendation process. But do all the models introduced earlier in this chapter make use of this sequential information? The answer is negative. Even the AFM or DIN model that introduces the attention mechanism only scores the importance of different actions, which is time-independent and sequence-independent.

Why is sequential information valuable for recommendation? The behavior of a typical e-commerce user can illustrate this point. For a common e-commerce business, the migration of user interests is actually very fast. For example, a user was picking a pair of basketball shoes last week. After he completes his purchase, his shopping interest this week may turn to buying a mechanical keyboard. The importance of sequence information lies in:

(1) It reinforces the influence of recent behavior on the prediction of the next behavior. In the previous example, the probability that the user has recently purchased a mechanical keyboard is significantly higher than the probability of buying another pair of basketball shoes.
(2) Sequential models can learn information about buying trends. In this example, the sequence model can establish the transition probability from "basketball shoes" to "mechanical keyboard" to a certain extent. If this transition probability is high enough in a global statistical sense, recommending a mechanical keyboard will be a good option when users buy basketball shoes. Intuitively, the user groups of the two are likely to be the same.

If the sequence information is abandoned, the model's ability to learn the time-based or trend-based information can be quite weak. The recommendation model without considering sequential dimension just generates a prediction based on the user's overall purchase history, rather than providing a 'next purchase' recommendation. Obviously, from a business point of view, the sequence model is the correct objective of a recommender system.

3.9.2 Network Structure of the DIEN Model

Based on the motivation of introducing "sequential" information, Alibaba has further developed the DIN model and eventually formed the structure of the DIEN model. As shown in Figure 3.25, the model is still composed of an input layer, an embedding layer, a connection layer, a multilayer fully connected neural network, and the final output layer. The colored "interest evolution network" in the figure is considered to be an embedding representation of user interest, and its final output is the user interest vector $h'(T)$. The innovation of the DIEN model is how to build an interest evolution network in a recommendation model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

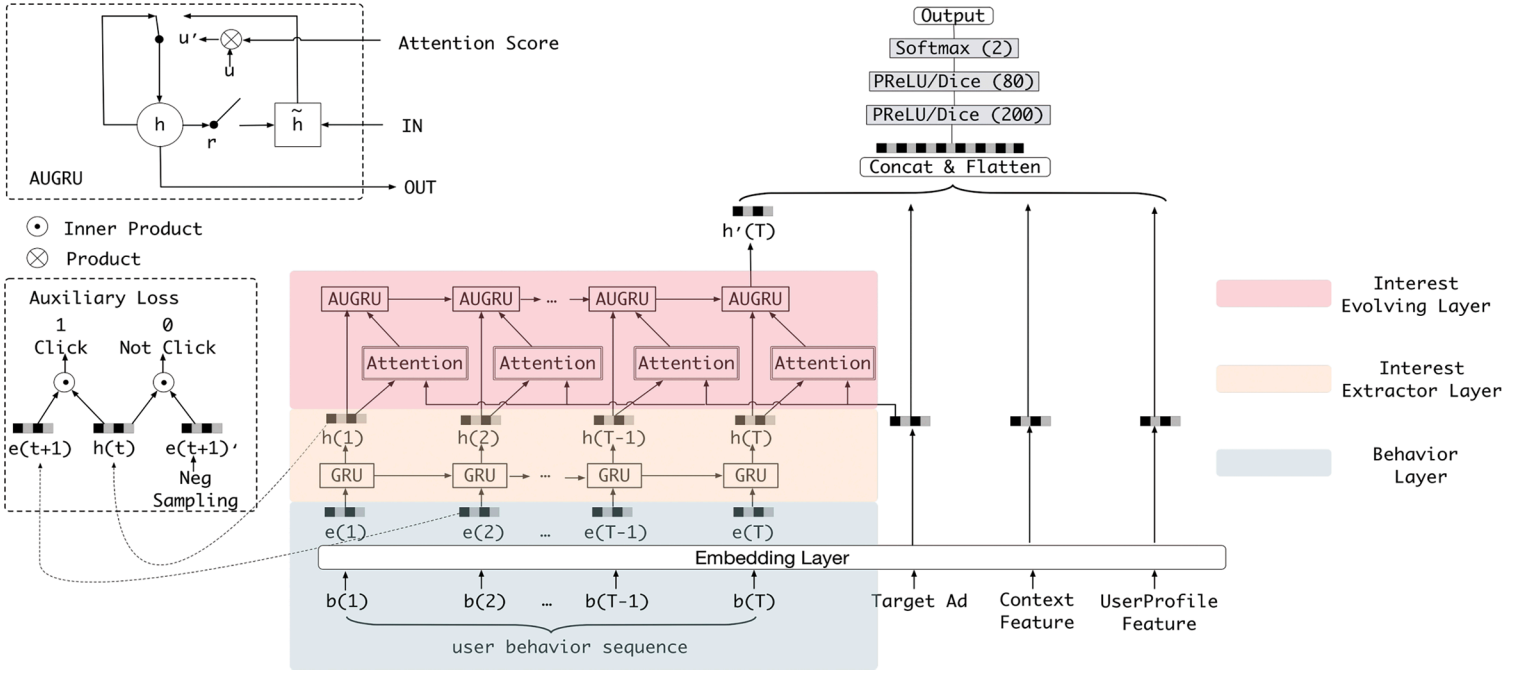

Figure 3.25 The structure diagram of the DIEN model.

The interest evolution network is divided into three layers from bottom to top:

(1) Behavior Layer (color green): converts the original behavior sequence into an embedding behavior sequence;
(2) Interest Extraction Layer (color beige): its main function is to extract user interests by simulating the process of user interest migration;
(3) Interest Evolving Layer (light red): this layer simulates the interest evolution process related to the current target advertisement by adding an attention mechanism based on the interest extraction layer.

In the interest evolution network, the structure of the behavior sequence layer is consistent with the typical embedding layer. The key to simulating the evolution of user interests lies mainly in the interest extraction layer and interest evolution layer.

3.9.3 Interest Extraction Layer

The basic structure of the interest extraction layer is a Gated Recurrent Unit (GRU) network. Compared with the traditional sequence model RNN (recurrent neural network), GRU solves the vanishing gradients problem commonly seen in RNN. Compared with LSTM (long short-term memory network), GRU has fewer parameters and faster training convergence speed. All of the aforementioned reasons result in the final adoption of the GRU network in the DIEN model.

The specific form of each GRU unit is defined as:

$$\begin{aligned} \boldsymbol{u}_t &= \sigma\left(\boldsymbol{W}^u \boldsymbol{i}_t + \boldsymbol{U}^u \boldsymbol{h}_{t-1} + \boldsymbol{b}^u\right) \\ \boldsymbol{r}_t &= \sigma\left(\boldsymbol{W}^r \boldsymbol{i}_t + \boldsymbol{U}^r \boldsymbol{h}_{t-1} + \boldsymbol{b}^r\right) \\ \widetilde{\boldsymbol{h}_t} &= \tanh\left(\boldsymbol{W}^h \boldsymbol{i}_t + r_t \circ \boldsymbol{U}^h \boldsymbol{h}_{t-1} + \boldsymbol{b}^h\right) \\ \boldsymbol{h}_t &= \left(1 - \boldsymbol{u}_t\right) \circ \boldsymbol{h}_{t-1} + \boldsymbol{u}_t \circ \tilde{\boldsymbol{h}}_t \end{aligned} \tag{3.17}$$

where σ is the sigmoid activation function, $\circ$ is the element-wise product operation, $\boldsymbol{W}^u, \boldsymbol{W}^r, \boldsymbol{W}^h, \boldsymbol{U}^z, \boldsymbol{U}^r, \boldsymbol{U}^h$ are six sets of parameter matrices to be learned. $\boldsymbol{i}_t$ is the input state vector, that is the embedding vector $e(t)$ of each behavior in the behavior sequence layer. $\boldsymbol{h}_t$ is the tth hidden state vector in the GRU network

Following the interest extraction layer with multiple GRUs, the user's behavior vector $\boldsymbol{b}(t)$ is further abstracted to form the interest state vector $\boldsymbol{h}(t)$. In theory, based on the sequence of interest state vectors, the GRU network can already predict the next interest state vector, but why does DIEN further add the interest evolution layer?

3.9.4 Structure of Interest Evolution Layer

The biggest distinction between the interest evolution layer and the interest extraction layer is the addition of an attention mechanism. This mechanism is in the same vein as DIN. It can be seen from the connection of the attention units in

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

Figure 3.25. The generation process of the attention score of the interest evolution layer is exactly the same as that of DIN, which is the result of the interaction between the current state vector and the target advertisement vector. That is to say, DIEN needs to consider the relevance of targeted advertisements in the process of simulating interest evolution.

This also answers the question at the end of Section 3.9.3. The interest evolution layer is added on top of the interest extraction layer in order to simulate the interest evolution path related to the target advertisement in a more targeted manner. Due to the characteristics of e-commerce such as Alibaba, users are very likely to purchase multiple categories of goods at the same time. For example, while purchasing a "mechanical keyboard," they are still viewing the goods under the "clothing" category. As a result, the attention mechanism is particularly important under such condition. When the target advertisement is an electronic product, the interest evolution path related to the purchase of "mechanical keyboard" is obviously more important than the evolution path of purchasing "clothes." Such distinction logic doesn't exist in the interest extraction layer.

The interest evolution layer achieves application of the attention mechanism by adopting the GRU with Attentional Update gate (AUGRU) structure. AUGRU adds the attention score to the structure of the update gate of the original GRU. The specific form is shown in Eq. 3.18:

$$\begin{aligned} \tilde{\boldsymbol{u}}_t^{'} &= \boldsymbol{a}_t \cdot \boldsymbol{u}_t^{'} \\ \boldsymbol{h}_t^{'} &= \left(1 - \tilde{\boldsymbol{u}}_t^{'}\right) \circ \boldsymbol{h}_{t-1}^{'} + \tilde{\boldsymbol{u}}_t^{'} \circ \tilde{\boldsymbol{h}}_t^{'} \end{aligned} \tag{3.18}$$

Comparing with Eq. 3.17, it can be seen that AUGRU adds the attention score a_t on the basis of the original $\boldsymbol{u}_t^{'}$, where $\boldsymbol{u}_t^{'}$ is the original update gating vector and similar to $\boldsymbol{u}_t$ in Eq. 3.17. The generation method of the attention score is basically the same as that of DIN, which uses the attention activation units.

3.9.5 Inspiration of the Sequence Model to Recommender Systems

This section introduces Alibaba's recommendation model DIEN that incorporates sequence models. Because the sequence model has a strong ability to express time series, it is very suitable for predicting the user's next action after a series of behaviors.

