Deep Neural Network Models

The previous section showed you how to use matrix factorization to learn embeddings. Some limitations of matrix factorization include:

- The difficulty of using side features (that is, any features beyond the query ID/item ID). As a result, the model can only be queried with a user or item present in the training set.
- Relevance of recommendations. As you saw in the first Colab (/machine-learning/recommendation/labs/movie-rec-programming-exercise), popular items tend to be recommended for everyone, especially when using dot product as a similarity measure. It is better to capture specific user interests.

Deep neural network (DNN) models can address these limitations of matrix factorization. DNNs can easily incorporate query features and item features (due to the flexibility of the input layer of the network), which can help capture the specific interests of a user and improve the relevance of recommendations.

Softmax DNN for Recommendation

One possible DNN model is softmax (/machine-learning/glossary#softmax), which treats the problem as a multiclass prediction problem in which:

- The input is the user query.
- The output is a probability vector with size equal to the number of items in the corpus, representing the probability to interact with each item; for example, the probability to click on or watch a YouTube video.

Input

The input to a DNN can include:

- dense features (for example, watch time and time since last watch)
- sparse features (for example, watch history and country)

Unlike the matrix factorization approach, you can add side features such as age or country. We'll denote the input vector by x.

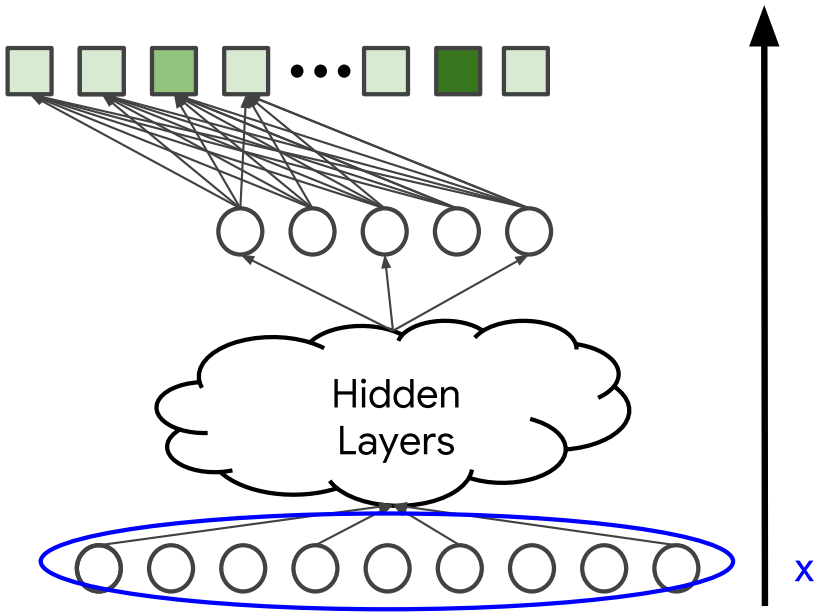

Figure 1. The input layer, x.

Model Architecture

The model architecture determines the complexity and expressivity of the model. By adding hidden layers and non-linear activation functions (for example, ReLU), the model can capture more complex relationships in the data. However, increasing the number of parameters also typically makes the model harder to train and more expensive to serve. We will denote the output of the last hidden layer by $\psi(x) \in \mathbb{R}^d$.

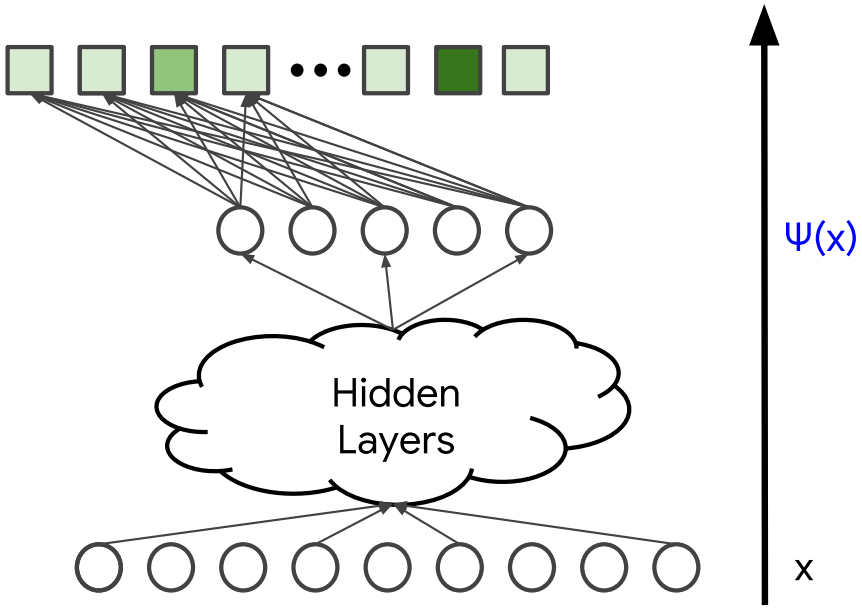

Figure 2. The output of the hidden layers, $\psi(x)$.

