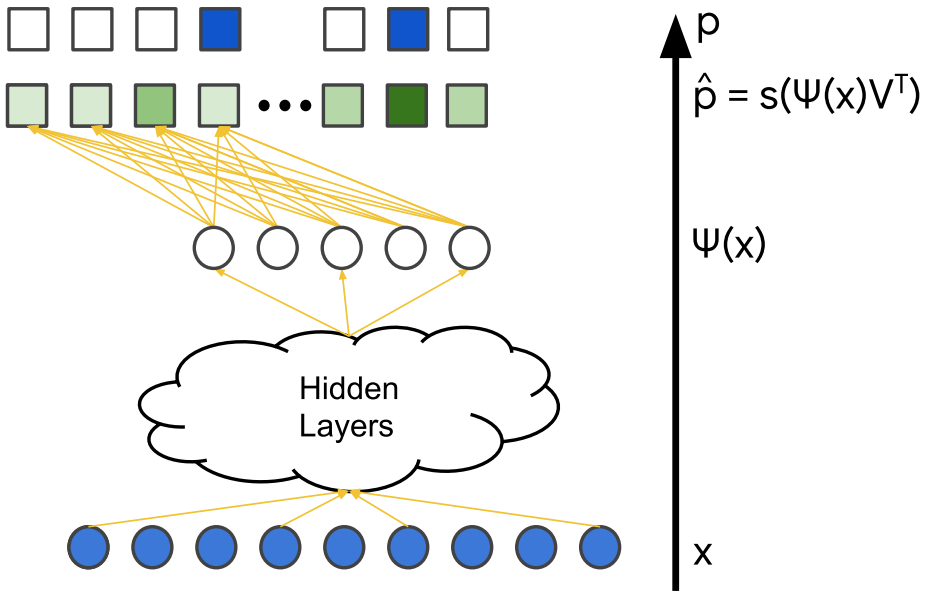
Softmax Training

The previous page explained how to incorporate a softmax layer into a deep neural network for a recommendation system. This page takes a closer look at the training data for this system.

Training Data

The softmax training data consists of the query features $\boldsymbol{x}$ and a vector of items the user interacted with (represented as a probability distribution $\boldsymbol{p}$). These are marked in blue in the following figure. The variables of the model are the weights in the different layers. These are marked as orange in the following figure. The model is typically trained using any variant of stochastic gradient descent.

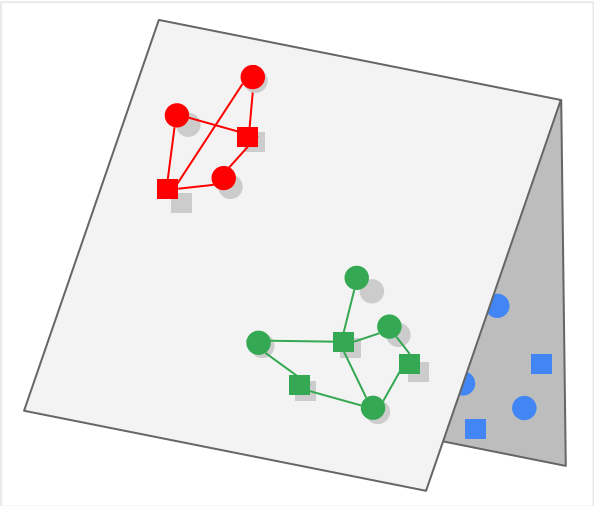


Negative Sampling

Since the loss function compares two probability vectors $\boldsymbol{p}, \hat{\boldsymbol{p}}(\boldsymbol{x}) \in \mathbb{R}^n$ (the ground truth and the output of the model, respectively), computing the gradient of the loss (for a single query $\boldsymbol{x}$) can be prohibitively expensive if the corpus size $\boldsymbol{n}$ is too big.

You could set up a system to compute gradients only on the positive items (items that are active in the ground truth vector). However, if the system only trains on positive pairs, the model may suffer from folding, as explained below.

Folding



In the following figure, assume that each color represents a different category of queries and items. Each query (represented as a square) only mostly interacts with the items (represented as a circle) of the same color. For example, consider each category to be a different language in YouTube. A typical user will mostly interact with videos of one given language.

The model may learn how to place the query/item embeddings of a given color relative to each other (correctly capturing similarity within that color), but embeddings from different colors may end up in the same region of the embedding space, by chance. This phenomenon, known as **folding**, can lead to spurious recommendations: at query time, the model may incorrectly predict a high score for an item from a different group.

Negative examples are items labeled "irrelevant" to a given query. Showing the model negative examples during training teaches the model that embeddings of different groups should be pushed away from each other.

Instead of using all items to compute the gradient (which can be too expensive) or using only positive items (which makes the model prone to folding), you can use negative sampling. More precisely, you compute an approximate gradient, using the following items:

- All positive items (the ones that appear in the target label)
- A sample of negative items (j in $1, \ldots, n$)

There are different strategies for sampling negatives:

- You can sample uniformly.
- You can give higher probability to items j with higher score $\psi(x).V_j$. Intuitively, these are examples that contribute the most to the gradient); these examples are often called hard negatives.

Resources:

For a more comprehensive account of the technology, architecture, and models used in YouTube, see Dee Neural Networks for YouTube Recommendations (https://research.google.com/pubs/pub45530.html).

See Xin et al., Folding: Why Good Models Sometimes Make Spurious Recommendations (https://dl.acm.org/citation.cfm?id=3109911) for more details on folding.

To learn more about negative sampling, see Bengio and Senecal, Adaptive Importance Sampling to Accel Training of a Neural Probabilistic Language Model. (https://ieeexplore.ieee.org/document/4443871/)

On Matrix Factorization Vs. Softmax

DNN models solve many limitations of Matrix Factorization, but are typically more expensive to train and query. The table below summarizes some of the important differences between the two models.

	Matrix Factorization	Softmax DNN
Query features	🚫 Not easy to include.	✓ Can be included.
Cold start	🚫 Does not easily handle out-of vocab queries or items. Some heuristics can be used (for example, for a new query, average embeddings of similar queries).	✓ Easily handles new queries.
Folding	✓ Folding can be easily reduced by adjusting the unobserved weight in WALS.	🚫 Prone to folding. Need to use techniques such as negative sampling or gravity.
Training scalability	✓ Easily scalable to very large corpora (perhaps hundreds of millions items or more), but only if the input matrix is sparse.	🚫 Harder to scale to very large corpora. Some techniques can be used, such as hashing, negative sampling, etc.
Serving scalability	✓ Embeddings U, V are static, and a set of candidates can be pre-computed and stored.	✓ Item embeddings V are static and can be stored. 🚫 The query embedding usually needs to be computed at query time, making the model more expensive to serve.

In summary:

- Matrix factorization is usually the better choice for large corpora. It is easier to scale, cheaper to query, and less prone to folding.
- DNN models can better capture personalized preferences, but are harder to train and more expensive to query. DNN models are preferable to matrix factorization for scoring because DNN models can use more features to better capture relevance. Also, it is usually acceptable for DNN models to fold, since you mostly care about ranking a pre-filtered set of candidates assumed to be relevant.

Previous
← Softmax Model (/machine-learning/recommendation/dnn/softmax)

Next
Retrieval (/machine-learning/recommendation/dnn/retrieval) →

Last updated 2020-02-10 UTC.