In fact, it is not only Alibaba that has successfully applied the sequence model to its e-commerce recommendation model, but video streaming companies such as YouTube and Netflix have also successfully applied the sequence model to their video recommendation models to predict the user's next streaming preferences (such as next watch).

However, it is necessary to pay attention to the high training cost of the model and the latency in online inferencing caused by serial prediction in a large sequence model. The complexity of sequence model undoubtedly increases the difficulty of its productization. So system optimization turns very important in the engineering implementation. Experiences with implementing a sequence model in production will be discussed in Chapter 8.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.10 Combination of Reinforcement Learning and Recommender Systems

Reinforcement learning is a very popular research topic in the field of machine learning in recent years. It is originated from the field of robotic studies, and aimed at modeling the decision-making and learning process of an agent in a changing environment. In the learning process of the agent, it will complete the collection of external feedback (Reward), change its own state (State), and then make decisions on the next action (Action) according to its current state, and continue to repeat the cycle. This process is usually referred to as the "action-feedback-state update" cycle.

The concept of agent is very similar to the robots, and the entire reinforcement learning process can be understood by analogizing the robots learning human actions. If the recommender system is viewed as an agent, with its learning and updating process equivalent to the agent's cycle of "action-feedback-state update," then applying reinforcement learning concepts to recommender systems becomes much more intuitive.

In 2018, the reinforcement learning model DRN [14] was firstly proposed by researchers from Penn State University and Microsoft Research Asia. This was an attempt to apply reinforcement learning knowledge to news recommendation.

3.10.1 Deep Reinforcement Learning Recommender Systems Framework

The deep reinforcement learning recommender systems framework is proposed based on the classic process of reinforcement learning. Readers can use the specific scenarios of the recommender system to further familiarize themselves with the concepts of agent, environment, state, action, and feedback in reinforcement learning. As shown in Figure 3.26, the diagram clearly shows the various components of the deep reinforcement learning recommender systems framework and the iterative process of the entire reinforcement learning. The specific explanation of each element in the recommender systems scenario is as follows:

- **Agent:** The recommender system itself, which includes a recommendation model based on deep learning, an exploration strategy, and related data storage (memory);
- **Environment:** The external environment of the entire recommender system consisting of news websites or apps, and users. In the environment, the user receives the recommended results and makes corresponding feedback;
- **Action:** For a news recommender system, an action refers to the system pushing ranked news to the user;
- **Feedback:** After the user receives the recommendation result, the user will give positive or negative feedback. For example, click behavior is considered to be a typical positive feedback, while impression but nonclick is a negative feedback

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

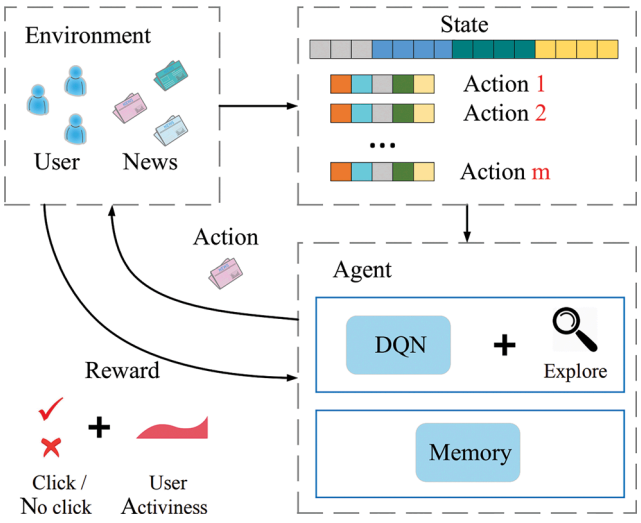

Figure 3.26 Deep reinforcement learning recommender systems framework.

signal. In addition, the user's activity level and the interval between the app opening are also considered as valuable feedback signals;

- **State:** State refers to the description of the environment and its current specific situation. In the news recommendation scenario, the state can be viewed as a feature vector representation of all actions and feedback received, as well as all relevant information about the user and news. From the perspective of traditional machine learning, "state" can be seen as the collection of all the data that has been received and can be used for training.

Under such a reinforcement learning framework, the learning process of the model can be iterated continuously. The iterative process mainly includes the following steps:

(1) Initialize the recommender system, which is the agent in this case.
(2) The recommender system ranks news (actions) based on the currently collected data (state) and pushes them to the website or app (environment).
(3) The user receives the recommendation list and clicks or ignores (feedback) the recommendation result.
(4) The recommender system receives feedback and updates the current state or updates the model through model training.
(5) Repeat the tasks from Step 2.

Readers may have realized that reinforcement learning models have an advantage over traditional deep models in that they can perform online learning. In other words, the reinforcement learning models can constantly update themselves with newly learned knowledge, and make timely adjustments and feedback. This is one of the advantages of applying reinforcement learning to recommender systems.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.10.2 Deep Reinforcement Learning Recommendation Models

The agent part is the core of the reinforcement learning framework. For the recommendation agent, the model is the "brain" of the system. In the DRN framework, the role of the "brain" is the Deep Q-Network, DQN for short, where Q is the abbreviation of "Quality." It means that by evaluating the quality of the action, the utility score of the action is calculated and used for decision-making.

The network structure of DQN is shown in Figure 3.27. The concepts of reinforcement learning – state vector and action vector – are applied in feature engineering. User features and context features are classified as state vectors, because they are action independent. User-news crossing features and news features are treated as action features since they are related to the action of recommending news.

User features and environmental features are fitted by the multilayer neural network on the left to generate a value score $V(\boldsymbol{s})$. The state vector and action vector are used to generate an advantage score $A(\boldsymbol{s}, \boldsymbol{a})$. Finally, the score from both parts are combined to obtain the final quality score $Q(\boldsymbol{s}, \boldsymbol{a})$.

3.10.3 Learning of the DRN Model

The learning process of DRN is the main focus of the entire reinforcement learning recommender systems framework. It is the online learning process that gives the reinforcement learning model more real-time advantages than other "static" deep learning models. Figure 3.28 vividly depicts the learning process of DRN in chronological order.

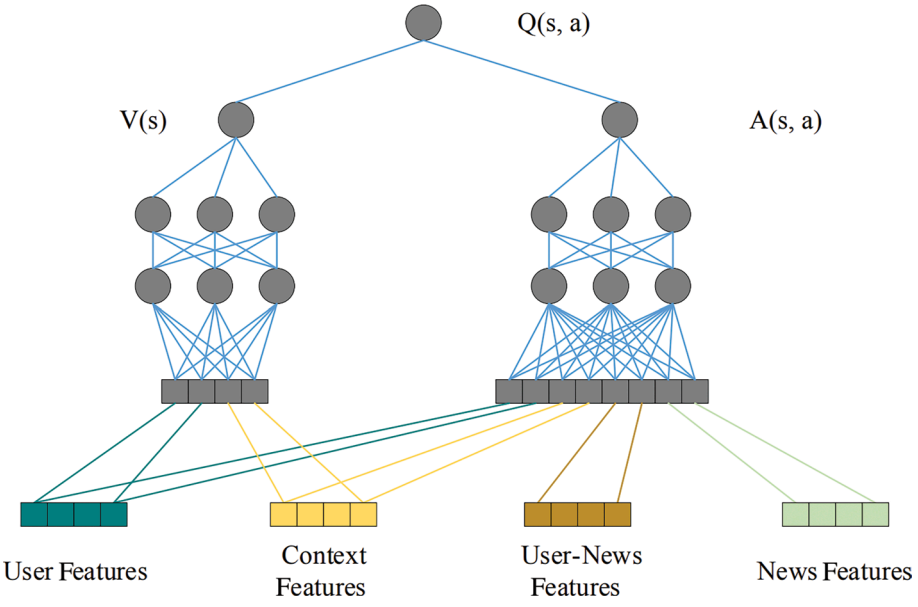

Figure 3.27 The model structure of the DQN model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

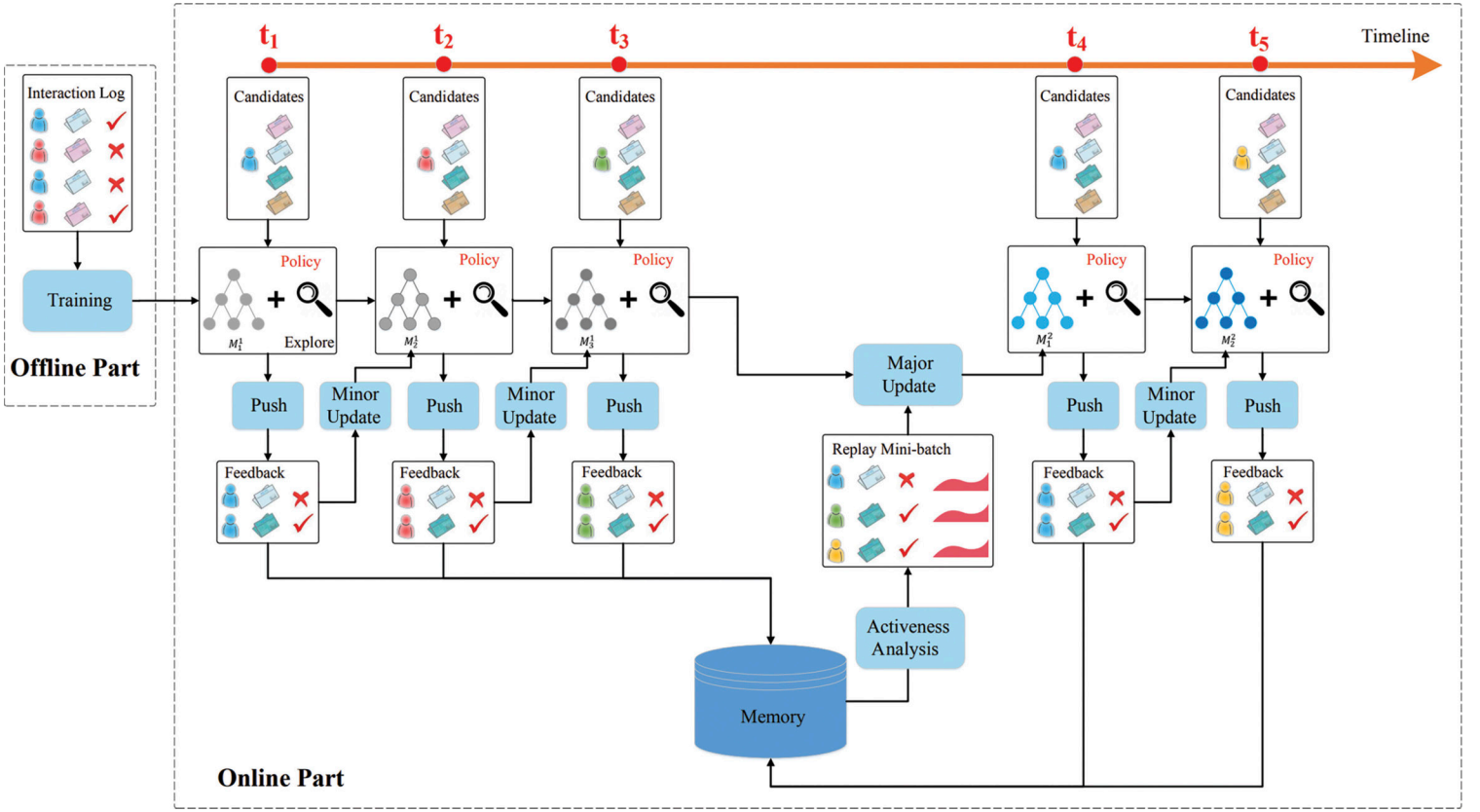

Figure 3.28 The learning process of the DRN model.