Softmax Output: Predicted Probability Distribution

The model maps the output of the last layer, $\psi(x)$, through a softmax layer to a probability distribution $\hat{p} = h(\psi(x)V^T)$, where:

- $h : \mathbb{R}^n \to \mathbb{R}^n$ is the softmax function, given by $h(y)_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$
- $V \in \mathbb{R}^{n \times d}$ is the matrix of weights of the softmax layer.

The softmax layer maps a vector of scores $y \in \mathbb{R}^n$ (sometimes called the **logits** (https://developers.google.com/machine-learning/glossary/#logits)) to a probability distribution.

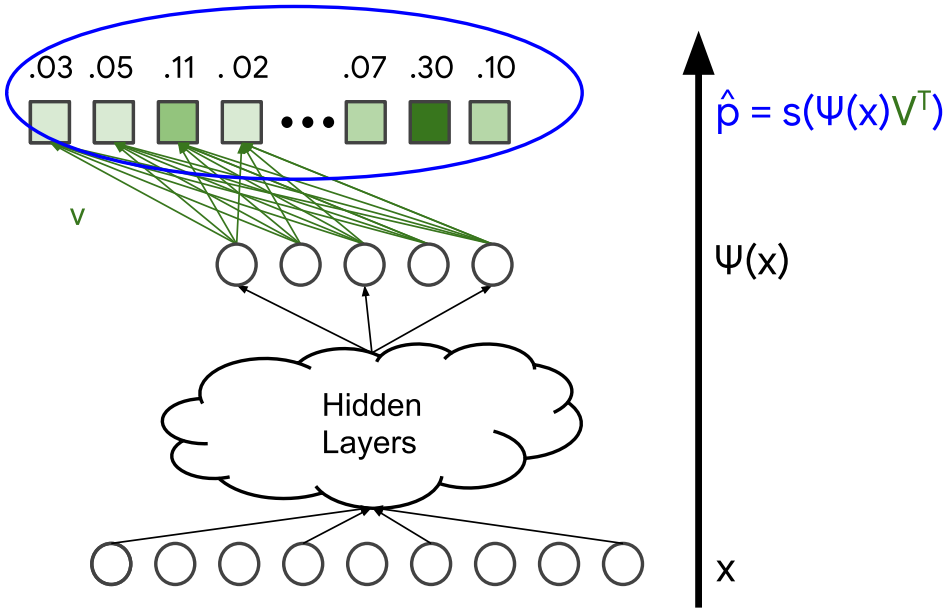

Figure 3. The predicted probability distribution, $\hat{p} = h(\psi(x)V^T)$.

u know?

ıme softmax is a play on words. A "hard" max assigns probability 1 to the item with the largest score y_i. B
st, the softmax assigns a non-zero probability to all items, giving a higher probability to items that have hi
:. When the scores are scaled, the softmax $h(\alpha y)$ converges to a "hard" max in the limit $\alpha \to \infty$.

Loss Function

Finally, define a loss function that compares the following:

- $\hat{p}$, the output of the softmax layer (a probability distribution)
- p, the ground truth, representing the items the user has interacted with (for example, YouTube videos the user clicked or watched). This can be represented as a

normalized multi-hot distribution (a probability vector).

For example, you can use the cross-entropy loss since you are comparing two probability distributions.

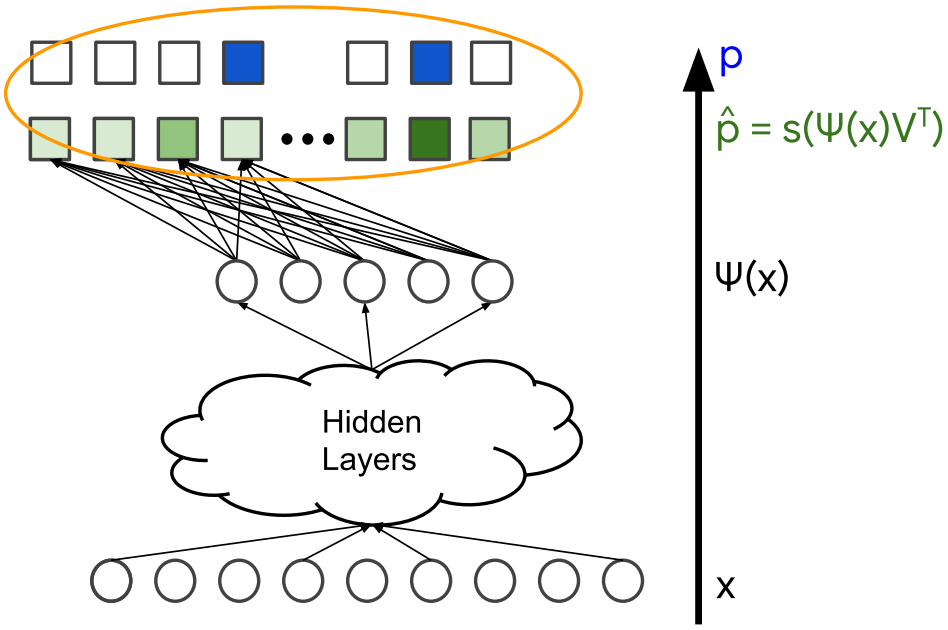

Figure 4. The loss function.

