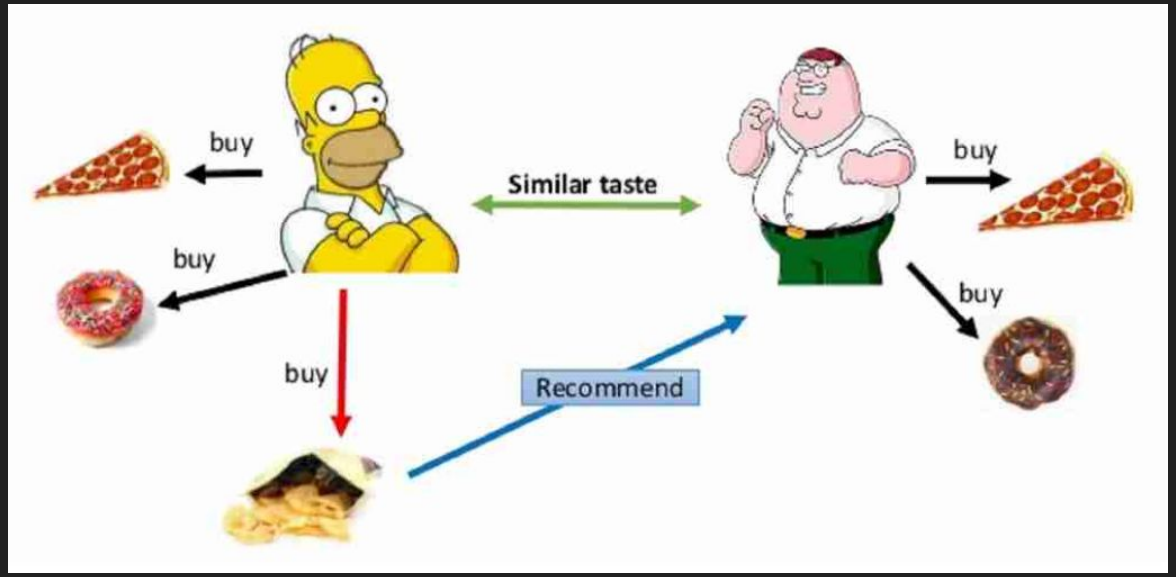
Variational Autoencoders for Collaborative Filtering

2021.04.08

Saehee Jeon

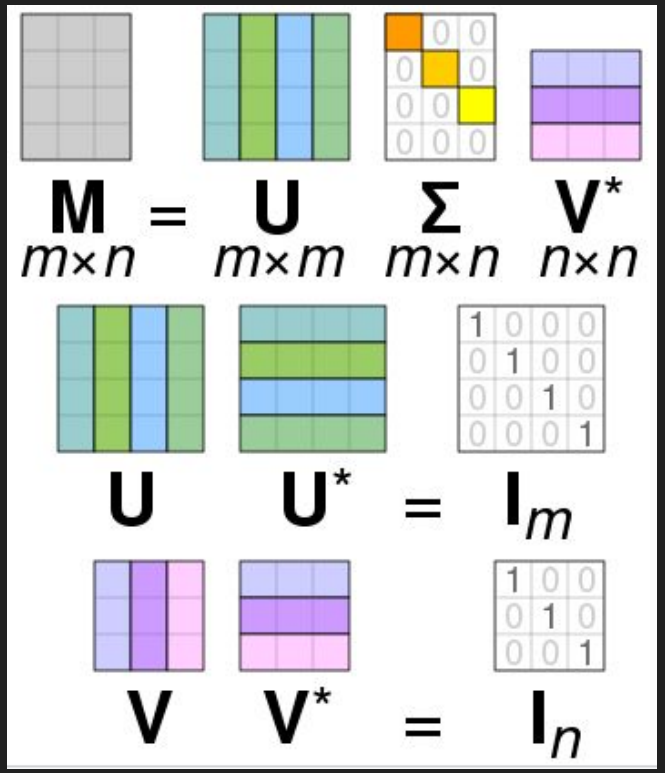
1. Introduction

What is Collaborative Filtering?