The important steps in the DRN learning process are illustrated in chronological order from left to right in Figure 3.28:

(1) In the offline part, the DQN model is trained according to the historical data as the initialization model of the agent;
(2) At the stage $t_1 \rightarrow t_2$, the initial model is used to power the recommendation in the push service for a period of time to accumulate feedback data;
(3) At the time point t_2, the user click data accumulated in the $t_1 \rightarrow t_2$ stage is used to perform a minor update of the model;
(4) At the time point t_4, a major update of the model is performed using the user click data and user activity data in the $t_1 \rightarrow t_4$ stage;
(5) Repeat Steps 2–4.

The model main update operation in Step 4 can be understood as retraining using historical data to replace the existing model with the trained model. So how does the minor update in Step 3 work? This involves a new online training method used by DRN – Dueling Bandit Gradient Descent Algorithm.

3.10.4 Online Learning of the DRN Model: Dueling Bandit Gradient Descent Algorithm

The flow of DRN's dueling bandit gradient descent algorithm is shown in Figure 3.29.

The main steps are as follows:

(1) For the current network Q that has been trained, add a small random perturbation $\Delta \boldsymbol{W}$ to its model parameter $\boldsymbol{W}$ to obtain a new model parameter $\widetilde{\boldsymbol{W}}$. Here the network corresponding to $\widetilde{\boldsymbol{W}}$ is called the exploration network $\tilde{Q}$;
(2) The recommendation lists L and $\tilde{L}$ are generated respectively with current network Q and the exploration network $\tilde{Q}$. Then, the two recommendation lists are combined into one recommendation list by interleaving (described in detail in Section 7.5) and pushed to the user.
(3) Collect user feedback in real-time. If the feedback of the content generated by the exploration network $\tilde{Q}$ is better than the current network Q, replace the current network with the exploration network and enter the next iteration, otherwise keep the current network

In the first step, the exploration network $\tilde{Q}$ is generated from the current network Q, and the formula for generating random disturbance is shown in Eq. 3.19,

$$\Delta \boldsymbol{W} = \alpha \bullet \text{rand}(-1,1) \bullet \boldsymbol{W} \tag{3.19}$$

where α is the exploration factor, which determines the degree of the exploration. $\text{rand}(-1,1)$ means a random number between $[-1,1]$.

The online learning process of DRN utilizes the idea of "exploration," and the granularity of the model updates can be refined to once per feedback. This process is very similar to the idea of stochastic gradient descent. Although the results of one sample may produce random disturbances, as long as the total decent trend is correct,

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

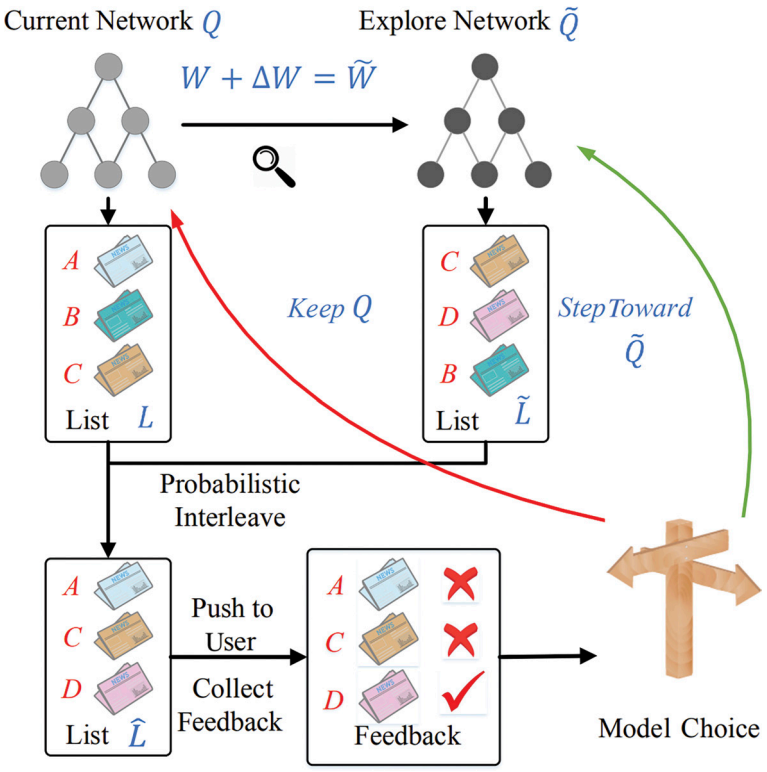

Figure 3.29 The online learning approach of the DRN model.

the optima can be finally reached through a large number of attempts. In this way, DRN keeps the model synchronized with the "freshest" data at all times, and integrates the latest feedback information into the model in real-time.

3.10.5 Inspiration of Reinforcement Learning for Recommender Systems

The application of reinforcement learning in recommender systems once again opens the world of recommendation models from a different angle. The difference between this and the other deep learning models mentioned earlier is that it changes the learning process from static to dynamic, which brings the importance of model real-time learning to a prominent position.

It also brings us a question worth thinking about – should we build a heavy-weight, "perfect" model with a large update delay, or should we build a lightweight and simple model that can be trained in real-time? Of course, there are no assumptions or conjectures in engineering systems, we can only tell which approach is better through actual experiment results. Also, the relationship between "weight" and "real-time" is by no means antagonistic, but before finalizing a technical solution, plenty of evaluation and experiments are necessary for this kind of real-world problem.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.11 Applications of BERT in a Recommendation Model

Bidirectional Encoder Representations from Transformers (BERT) is a powerful natural language processing model that was introduced by Google in 2018 [15] and achieved state-of-the-art performance in multiple NLP tasks. Like the attention mechanism introduced in Section 3.8, the BERT model was also borrowed into the recommendation world after its demonstrated success in the NLP field. Part of the application of the BERT model in recommender systems is still utilizing its ability to process and understand the natural languages and capture the semantic interpretations for the text data, which will not be covered in this section. In this section, we will mainly walk through two BERT-based recommendation models, the BERT for Recommendation (BERT4Rec) model [16] and User-News Matching BERT (UNBERT) model [17], which adopt the BERT model structure in the sequential recommendation scenarios.

Before introducing the BERT4Rec and UNBERT models, let us briefly review some foundations of the BERT – the Transformer model and self-attention mechanism.

Basics: The Transformer Model and Self-Attention Mechanism

The Transformer model is a revolutionary deep learning architecture that has had a significant impact on NLP tasks, and it builds a foundation for the development of many succeeding language models. Two well-known succeeding language models are the BERT model and the GPT model. The Transformer model was introduced in the famous paper titled "Attention is All You Need" by Vaswani et al. in 2017 [16]. Compared to the traditional RNN (Recurrent Neural Networks) and LSTM (Long Short-Term Memory Networks), the Transformer model has demonstrated state-of-the-art performance in various NLP tasks as well as increasing model training efficiency by increasing the training parallelism.

Now, we will briefly introduce the structure of the Transformer model (as depicted in Figure 3.30).

The transformer model consists of an encoder component (left in Figure 3.30) and a decoder component (right side in Figure 3.30).

Both the encoder and decoder are composed of a stack of N identical layers. On the encoder side, each layer has two sub-layers, one a multihead self-attention mechanism layer and one position-wise fully connected feed-forward layer. The output of each sub-layer is followed by a layer normalization and connected with the residual connections. On the decoder side, in addition to the multihead attention layer and feed-forward sublayer, there is a third sublayer to connect the output of the encoder stack with a multihead attention module.

To better understand the encoder and decoder structures, we need to grasp several key components and features first.

Self-Attention Mechanism: Unlike the attention mechanism introduced in Section 3.8, the self-attention mechanism is used to encode the sequence directly.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

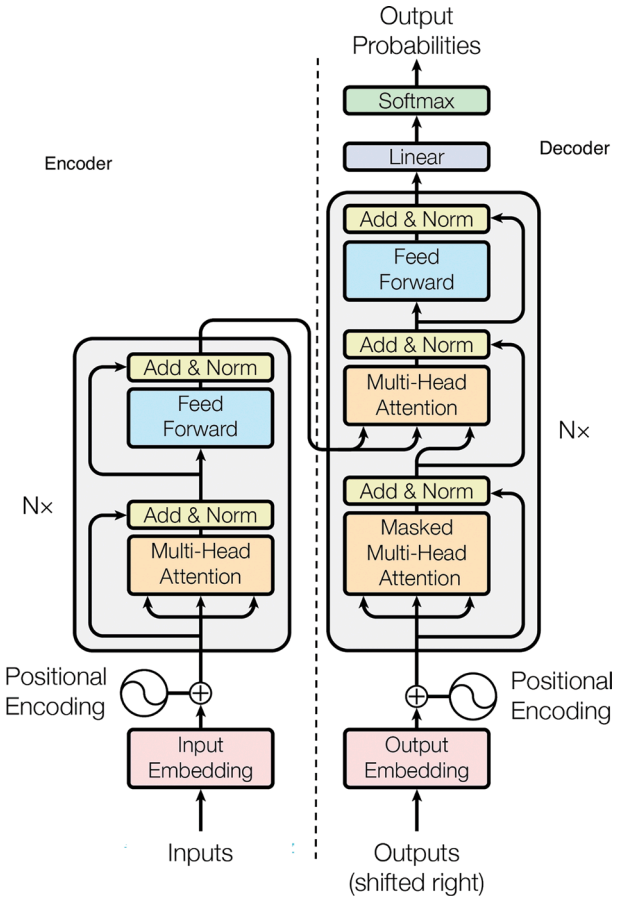

Figure 3.30 The Transformer model architecture [18].