Softmax Embeddings

The probability of item j is given by $\hat{p}_j = \frac{\exp(\langle \psi(x), V_j \rangle)}{Z}$, where Z is a normalization constant that does not depend on j.

In other words, $\log(\hat{p}_j) = \langle \psi(x), V_j \rangle - log(Z)$, so the log probability of an item j is (up to an additive constant) the dot product of two d-dimensional vectors, which can be interpreted as query and item embeddings:

- $\psi(x) \in \mathbb{R}^d$ is the output of the last hidden layer. We call it the embedding of the query x.
- $V_j \in \mathbb{R}^d$ is the vector of weights connecting the last hidden layer to output j. We call it the embedding of item j.

Since $\log$ is an increasing function, items j with the highest probability $\hat{p}_j$ are the items with the highest c
ct $\langle \psi(x), V_j \rangle$. Therefore, the dot product can be interpreted as a similarity measure in this embedding sp

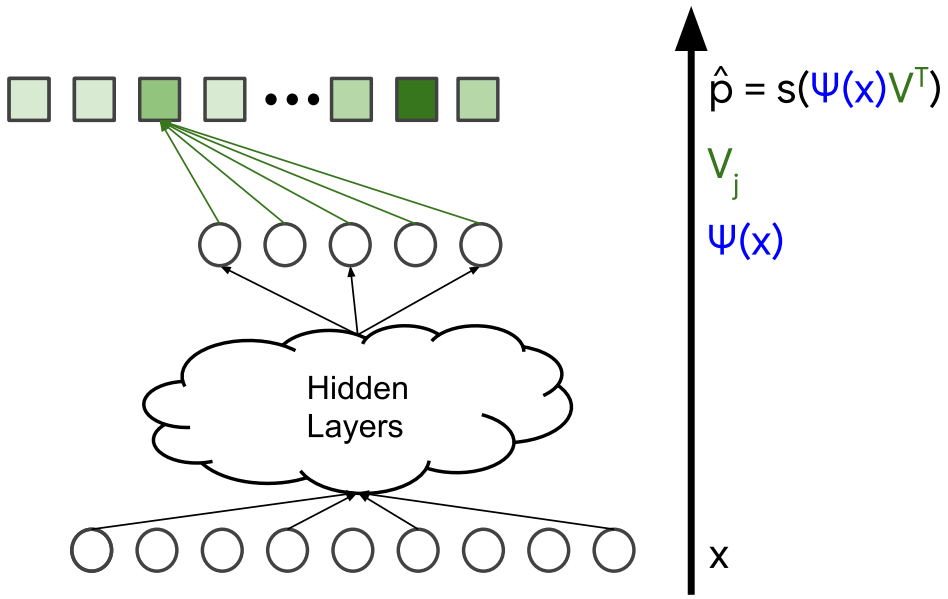

Figure 5. Embedding of item j, $V_j \in \mathbb{R}^d$

DNN and Matrix Factorization

In both the softmax model and the matrix factorization model, the system learns one embedding vector V_j per item j. What we called the *item embedding matrix* $V \in \mathbb{R}^{n \times d}$ in matrix factorization is now the matrix of weights of the softmax layer.

The query embeddings, however, are different. Instead of learning one embedding U_i per query i, the system learns a mapping from the query feature x to an embedding $\psi(x) \in \mathbb{R}^d$. Therefore, you can think of this DNN model as a generalization of matrix factorization, in which you replace the query side by a nonlinear function $\psi(\cdot)$.

Can You Use Item Features?

Can you apply the same idea to the item side? That is, instead of learning one embedding per item, can the model learn a nonlinear function that maps item features to an embedding? Yes. To do so, use a two-tower neural network, which consists of two neural networks:

- One neural network maps query features x_{query} to query embedding $\psi(x_{\text{query}}) \in \mathbb{R}^d$
- One neural network maps item features x_{item} to item embedding $\phi(x_{\text{item}}) \in \mathbb{R}^d$

The output of the model can be defined as the dot product of $\langle \psi(x_{\text{query}}), \phi(x_{\text{item}}) \rangle$. Note that this is not a softmax model anymore. The new model predicts one value per pair $(x_{\text{query}}, x_{\text{item}})$ instead of a probability vector for each query x_{query}.

Previous
← Movie Recommendation System Exercise (/machine-learning/recommendation/labs/movie-rec-programming-exercise)

Next
Softmax Training → (/machine-learning/recommendation/dnn/training)