Collaborative filtering predicts what items a user will prefer by discovering and exploiting the similarity patterns across users and items

Latent factor models



- 👉 Linear
- 👉 Need Nonlinear features!
- 👉 Use Neural Networks

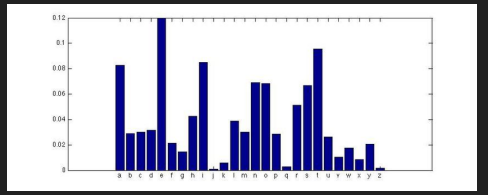
VAEs (Variational Auto Encoders)

Generalization of linear latent-factor models
👉 can explore non-linear probabilistic latent-variable models.

Although VAEs have been extensively studied for image fields, there is surprisingly little work to RecSys.

🌟 **Two adjustments** to get SOTA results with VAEs.

- Use a **multinomial likelihood** for the data distribution.
  (simple but essential)
  👉 outperforms Gaussian and logistic likelihoods



- Reinterpret and adjust the standard VAE objective

2. Method

Notation

u : index of users, u = 1,2,...,U

i : index of items, i = 1,2,...,I

x_u : a bag-of-words vector with the number of clicks for each item for user u (dim : I)

1) Model

Generative process : Similar to the deep latent Gaussian model

For each user u, the model starts by sampling a K-dim latent representation z_u from a standard Gaussian prior.

$$\mathbf{z}_u \sim \mathcal{N}(0, \mathbf{I}_K), \quad \pi(\mathbf{z}_u) \propto \exp\{f_\theta(\mathbf{z}_u)\},$$
$$\mathbf{x}_u \sim \text{Mult}(N_u, \pi(\mathbf{z}_u)).$$

softmax

$$\mathbf{z}_u \sim \mathcal{N}(0, \mathbf{I}_K), \quad \pi(\mathbf{z}_u) \propto \exp\{f_\theta(\mathbf{z}_u)\},$$
$$\mathbf{x}_u \sim \text{Mult}(N_u, \pi(\mathbf{z}_u)).$$

Non-linear function $f_\theta(\cdot)$: a multilayer perceptron with parameter \theta.

- N_u : Total number of clicks $N_u = \sum_i x_{ui}$ from user u
- x_u : the observed bag-of-words vector(sampled from a multinomial distribution)

The log-likelihood for user u (conditioned on the latent representation) is:

$$\log p_\theta(\mathbf{x}_u \mid \mathbf{z}_u) \stackrel{c}{=} \sum_i x_{ui} \log \pi_i(\mathbf{z}_u). \quad (2)$$

We present two popular choices of likelihood functions used in latent-factor collaborative filtering: Gaussian and logistic likelihoods

Define $f_\theta(\mathbf{z}_u) \equiv [f_{u1}, \ldots, f_{uI}]^\top$: output of the generative function

The Gaussian log-likelihood for user u is

$$\log p_\theta(\mathbf{x}_u \mid \mathbf{z}_u) \stackrel{c}{=} -\sum_i \frac{c_{ui}}{2}(x_{ui} - f_{ui})^2.$$

The logistic log-likelihood for user u is

$$\log p_\theta(\mathbf{x}_u \mid \mathbf{z}_u) = \sum_i x_{ui} \log \sigma(f_{ui}) + (1 - x_{ui}) \log(1 - \sigma(f_{ui})),$$

2) Variational inference

$$\mathbf{z}_u \sim \mathcal{N}(0, \mathbf{I}_K), \quad \pi(\mathbf{z}_u) \propto \exp\{f_\theta(\mathbf{z}_u)\},$$
$$\mathbf{x}_u \sim \text{Mult}(N_u, \pi(\mathbf{z}_u)).$$

To learn the generative model in Eq, we are interested in estimating θ

$$q(\mathbf{z}_u) = \mathcal{N}(\boldsymbol{\mu}_u, \text{diag}\{\boldsymbol{\sigma}_u^2\})$$

🌟 The objective of variational inference 🌟

To optimize the free variational parameters

👉 the KL divergence $\text{KL}(q(\mathbf{z}_u) \| p(\mathbf{z}_u | \mathbf{x}_u))$ is minimized.

2.1 > Amortized inference and the variational autoencoder

Too many parameters to optimize!