It allows the Transformer model to weigh the importance of each token in the sequence with respect to all other tokens in the same sequence. The attention matrix (as shown in Figure 3.31(a)) is defined as

$$\text{Attention}\left(Q,K,V\right)=\text{soft}\max\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.20}$$

where matrices Q, K, and V are corresponding to "Query," "Key," and "Value," respectively. These matrices don't have actual physical meanings in the Transformer model; rather, they are borrowed from information retrieval contexts to help understanding.

Multihead Attention: The Transformer model adopted a multihead attention mechanism in both the encoder and decoder. The multihead attention layer is depicted in Figure 3.31(b). Compared to a single-head attention structure, the multihead attention mechanism leverages different linear projects to learn

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

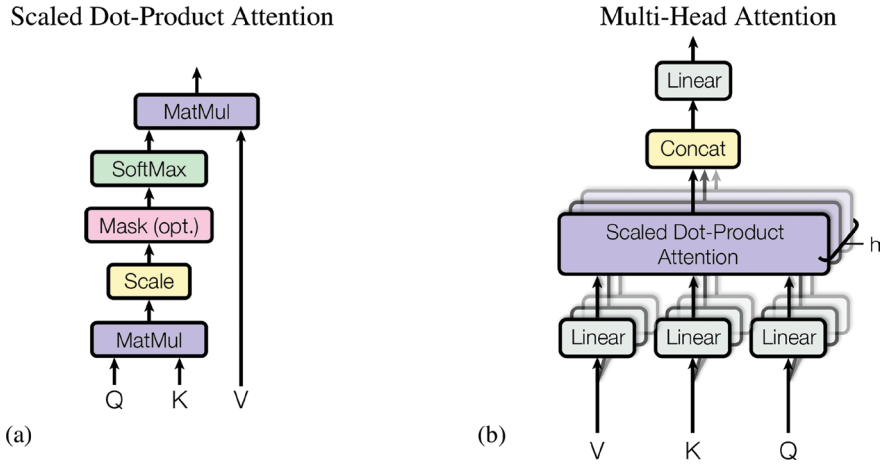

Figure 3.31 (a) Scaled dot-production attention unit; (b) illustration of multihead attention layer [18].

different relationships from training data. The multihead attention function can be described as:

$$\begin{aligned} &\text{Multihead Attention}\left(Q, K, V\right) = \text{Concat}\left(head_1, \ldots, head_2\right) W^O \\ &\text{where } head_i = Attention\left(Q_i, K_i, V_i\right) \end{aligned} \tag{3.21}$$

where Q_i, K_i, and V_i are query, key, and value matrices for $head_i$.

One benefit of the multihead attention mechanism is that each head attention matrices can be computed in parallel, which significantly improves training efficiency.

Positional Encoding: Unlike the traditional sequence models, the Transformer does not understand the positional order of different tokens in a sequence. In order to solve this problem, a positional encoding function is added to the input embeddings at the beginning of both encoder and decoder tasks. In the original paper, authors used the sine and cosine functions of different frequencies as follows,

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos / 1000^{2i/d_{model}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos / 1000^{2i/d_{model}}\right) \end{aligned} \tag{3.22}$$

where pos is the position and i is the dimension. According to the authors, these positional encoding functions were chosen because they could allow the model to easily learn to attend by relative positions.

There are some other key components in the Transformer model structure, like layer normalization, masking, and so on. These contents won't be discussed in detail; readers can refer to the original paper for more information.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.11.1 Relationship between BERT and Transformer

The BERT model is a specific implementation of the Transformer architecture, so it is actually one type Transformer model. But compared with the original Transformer model proposed in the "Attention is All You Need" paper [18], the BERT model has the following differences:

- **Model Structure:** Instead of using both encoder and decoder stacks, BERT just used a stack of encoders in the model structure.
- **Training:** The training steps of a BERT model in an NLP application usually involve two steps of training – pre-training and fine-tuning. BERT uses Masked Language Model (MLM) objectives and task-specific objectives in the fine-tuning task. In pre-training, the MLM objective enables the BERT model to fuse both left and right contexts. This is also where "bi-directional" is from, in the BERT model name.
- **Model Usage:** As the BERT model only includes encoder stacks, its direct output are vectors. As a result, the major applications of BERT model are embedding generations and classifications. However, the major use case for the Transformer model is sequence-to-sequence generation.

The following section mainly focuses on the BERT model's applications and its derivatives in recommender systems.

3.11.2 BERT4Rec: BERT for Recommendation Model

After the success of the BERT model in the NLP fields, people started to wonder if the BERT model structure could be also applied to some other fields to handle some other sequential machine learning tasks. In 2019, the BERT4Rec model was introduced, and it successfully transferred the BERT model approaches to sequential recommendations. Section 3.9 introduces a sequence recommendation model, DIEN, which uses an RNN to represent the user's historical behaviors and demonstrates the benefits of a sequential model in the next-item predictions. In contrast to the conventional RNN-based sequential model (as shown in Figure 3.32), the benefits of the BERT4Rec model structure mainly include:

(1) It can gather the learning from both previous and future items during the training.
(2) The multihead attention structure can make overall learning more efficient.

As with many other sequential models, the BERT4Rec model also targeted solving the next-item prediction problem, which can be described as predicting the interaction probability of each item in the candidate pool given the interaction history S_u for user u, The BERT4Rec model architecture is depicted in Figure 3.32(b). It consists of multiple stacks of transformer layer. The details in the Transformer units are illustrated in Figure 3.32(a). As with its predecessor, the BERT4Rec model only used the encoder in the Transformer units. For the output layer, a two-layer

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

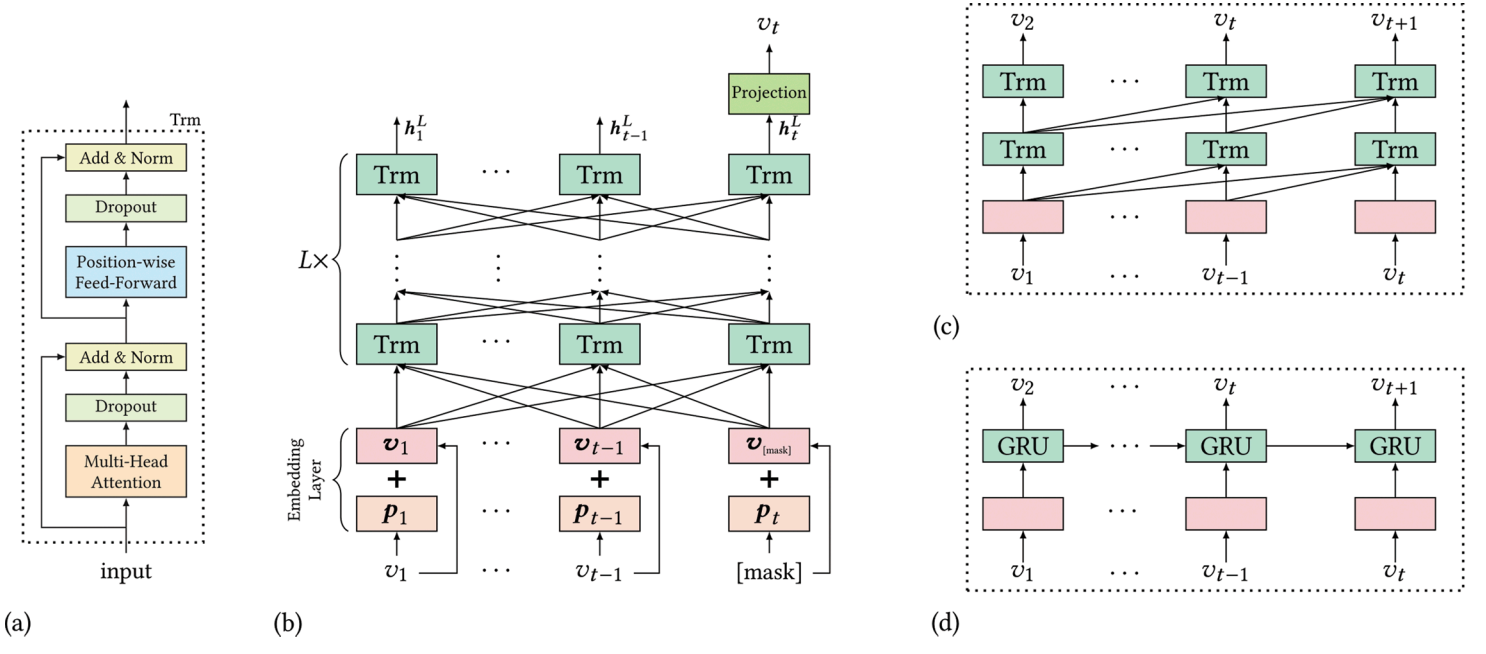

Figure 3.32 (a)Transformer layer, and its differences comparing with (b) BERT4Rec model architecture with the (c) unidirectional model SASRec and (d) conventional RNN-based sequence model [16].

Input: $[v_1, v_2, v_3, v_4, v_5] \xrightarrow{\text{randomly mask}} [v_1, [\text{mask}]_1, v_3, [\text{mask}]_2, v_5]$
Labels: $[\text{mask}]_1 = v_2, \quad [\text{mask}]_2 = v_4$

Figure 3.33 Randomly masked interaction sequence and training data generation.

feed-forward network with GELU activation to generate the probability distribution of all the candidate items,

$$P(v) = \text{softmax}(\text{GELU}(\boldsymbol{h}_t^L \boldsymbol{W}^p + \boldsymbol{b}^p)\boldsymbol{E}^T + \boldsymbol{b}^O \tag{3.23}$$

where $\boldsymbol{W}^p$ is the learnable projection matrix, $\boldsymbol{b}^p$ and $\boldsymbol{b}^O$ are the bias terms, and $\boldsymbol{E} \in \mathbb{R}^{|V| \times \boldsymbol{d}}$ is the embedding matrix for the item set V.

In the model training, BERT4Rec model adopted the same objective as the original BERT model, Masked Language Model objective, in the sequence recommendation to avoid information leaking. The items in the user behavior sequence were randomly masked as shown in Figure 3.33, and the correspondingly generated hidden vectors are passed into the output layer to generate the softmax matrix for training.