👉 Bad for real-time recommendation

👉 The **VAE** replaces individual variational parameters with a **data-dependent function** (commonly called an inference model)

$$g_\phi(\mathbf{x}_u) \equiv [\mu_\phi(\mathbf{x}_u), \sigma_\phi(\mathbf{x}_u)] \in \mathbb{R}^{2K}$$

parametrized by ϕ with both $\mu_\phi(\mathbf{x}_u)$ and $\sigma_\phi(\mathbf{x}_u)$ being K-vectors and sets the variational distribution as follows:

$$q_\phi(\mathbf{z}_u \mid \mathbf{x}_u) = \mathcal{N}(\mu_\phi(\mathbf{x}_u), \text{diag}\{\sigma_\phi^2(\mathbf{x}_u)\}).$$

That is, using the observed data x_u as input, the inference model outputs the corresponding variational parameters of variational distribution q_ϕ(z_u | x_u).

$$q_\phi(\mathbf{z}_u \mid \mathbf{x}_u) \longrightarrow p(\mathbf{z}_u \mid \mathbf{x}_u)$$

approximate

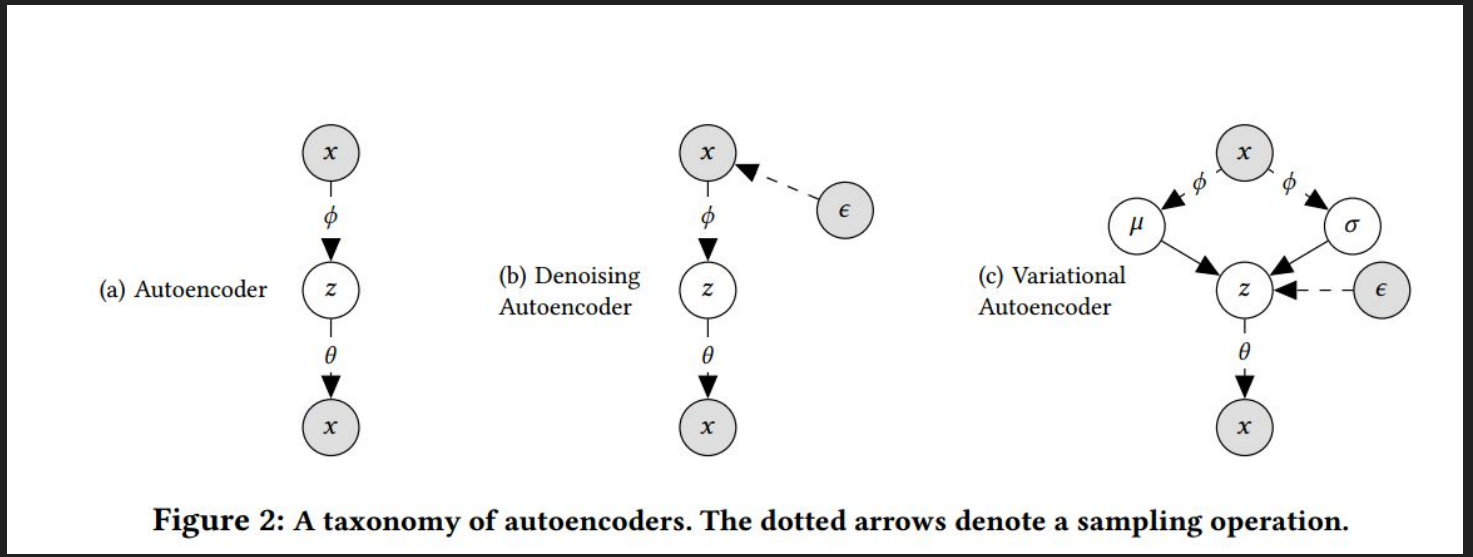


Figure 2: A taxonomy of autoencoders. The dotted arrows denote a sampling operation.

Learning VAEs

$$\log p(\mathbf{x}_u;\theta) \geq \mathbb{E}_{q_\phi(\mathbf{z}_u \mid \mathbf{x}_u)}\left[\log p_\theta(\mathbf{x}_u \mid \mathbf{z}_u)\right] - \mathrm{KL}(q_\phi(\mathbf{z}_u \mid \mathbf{x}_u) \| p(\mathbf{z}_u))$$
$$\equiv \mathcal{L}(\mathbf{x}_u;\theta,\phi)$$

Reparametrize $\mathbf{z}_u = \mu_\phi(\mathbf{x}_u) + \boldsymbol{\epsilon} \odot \sigma_\phi(\mathbf{x}_u)$ $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}_K)$

Algorithm 1: VAE-SGD Training collaborative filtering VAE with stochastic gradient descent.