The negative log-likelihood of the masked targets is defined as the loss function for each masked input,

$$\mathcal{L} = \frac{1}{\left|S_u^m\right|} \sum_{v_m \in S_u^m} -\log P\left(v_m = v_m^* \mid S_u^{'}\right) \tag{3.24}$$

where $S_u^{'}$ is the masked version for user behavior history S_u, S_u^m is the random masked items in the user behavior history, and v_m^* is the true item for the masked item v_m.

For model inference, the special token "[mask]" is appended to the user interaction history, and then the entire input sequence is fed into the model to generate the predicted probabilities that the user interacts with each item. The item with the maximum probability will be the next recommended item. To make the BERT4Rec model output cover the target sequential recommendation task (that is, predicting the next item after a sequence of interacted items), the authors also added training samples that only mask the last item in training data.

3.11.3 UNBERT: A BERT-Based Model Combining Sequential Recommendation and NLP

The BERT4Rec model only borrows the model structure from BERT and transfers it to a sequential recommendation model. It does not have the advantage of the original BERT model's natural language understanding capabilities. This section introduces the UNBERT model (User-News Matching BERT model), which combines both language understanding and sequential recommendations in the same piece.

The application scenario of UNBERT mode is given a user u with a sequence of clicked news $[n_1^u, n_2^u, \ldots, n_{|n_u|}^u]$ and a set of candidate news $V_u = \{v_1, v_2, \ldots, v_{|V_u|}\}$. The objective of this model is to predict the click probability on the i-th candidate news v_i by user u. The probability score can be denoted by $\hat{y} = f(u, v_i)$.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

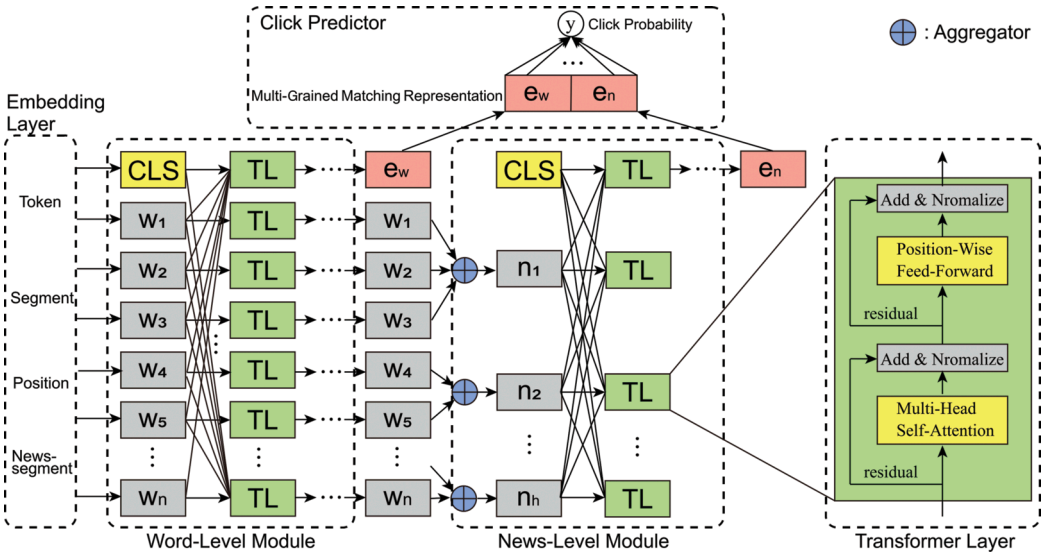

Figure 3.34 The overall architecture of the UNBERT model [17].

The model structure is depicted in Figure 3.34. The UNBERT model is mainly composed of two key modules – word-level module and news-level module, each of which can be considered a separate "BERT module." Each module consists of multiple layers of Transformer stack, including a multihead self-attention sublayer and a position-wise feed-forward layer.

Input Sequence and Embedding Layer

The input sequence construction and embedding layers are illustrated in Figure 3.35. It includes "News Sentence" and "User Sentence," where the News Sentence is simply the text description of the candidate news item, and the User Sentence is the concatenation of the news sequence that the user clicked in the history. The historical news items are separated by a special segment token (NSEP). Each clicked news is also represented by some text-based descriptions. The News Sentence and User Sentence are separated by another special token (SEP). Additionally, a classification token (CLS) is added at the beginning of concatenated sequence to help generate the classification embedding e_w as shown in Figure 3.34.

There are four layers of embeddings generated for each token – token embedding, segment embedding, position embedding, and news segment embedding. The token, segment, and position embeddings are trained using masked LM, and segment embedding is randomly initialized and further updated in the fine-tuning task. The final input token representation E_t is constructed by summing all four embeddings.

Word-Level Module

The word-level module (WLM) mainly applies the Transform Layers to the concatenated input sequence and generates hidden representations for the input tokens. The conventional encoder structure is adopted in the Transformer unit, including the

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

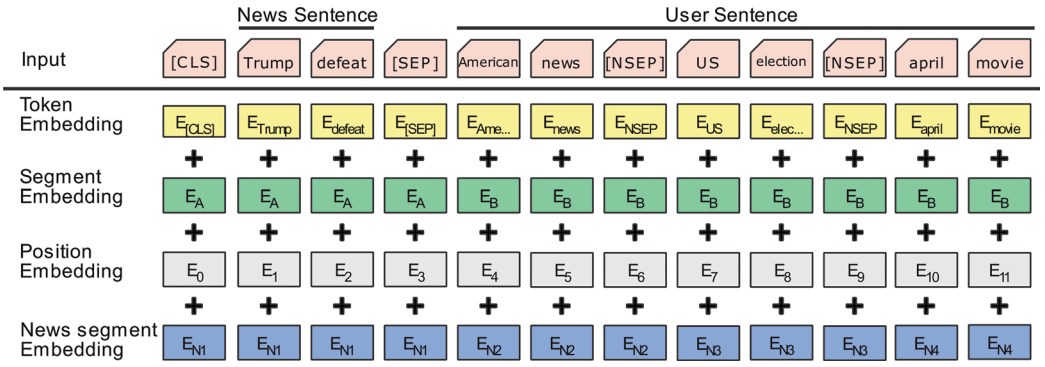

Figure 3.35 UNBERT input sequence construction and embedding layer structures.

multihead self-attention layer, the position-wise feed-forward layer, plus the residual connections and layer normalization between the two layers.

News-Level Module

The news-level module (NLM) aggregates the word's hidden representations of each news from the world-level module and feeds the aggregated vectors to multiple transformer layers to generate the final news representations and matching signal at the news level.

Three different aggregations were studied:

(1) The NSEP Aggregator directly used the generated embeddings of special tokens (NSEP) from the WLM output.
(2) The Mean Aggregator averages the word embeddings for each news segment.
(3) Attention Aggregators apply a lightweight attention network. The attention network applied a fully connected neural network with a *tanh* activation function. Then it connects with another fully connected neural network to generate the combination weights f. The weights then are applied in the linear combination of word embeddings as in Eq. 3.25,

$$n_j = \sum_{i \in S_j} f_i w_i \Big/ \sum_{i \in S_j} fi \tag{3.25}$$

where the w_i is the word embeddings from WLM for i-th word and S_j is the j-th news representation.

Click Predictor

The click predictor module takes the word-level matching signal e_w from WLM and news-level matching signal e_n from NLM to generate the user click probability of each item. The prediction function is as follows,

$$y = \text{softmax}\left(\left[e_w; e_n\right] W^c + b^c\right) \tag{3.26}$$

In the UNBERT training, the pre-trained *bert-base-uncased* model weight is used directly to initialize the word-level module. Then, the entire model was fine-tuned using the MIND datasets – a real-world news recommendation dataset collected from MSN News logs.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

Since the UNBERT model has used a pre-trained BERT model as the foundation model, so it can capture some generalized knowledge outside the fine-tuning dataset. As a result, the UNBERT model has proved excellent performance on cold-start items. This strength is very beneficial for News recommender systems as there are tons of new news generated every day. Considering the importance of news freshness to the user, it is very important that the model can pick up new news items from the candidate pools and recommend to the relevant user in a timely way.

3.11.4 Inspiration of BERT Applications in Recommender Systems

The BERT model's application in recommender systems provides another efficient way of handling sequential input data in both text form and user behavior sequence form. The multihead self-attention mechanism lets the model capture the contexts from long-ranged surrounding items. Besides, it can provide the hidden representations of text-based features, which inherent the original use case of BERT model in NLP tasks.

3.12 LLM: The New Revolution in AI and Its Application in Recommender Systems

Since the introduction of ChatGPT by OpenAI in 2022, it has had a profound impact on the AI field. An example conversation with ChatGPT is shown in Figure 3.36. At the beginning of the conversation, the user provides an input text or instruction to initiate the request for information or assistance. This input text is usually referred to as a "prompt" in the ChatGPT context. ChatGPT will generate the response corresponding to the prompt provided by the user. As we can see in this example, ChatGPT presents astonishing capabilities of reasoning and understanding, as well as the ability to generate more human-like dialogs based on given contexts and questions.

The success of ChatGPT pushes Large Language Models (LLM) to the front of stage and attracts tremendous interest. The LLM is not just limited to the models that support ChatGPT; it is rather a general term to represent a bunch of different language models with large parameter size and based on some neural network structures (for example, the Transformer structure as introduced in Section 3.1). It is usually pretrained with massive text corpuses from the public sources such as articles, Wikipedia, books, and some other Q&A-type conversational data. From the pretraining step, LLMs can learn numerous generalized knowledge from the public domain and transfer the learnings to the downstream tasks. Sometimes, LLMs are also fine-tuned to let them pick the knowledge within specific domains to improve their performance on corresponding tasks. Thanks to LLMs' powerful performance, it has opened up new possibilities of using them in many different domains outside NLP fields.

One of the extensions is adapting the LLM to recommender systems for model performance and user experience improvements. In this section, we will follow a recently published literature survey [19] to explore "where" and "how" an LLM can be applied in recommender systems. The high-level scheme is illustrated in Figure 3.37.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

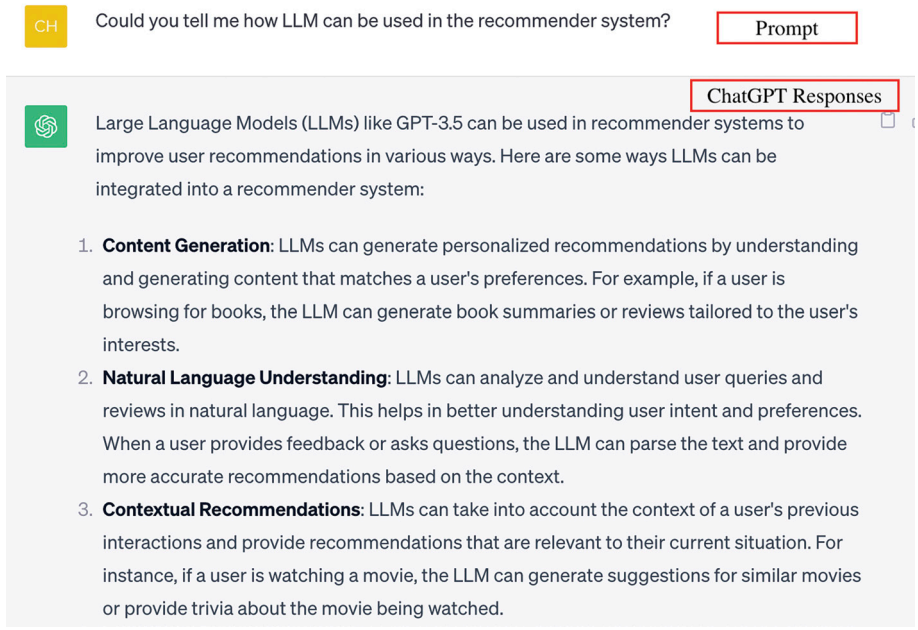

Figure 3.36 An example of a conversation with ChatGPT.

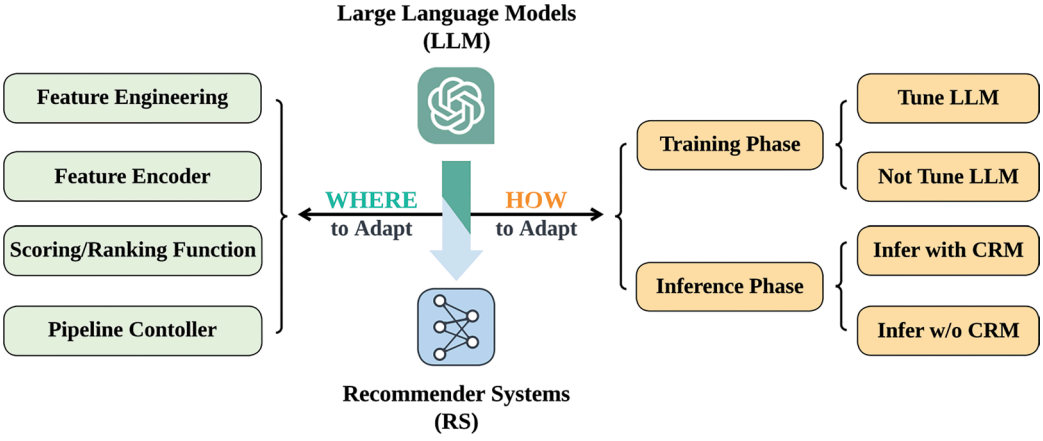

Figure 3.37 The decomposition of "where" and "how" to adapt the LLM in the recommender systems.

3.12.1 Where to Adapt LLM

In the survey, authors abstracted the following key components and elaborated on the applications of LLM in each of these key areas:

- Feature engineering
- Feature encoder

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

- Scoring/ranking function
- Pipeline Controller

3.12.1.1 LLM in Feature Engineering

In the feature engineering application, LLM mainly generates the auxiliary textual features based on original input data (for example, user profiles, item descriptions, and so on) to augment the input features. During the feature generation, the strength of LLM in reasoning, understanding, and summarization can be leveraged to make the generated text-based features more accurate and concise than the original output.

One example of a feature engineering application is the MINT framework introduced in [20]. MINT is an approach that targets the narrative-driven recommendation (NDR), where the user gives a verbose query including contextual information and requests, and the recommender system recommends the item based on the user's query and interacted item history. One challenge of NDR model training is that it always lacks training data. MINT is designed to mainly generate synthetic training data pairs utilizing InstructGPT for final retrieval model training. The synthetic data generation and model training are depicted in Figure 3.38.

In the MINT approach, authors used InstructGPT model to generate the narrative queries. The prompt examples are shown in Figure 3.39. The few-shot strategy was adopted in the query generation task with a few examples are provided in the prompt. It is expected that InstructGPT will follow the examples provided in the prompt, capture the relationship between different parts of the example, and then finally generate the synthetic queries to complete the target task. User's historical interacted items, past review and actual user narrative query are provided in each few-shot example. The intention is to let LLM capture the interests and preferences from past activities and then artificially generate the corresponding queries, mimicking what a user may ask in a query.

The synthetic queries are then collected and paired with all the interacted items. Since the synthetic queries may only capture part of the interests, not all the items are relevant to the synthetic queries. So the authors used a filter model to filter low-relevance pairs. Finally, these synthetically generated data pairs are then used as the training samples for a retrieval model based on a bi-encoder structure.

This work provides an example of how LLM is being used to generate the synthetic data or input features to help with the model training. Through this approach, the general

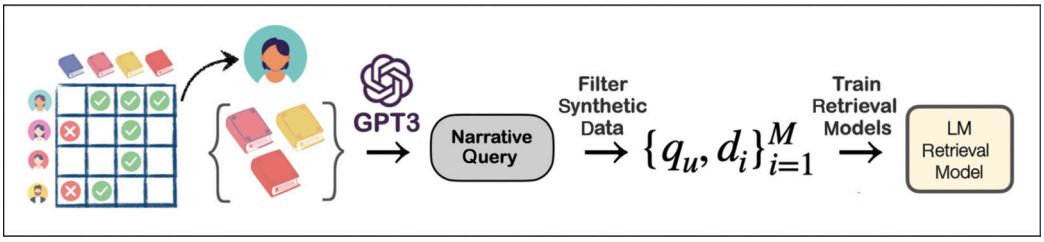

Figure 3.38 The illustration of the MINT approach to generate the narrative queries for set items liked by a user with an LLM. The generated queries will be paired with the item to form a training sample for an LM-based retrieval model.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

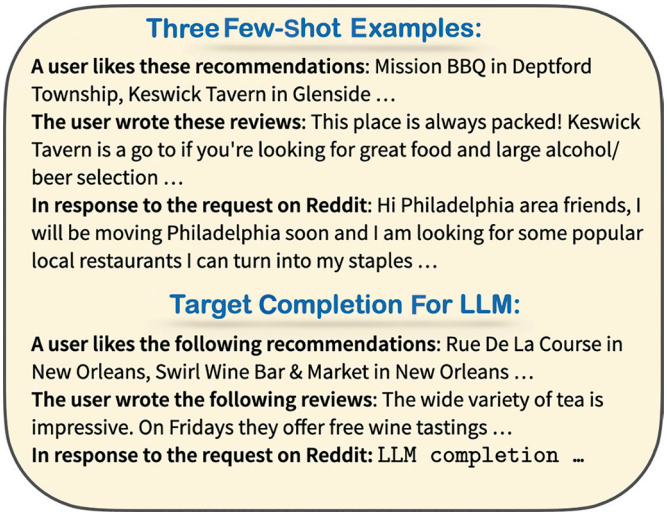

Figure 3.39 Prompts used in InstructGPT to generate narrative queries.

knowledge and reasoning abilities from the LLM can be carried over to text generation tasks, which can provide additional latent signals into the text input features.

3.12.1.2 LLM as Feature Encoder

In this application, LLM is used as a feature encoder to encode the textual features and use the encoded representations in the recommendation model. The benefits of using a LLM as a feature encoder are:

(1) Enriching the user or item representations with more semantic meanings.
(2) Transferring generalized knowledge from a pretrained LLM foundation model for cross-domain or cold start recommendations.

The UNBERT model introduced in Section 3.11.2 falls within this bucket. In the UNBERT model, a pre-trained BERT is adopted in the Word Level Module to encode the concatenated texts for target news and user-interacted news. Readers can refer to Section 3.11.2 for more details.

3.12.1.3 LLM as Scoring/Ranking Function

In this application, LLM is used to directly generate (1) the rating for the candidate item, (2) the recommendation list of the items, and (3) both rating and recommendation lists with a multitask setup.

In this section, we will briefly introduce one work by the Google team [21] and present an example of using LLM to finish the scoring task. In this work [21], authors explored the LLM's ability to generate ratings with zero-shot, few-shot, and fine-tuned settings. The task is to predict users' ratings based on their viewing history.

The prompt design for zero-shot and few-shot are presented in Figure 3.40 (a) and (b), respectively. In the zero-shot prompt, the user's interaction history is just listed down with the user rating, whereas a few rating prediction examples are included in

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

(a)

Zero-Shot User Rating Predictor

Given a user's past movie ratings in the format: Title, Genres, Rating. Ratings range from 1.0 to 5.0.

Babe (1995), Children's|Comedy|Drama, 4
There's Something About Mary (1998), Comedy, 4
Awakenings (1990), Drama, 4
Simple Plan, A (1998), Crime|Thriller, 4
Bug's Life, A (1998), Animation|Children's|Comedy, 5
Twelve Monkeys (1995), Drama|Sci-Fi, 5
Pleasantville (1998), Comedy, 4
Apollo 13 (1995), Drama, 4
Misery (1990), Horror, 4
South Park: Bigger, Longer and Uncut (1999), Animation|Comedy, 4

The candidate movie is Player, The (1992), Comedy|Drama. What's the rating that the user will give?

(b)

Few-Shot User Rating Predictor

You're required to predict user ratings for movie recommendations. The rating ranges from 1.0 to 5.0. You're given the user's past rating history, in the format of in the format: Title, Genres, Rating.