Input: Click matrix $\mathbf{X} \in \mathbb{R}^{U \times I}$
Randomly initialize θ, ϕ
while *not converged* **do**
  Sample a batch of users $\mathcal{U}$
  forall $u \in \mathcal{U}$ **do**
    Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}_K)$ and compute $\mathbf{z}_u$ via reparametrization trick
    Compute noisy gradient $\nabla_\theta \mathcal{L}$ and $\nabla_\phi \mathcal{L}$ with $\mathbf{z}_u$
  end
  Average noisy gradients from batch
  Update θ and ϕ by taking stochastic gradient steps
end
return θ, ϕ

3. A taxonomy of autoencoders

Maximum-likelihood estimation in a regular autoencoder takes the following form :

$$\theta^{\mathrm{AE}}, \phi^{\mathrm{AE}} = \arg\max_{\theta,\phi} \sum_u \mathbb{E}_{\delta(\mathbf{z}_u - g_\phi(\mathbf{x}_u))} [\log p_\theta(\mathbf{x}_u \mid \mathbf{z}_u)]$$
$$= \arg\max_{\theta,\phi} \sum_u \log p_\theta(\mathbf{x}_u \mid g_\phi(\mathbf{x}_u))$$

We find that learning autoencoders is extremely prone to **overfitting** as the network learns to put all the probability mass to the non-zero entries in x_u.

👉 Dropout

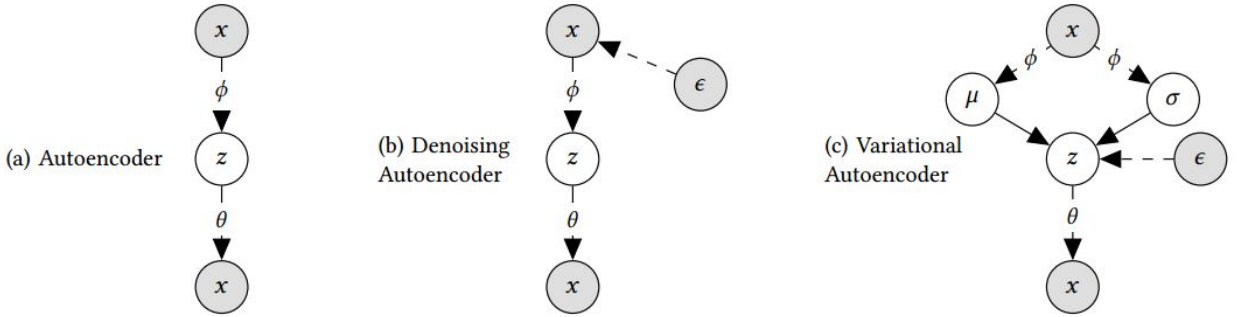


Figure 2: A taxonomy of autoencoders. The dotted arrows denote a sampling operation.

(a) It is trained to reconstruct input with the same objective.
(b) Adding noise to the input(or hidden representation) of an autoencoder
(c) Use an inference model parametrized by ϕ to produce the mean and variance of the approximating variational distribution

$$\mathcal{L}_\beta(\mathbf{x}_u;\theta,\phi) \equiv \mathbb{E}_{q_\phi(\mathbf{z}_u \mid \mathbf{x}_u)}[\log p_\theta(\mathbf{x}_u \mid \mathbf{z}_u)] - \beta \cdot \mathrm{KL}(q_\phi(\mathbf{z}_u \mid \mathbf{x}_u) \| p(\mathbf{z}_u))$$

The original vae formulation : $\beta = 1$
Our model Multi-VAE PR : $\beta \in [0, 1]$.

4. Prediction

$$\mathbf{z}_u \sim \mathcal{N}(0, \mathbf{I}_K), \quad \pi(\mathbf{z}_u) \propto \exp\{f_\theta(\mathbf{z}_u)\},$$
$$\mathbf{x}_u \sim \text{Mult}(N_u, \pi(\mathbf{z}_u)).$$

Given a user's click history x, we **rank** all the items based on the un-normalized predicted multinomial probability f_θ(z)

For Multi-VAE pr, we simply take the mean of the variational distribution z = μϕ(x); for Mult-DAE, we take the output z = gϕ(x).

Advantages of AE

1. Effectively make predictions for users by evaluating two functions

   : the inference model (encoder) $g_\phi(\cdot)$ / the generative model (decoder) $f_\theta(\cdot)$

2. Predictions is made cheaply and with low latency

Empirical study

We evaluate the performance of Mult-VAE PR and Mult-DAE

<Results>

- Mult-VAE^{PR} achieves state-of-the-art results on three real-world datasets when compared with various baselines, including recently proposed neural-network-based collaborative filtering models.
- For the denoising and variational autoencoder, the multinomial likelihood compares favorably over the more common Gaussian and logistic likelihoods.
- Both Mult-VAE^{PR} and Mult-DAE produce competitive empirical results. We identify when parameterizing the uncertainty explicitly as in Mult-VAE^{PR} does better/worse than the point estimate used by Mult-DAE and list pros and cons for both approaches.

1. Datasets

1) MovieLens-20M (ML-20M)

2) Netflix Prize (Netflix)

3) Million Song Dataset (MSD)

Table 1: Attributes of datasets after preprocessing. Interactions are non-zero entries. % of interactions refers to the density of the user-item click matrix X. # of the held-out users is the number of validation/test users out of the total number of users in the first row.

	ML-20M	Netflix	MSD
# of users	136,677	463,435	571,355
# of items	20,108	17,769	41,140
# of interactions	10.0M	56.9M	33.6M
% of interactions	0.36%	0.69%	0.14%
# of held-out users	10,000	40,000	50,000

2. Metrics

1) Recall@R

$$\text{Recall@}R(u,\omega) := \frac{\sum_{r=1}^{R} \mathbb{I}[\omega(r) \in I_u]}{\min(M, |I_u|)}$$

All items ranked within the first R to be equally important

2) NDCG@R

$$\text{DCG@}R(u,\omega) := \sum_{r=1}^{R} \frac{2^{\mathbb{I}[\omega(r) \in I_u]} - 1}{\log(r+1)}$$

5. Experimental results and analysis

Aim to answer the following two questions

1. How does **multinomial** likelihood compare with other commonly used likelihood models for collaborative filtering?

2. When does **Mult-VAE PR** perform better/worse than Mult-DAE?

Quantitative results

(a) ML-20M

	Recall@20	Recall@50	NDCG@100
Mult-VAE^{PR}	**0.395**	**0.537**	**0.426**
Mult-DAE	0.387	0.524	0.419
WMF	0.360	0.498	0.386
SLIM	0.370	0.495	0.401
CDAE	0.391	0.523	0.418

(b) Netflix

	Recall@20	Recall@50	NDCG@100
Mult-VAE^{PR}	**0.351**	**0.444**	**0.386**
Mult-DAE	0.344	0.438	0.380
WMF	0.316	0.404	0.351
SLIM	0.347	0.428	0.379
CDAE	0.343	0.428	0.376

(c) MSD

	Recall@20	Recall@50	NDCG@100
Mult-VAE^{PR}	**0.266**	**0.364**	**0.316**
Mult-DAE	**0.266**	**0.363**	0.313
WMF	0.211	0.312	0.257
SLIM	—	—	—
CDAE	0.188	0.283	0.237

Table 3: Comparison between NCF and Mult-DAE with $[I \rightarrow 200 \rightarrow I]$ architecture. We take the results of NCF from He et al. [14]. Mult-DAE model significantly outperforms NCF without pre-training on both datasets and further improves on Pinterest even comparing with pre-trained NCF.

(a) ML-1M

	NCF	NCF (pre-train)	Mult-DAE
Recall@10	0.705	**0.730**	0.722
NDCG@10	0.426	**0.447**	0.446

(b) Pinterest

	NCF	NCF (pre-train)	Mult-DAE
Recall@10	0.872	0.880	**0.886**
NDCG@10	0.551	0.558	**0.580**

Multinomial likelihood really performs well?