Example:
Q: The user history is:
Jerry Maguire (1996), Drama|Romance, 2.0
Fight Club (1999), Action|Crime|Drama|Thriller, 4.0
…
Ghostbusters (a.k.a. *Ghost Busters*) (1984), Action|Comedy|Sci-Fim 4.0
The user now watched (Fifth Element, Sci-fi|Action), give a single number as rating without saying anything else. Do not give reasoning.
A: 4.0

Q: The user history is:
Babe (1995), Children's|Comedy|Drama, 4
There's Something About Mary (1998), Comedy, 4
…
Misery (1990), Horror, 4
South Park: Bigger, Longer and Uncut (1999), Animation|Comedy, 4
The user now watched (Player, The (1992), Comedy|Drama), give a single number as rating without saying anything else. Do not give reasoning.
A:

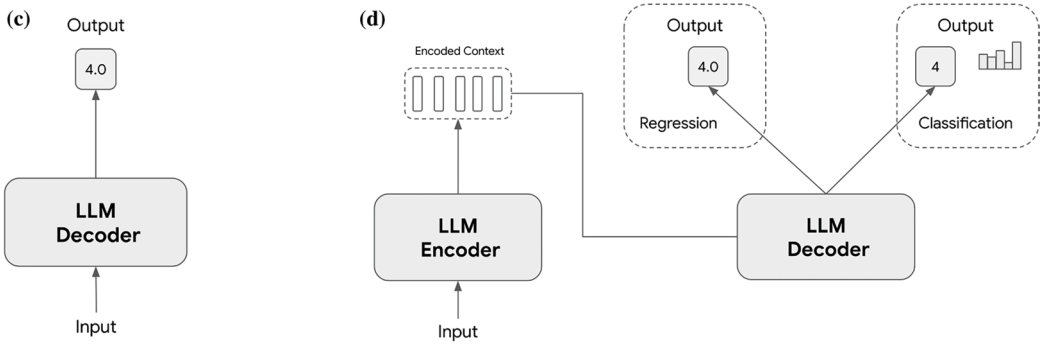

Figure 3.40 (a) A zero-shot example; (b) a few-shot example; (c) a decoder-only fine-tuning model; (d) an encoder-decoder-based fine-tuning model for the rating task.

the prompt in the few-shot setting. Throughout the experiment, the authors found LLM can be very sensitive to the provided prompt and doesn't always follow instructions.

Then, the authors adopted fine-tuning and fine-tuned several models in the Flan-T5 model families. In both decoder-only (Figure 3.40(c)) and encoder-decoder models (Figure 3.40(d)), a projection layer is added to generate the output for either the classification task or the regression task. Then, the model is fine-tuned with training samples to better fit the prediction task.

Through the experiments, the authors concluded that zero-shot and few-shot LLM approaches have lower performance than the fully supervised methods. Fine-tuned LLMs can help close the gap and bring benefits in (1) higher training data efficiency (that is, smaller training data size is needed), (2) much easier feature processing and modeling, and (3) new capabilities expansion with conversational recommendations.

3.12.1.4 LLM as Pipeline Controller

It has been proven that LLM doesn't only understand textual information, but also presents strong capabilities for in-context learning and logical reasoning. As a result, it could also play a role as a controller to decide where logic flow should go in the entire recommendation pipeline.

In a recent work [22], the CHAT-REC system was introduced to bridge the recommender systems and LLMs. This system consists of several key components: the prompt constructor, the LLM (ChatGPT), and the conventional recommender system. The workflow of the CHAT-REC is depicted in Figure 3.41. We will use a pseudo example to illustrate how the system works:

(1) The user sent a query, "Could you recommend some action movies to me?"
(2) The **prompt constructor** collects different contexts to generate the prompt. The contexts' sources include user raw query, recommender system interfaces, user profile, user-item history, and dialog history.
(3) The generated prompt is then fed into the **LLM (ChatGPT)** module to generate the output. In the output, ChatGPT will decide if the conventional **recommender system** will be called.
(4) In the first pass, the **LLM** decides to call the **recommender system** to generate the candidate set. Then the recommender system will generate the candidate sets and send them back to the **prompt constructor**.
(5) The **prompt constructor** then generates the new prompt with the recommended candidate set, and sends it to the **LLM module**.
(6) The **LLM** decides that no recommendation call is needed, and then conducts the reranking to pick the top five candidates to return to the user.
(7) If he user asks for an explanation, the user query and other contexts will repeat the process from Step (1). The **LLM module** can determine that no recommendation call is needed and generate the responses to the user directly.

From this example workflow, we can see that the LLM model is being used as an orchestrator in the entire system by leveraging its excellent ability of reasoning and understanding and drives the interaction between the user and recommender systems.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

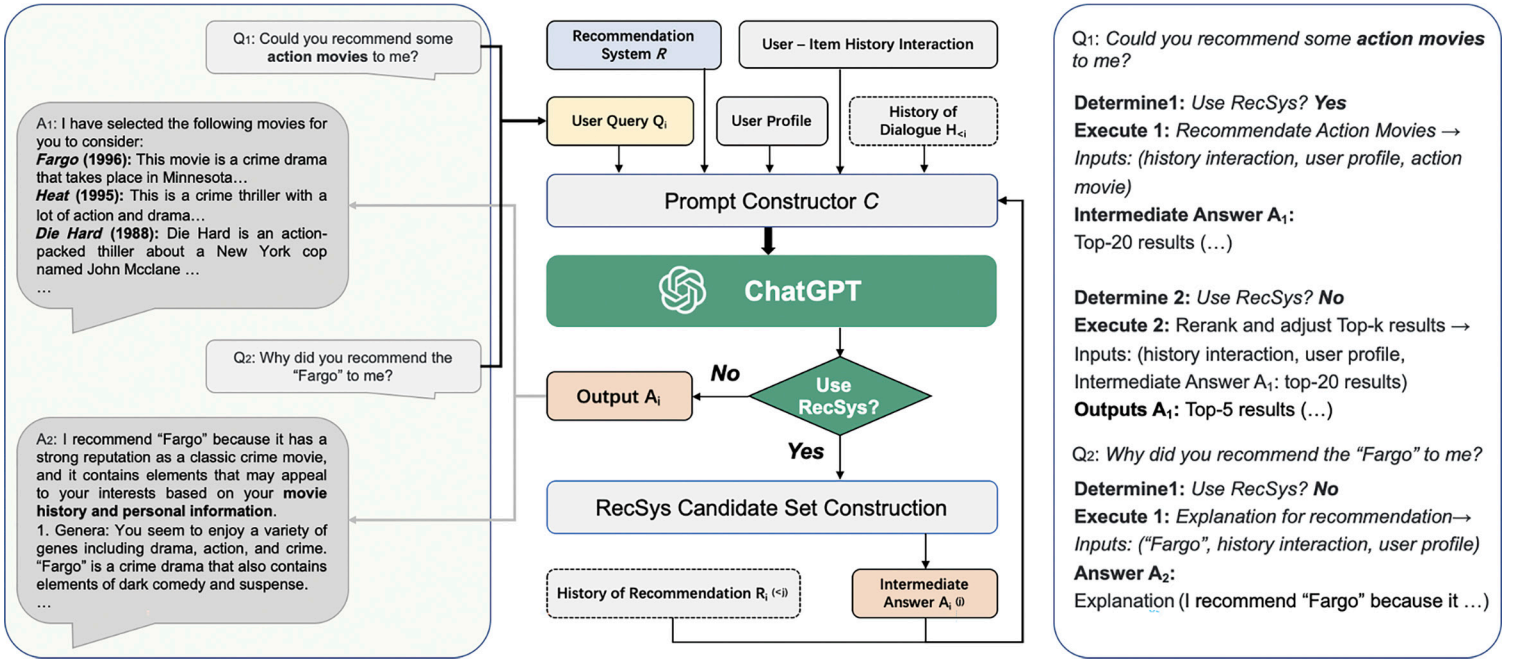

Figure 3.41 Overview of CHAT-REC. The left side is the dialog used in the communication with ChatGPT. The middle shows the flowchart, and the right side shows the specific judgment in the flow [22].

3.12.2 How to Adapt LLM

In this survey paper [19], the authors divided the usage of LLM in recommender systems into four quadrants, as shown in Figure 3.42. The four quadrants were:

(1) Tuned LLM
(2) Not tuned LLM
(3) Infer with conventional recommendation model (CRM)
(4) Not infer with conventional recommendation model (CRM)

In Figure 3.42, we can see the overall research development trajectory starts from tuned-LLM + infer with CRM and firstly moves toward the not-tuned LLM + infer w/o CRM quadrant. In this trajectory, the model size is significantly increased. As there is no model training in this quadrant, it is very fast for the model development, but the performance is sacrificed as a trade-off consequence. Then, the researchers start to diverge in both directions to two other quadrants – not tuned LLM + CRM and tuned LLM + w/o CRM. The main motivations are to achieve better model performance as well as reduce the model size for faster training and inferencing.

As we have introduced one model in each quadrant in the previous section to show the adaptations of LLM with recommender systems, we will not expand to cover the other models here. Interested readers can follow the references from this survey paper [19] to continue exploring this new tide of revolution.

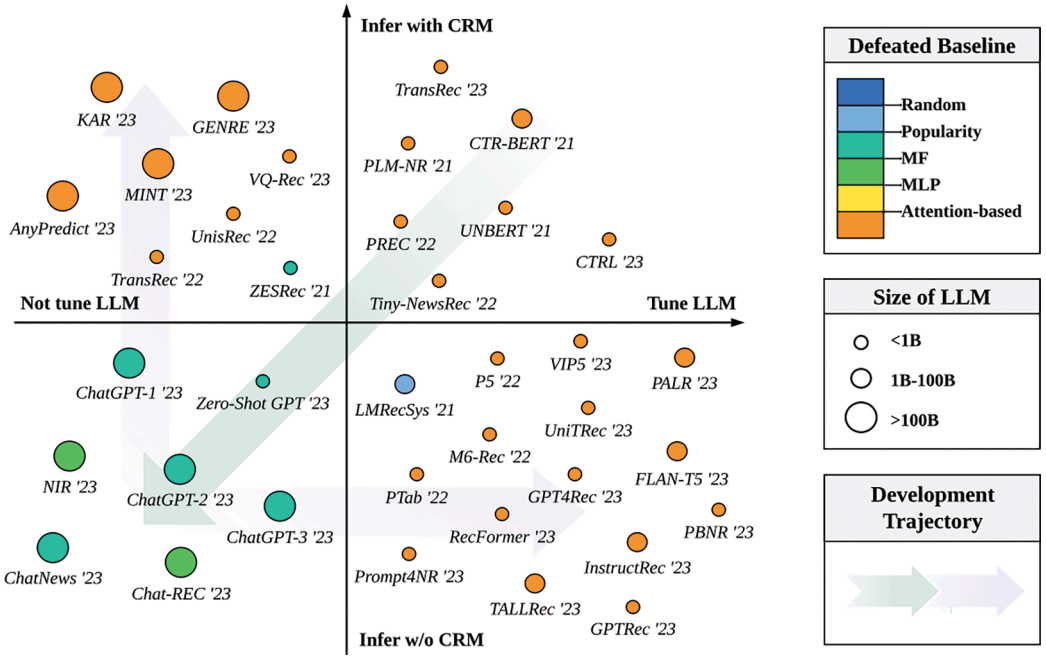

Figure 3.42 Four quadrant classification about how to adapt LLM to recommender systems. Circle size denotes the model size and the colors indicate the best benchmarking model that each model can beat. The light-colored arrows show the overall development trajectory.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

3.12.3 Inspiration and Challenges of LLM Adaptation in Recommender Systems

The recent developments of LLMs have not only attracted the world's attention to the AI field, but also opened a new "gate" for recommender systems. The LLMs' astonishing understanding and reasoning abilities give us a new angle on building recommender systems, and also add a new powerful tool to our toolbox. However, we also need to acknowledge the challenges that we are facing in the LLM world.