Table 4: Comparison of Mult-VAE^{PR} and Mult-DAE with different likelihood functions at the output layer on ML-20M. The standard error is around 0.002 (the results on the other two datasets are similar.) The multinomial likelihood performs better than the other two commonly-used likelihoods from the collaborative filtering literature.

	Recall@20	Recall@50	NDCG@100
Mult-VAE^{PR}	**0.395**	**0.537**	**0.426**
Gaussian-VAE^{PR}	0.383	0.523	0.415
Logistic-VAE^{PR}	0.388	0.523	0.419
Mult-DAE	**0.387**	**0.524**	**0.419**
Gaussian-DAE	0.376	0.515	0.409
Logistic-DAE	0.381	0.516	0.414

👉 The multinomial likelihood performs better than the other likelihoods.

When does Mult-VAE PR perform better/worse than Mult-DAE?

Intuitively, Mult- VAE PR imposes stronger modeling assumptions and therefore **could be more robust when user-item interaction data is scarce.**

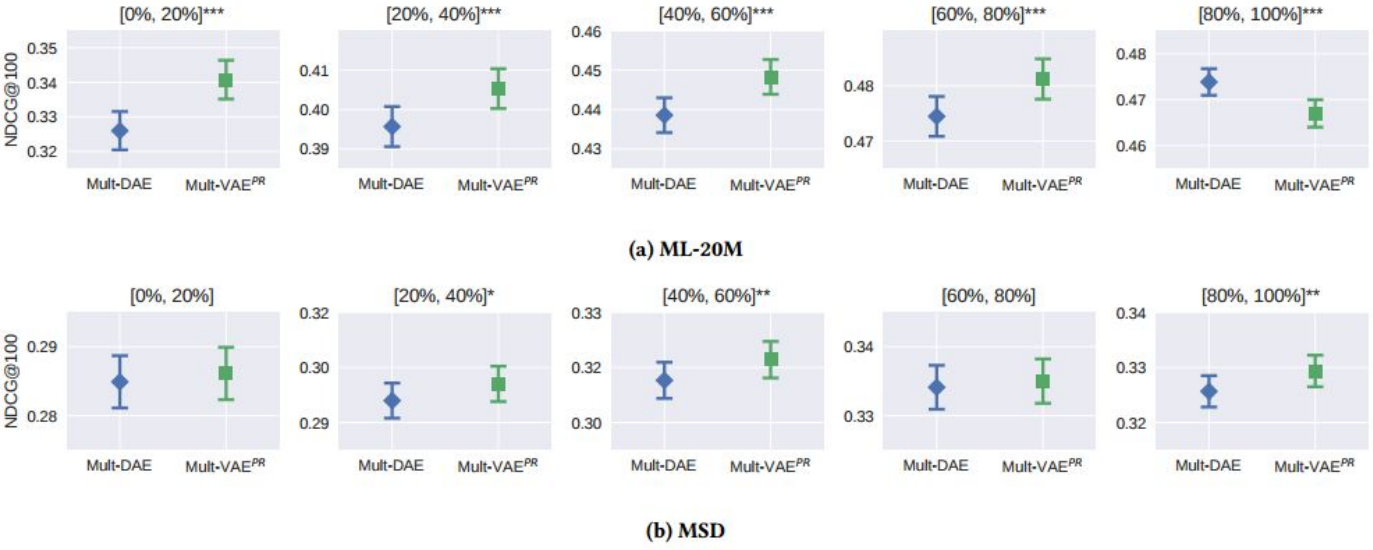


Figure 3: NDCG@100 breakdown for users with increasing levels of activity (starting from 0%), measured by how many items a user clicked on in the fold-in set. The error bars represents one standard error. For each subplot, a paired t-test is performed and * indicates statistical significance at $\alpha = 0.05$ level, ** at $\alpha = 0.01$ level, and * at $\alpha = 0.001$ level. Although details vary across datasets, Mult-VAE^{PR} consistently improves recommendation performance for users who have only clicked on a small number of items.**

Conclusion

We introduce a generative model with a multinomial likelihood function parameterized by neural network.

🌟**Multinomial likelihood** is particularly well suited to modeling user-item implicit feedback data.🌟

Thank You!