At the end of the survey [19], the authors summarized the challenges from three aspects: (1) efficiency, (2) effectiveness, and (3) ethics:

(1) **Efficiency:** This includes both training and inference latency. As the model becomes bigger, it requires more training data to train the model effectively. Both the larger model size and larger training data can significantly increase the training efficiency. Also, the increased model parameter amount makes it challenging to finish the inferencing task under the limited time constraints in the online service.
(2) **Effectiveness:** Even though many researchers have demonstrated the powerfulness of the LLMs, still the LLM can have its own shortcomings and limitations. Two examples are limited context window size and ID feature understanding. From past studies, we can see LLMs show a reduction of understanding ability when the input texts are too long in the prompt. For the other limitation, since the ID features are not semantically meaningful, so it will be quite hard for the LLMs to understand and differentiate the IDs in the model input.
(3) **Ethics:** This is a quite common topic in recommender systems. The practitioners in the recommendation field have been studying many approaches for removing bias from recommender systems. It has also been found that LLMs can present certain biases originating from the pre-training corpus and could potentially generate harmful or offensive content.

Luckily, numerous researchers and engineers have been working on each aspect of these challenges and to create solutions to solve them. The reader can refer to the references in the survey to get more details about those solutions, to inspire research and actual implementations.

3.13 Summary: The Deep Learning Era of Recommender Systems

This section describes the relevant knowledge of state-of-the-art deep learning recommendation models, echoing the evolution diagram of deep learning models at the beginning of the chapter. In this section, we will summarize the key knowledge of deep learning recommendation models (as shown in Table 3.2).

With so many deep learning recommendation model options, the premise for readers not to get lost is to be familiar with the relationship between each model and its applicable scenarios. It needs to be clear that in the era of deep learning, no specific model can be competent for all business scenarios, and it can be seen from Table 3.2 that the characteristics of each model are different.

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

Table 3.2 Key points of deep learning recommendation models

Model Name	Mechanisms	Characteristics	Limitations
AutoRec	Based on the auto-encoder, encode users or items, and use the generalization ability of the auto-encoder to make recommendations	The single hidden layer neural network has a simple structure, enabling fast training and deployment	Limited expressivity
Deep Crossing	Utilizing the classic deep learning framework of "Embedding layer + multihidden layer + output layer," automatically finish the deep crossover of the features	Classic deep learning recommendation model framework	Use fully connected hidden layers for feature crossing, lacks specificity
NeuralCF	Replace the dot product operation of the user vector and the item vector in the traditional matrix factorization with the interoperation of the neural network	Expressive enhanced version of matrix factorization model	Only the ID features of users and items are used, and no other features are added
PNN	For cross operations between different feature domains, define multiple product operations such as "inner product" and "outer product"	Improving the feature crossover on the top of classic deep learning framework	The "outer product" operation is approximated, which affects its expressivity to a certain extent.
Wide&Deep	Use the wide part to strengthen the "memorization" of the model, and use the deep part to strengthen the "generalization" of the model	Pioneered the construction method of the ensembled model, which has a significant impact on the subsequent development of the deep learning recommendation model	The wide part requires manual feature cross selection
Deep and Cross	Replacing the wide part in the Wide&Deep model with a cross network	Solved the problem of manual feature interaction in the Wide&Deep model	The complexity of the feature cross network is high
FNN	Use the parameters of FM to initialize the parameters of the embedding layer of the deep neural network	Use FM to initialize the parameters to speed up the convergence of the entire network	The main structure of the model is relatively simple, and there is no objective-oriented feature crossover layer
DeepFM	On the basis of Wide&Deep model, replace the original linear wide part with FM	Enhanced the feature interactions of the wide part	No significant structural difference with the classic Wide&Deep model
NFM	Replace the operation of second-order hidden vector crossover in FM with a neural network	Compared with FM, NFM has stronger expressivity and feature intersection ability	Very similar to the structure of the PNN model

(continued)

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

Table 3.2 (cont.)

Model Name	Mechanisms	Characteristics	Limitations
AFM	On the basis of FM, an attention score is added to each crossed result after the second-order hidden vector cross, and the attention score is learned through the attention network	Different crossed features have different importance	The training process of the attention network is complicated
DIN	Based on the traditional deep learning recommendation model, an attention mechanism is introduced. The attention score is calculated by using the correlation between user behavior history items and target advertising items	Make more targeted recommendations given different advertising items	Not take advantage of the other features other than "historical behavior"
DIEN	Combine the sequence model with the deep learning recommendation model, and use the sequence model to simulate the evolution process of users' interests	The sequence model enhances the system's ability to express the changes of user interests, so that the recommender system begins to consider the valuable information in the time-related behavior sequences	The training of the sequence model is complicated, and the latency of the online inferencing is relatively large. It requires engineering optimization in production.
DRN	Apply the idea of reinforcement learning to the recommender system, and conduct online real-time learning and updating of the recommendation model	The ability of the model to utilize the real-time data is greatly enhanced	The online inferencing is more complicated, and the engineering implementation is more difficult
BERT4Rec and UNBERT	The applications of BERT model in recommender systems provide an efficient way to handle the sequential input data in both text form and user behavior sequence form	The multihead self-attention mechanism lets the model capture the contexts from long-ranged surrounding items. Besides, it can provide the hidden representations of text-based features, which inherent the original use case of BERT model in NLP tasks	The model complexity and online serving resources is much higher than other recommendation model
LLM	Rebuild recommender system with LLM	The LLMs' astonishing understanding and reasoning abilities give us a new angle to build the recommender system, and also add a new powerful tool to our toolbox	It's a totally new domain to combine LLM with recommender system. There are still lots of new challenges that we are facing in the LLM world

https://doi.org/10.1017/9781009447515.005 Published online by Cambridge University Press

For this reason, this chapter does not list any model performance benchmarking, because it is impossible to form authoritative test results with different datasets, different application scenarios, different evaluation methods and evaluation indicators. In the actual application process, it is also necessary for the engineers to select the most suitable deep learning recommendation model after sufficient parameter tuning and comparison based on their own business data.

The deep learning recommendation model has never stopped its development. From Alibaba's multimodal and multiobjective deep learning model, to YouTube's session-based recommendation model, to the LLM revolution, the deep learning recommendation model not only evolves faster and faster, but also has been applied to wider application scenarios. The following chapters introduce the application of deep learning models in recommender systems from different perspectives. We also hope that readers will continue their exploration into the latest development of deep learning recommendation models based on the knowledge introduced in this chapter.

References

[1] Suvash Sedhain, et al. Autorec: Autoencoders meet collaborative filtering. Proceedings of the 24th International Conference on World Wide Web, May 18, 2015 (pp. 111–112).

[2] Ying Shan, et al. Deep crossing: Web-scale modeling without manually crafted combinatorial features. Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, August 13, 2016 (pp. 255–262).

[3] Kaiming He, et al. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, June 27–30, 2016 (pp. 770–778).

[4] Xiangnan He, et al. Neural collaborative filtering. Proceedings of the 26th international conference on world wide web. International World Wide Web Conferences Steering Committee, April 3, 2017 (pp. 173–182).

[5] Yanru Qu, et al. Product-based neural networks for user response prediction. IEEE 16th International Conference on Data Mining (ICDM), December 12, 2016 (pp. 1149–1154).

[6] Heng-Tze Cheng, et al. Wide & deep learning for recommender systems. Proceedings of the 1st workshop on deep learning for recommender systems, September 15, 2016 (pp. 7–10).

[7] Ruoxi Wang, et al. Deep & cross network for ad click predictions. Proceedings of the ADKDD'17, August 14, 2017 (pp. 1–7).

[8] Weinan Zhang, Tianming Du, Jun Wang. Deep learning over multi-field categorical data – a case study on user response prediction. Advances in Information Retrieval: 38th European Conference on Information Retrieval, March 20–23, 2016 (pp. 45–57).

[9] Huifeng Guo, et al. DeepFM: A factorization-machine based neural network for CTR prediction: arXiv preprint arXiv:1703.04247 (2017).

[10] Xiangnan He, Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, August 7, 2017 (pp. 355–364).

[11] Jun Xiao, et al. Attentional factorization machines: Learning the weight of feature interactions via attention networks: arXiv preprint arXiv: 1708.04617 (2017).

[12] Guorui Zhou, et al. Deep interest network for click-through rate prediction. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, July 19, 2018 (pp. 1059–1068).
[13] Guorui Zhou, et al. Deep interest evolution network for click-through rate prediction. Proceedings of the AAAI Conference on Artificial Intelligence, 33(1), 2019: 5941–5948.
[14] Guanjie Zheng, et al. DRN: A deep reinforcement learning framework for news Recommender. Proceedings of the 2018 World Wide Web Conference. International World Wide Web Conferences Steering Committee, April 23, 2018 (pp. 167–176).
[15] Jacob Devlin, et al. Bert: Pre-training of deep bidirectional transformers for language understanding: arXiv preprint arXiv:1810.04805 (2018).
[16] Fei Sun, et al. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. Proceedings of the 28th ACM International Conference on Information and Knowledge Management, November 3, 2019 (pp. 1441–1450).
[17] Qi Zhang, et al. UNBERT: User-news matching BERT for news recommendation. *IJCAI*, 21, 2021: 3356–3362.
[18] Ashish Vaswani, et al. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.
[19] Jianghao Lin, et al. How can recommender systems benefit from large language models: A survey: arXiv preprint arXiv:2306.05817 (2023).
[20] Sheshera Mysore, Andrew McCallum, Hamed Zamani. Large language model augmented narrative driven recommendations. Proceedings of the 17th ACM Conference on Recommender Systems, September 14, 2023 (pp. 777–783).
[21] Wang-Cheng Kang, et al. Do LLMs understand user preferences? Evaluating LLMs on user rating prediction: arXiv preprint arXiv:2305.06474 (2023).
[22] Yunfan Gao, et al. Chat-rec: Towards interactive and explainable LLMs-augmented recommender system: arXiv preprint arXiv:2303.14524 (2023).