# AutoRec: Autoencoders Meet Collaborative Filtering

# Introduction

Problem: Given U users and their preferences (star ratings) about some of the given I items. The task is to find preferences about the remaining items.

This can also be viewed as a problem of recommendation where we want to recommend items to users  depending on the preferences which they have given.

# Collaborative Filtering

Collaborative Filtering (Main Idea) - If a person *A* has the same opinion as a person *B* on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

Mathematical Representation - $$r_{u,i} = \bar{r_u} + k \sum_{u' \in U} \mathrm{simil}(u, u')(r_{u',i} - \bar{r_{u'}})$$

where $\bar{r_u}$ is average rating given by user u and k is defined as $k = 1/\sum_{u' \in U} |\mathrm{simil}(u, u')|$

where simil(u,v) is similarity measure which can be cosine similarity or any other similarity measure. This is basic collaborative filtering approach.

# Problem setting

- Rating-based collaborative filtering:
  - m users
  - n items
  - Partially observed rating matrix R
  - Each item - partially observed vector $\mathbf{r}^{(i)} = (R_{1i}, \ldots R_{mi})$
- Task: Design an item-based autoencoder which can:
  - take as input each partially observed r(i)
  - project it into a low-dimensional latent space
  - and then reconstruct r(i) in the output space

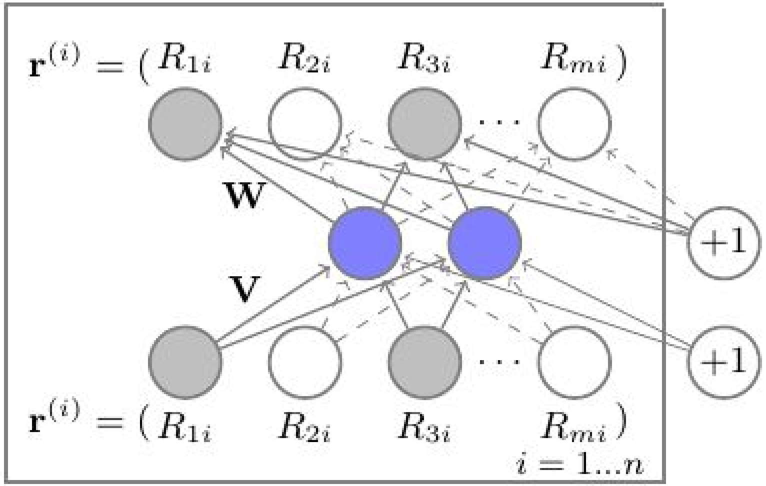
# Formulation

$$\min_{\theta} \sum_{\mathbf{r} \in \mathbf{S}} ||\mathbf{r} - h(\mathbf{r};\theta)||_2^2,$$

where $h(\mathbf{r};\theta)$ is the *reconstruction* of input $\mathbf{r} \in \mathbb{R}^d$,

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

# Model




# of parameters: 2mk + m + k

Objection function with regularization:

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

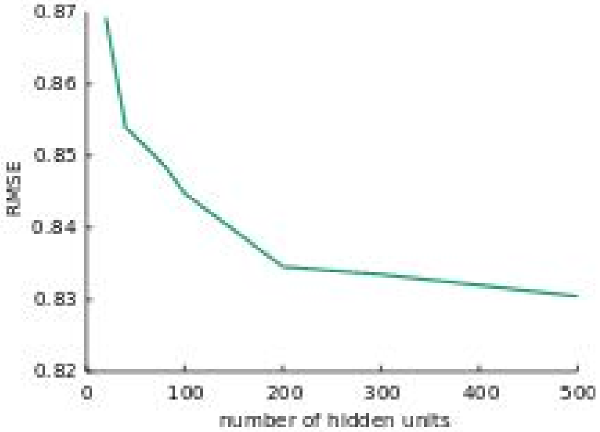
Using the model:

$$\hat{R}_{ui} = (h(\mathbf{r}^{(i)}; \hat{\theta}))_u$$

# Experimental Results

1. The dataset used for experimentation was movielens dataset which consisted of movie ratings of 1 million users.
2. The RMSE values obtained for different methods are given below.
3. Performance steadily increases with the number of hidden units, but with diminishing returns.

| | ML-1M | ML-10M |
|---|---|---|
| U-RBM | 0.881 | 0.823 |
| I-RBM | 0.854 | 0.825 |
| U-AutoRec | 0.874 | 0.867 |
| I-AutoRec | **0.831** | **0.782** |



# Second order optimization methods:

A very brief overview

# Newton's method

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma[\mathbf{H}f(\mathbf{x}_n)]^{-1}\nabla f(\mathbf{x}_n)$$

- Computing the inverse when the data is high dimensional could be very expensive
  - Alternative 1: Pose the problem of finding updates as a system of linear equations
  - Alternative 2: Compute the Hessian (or its inverse directly) from changes in the gradient (Quasi-Newton methods).

# BFGS

- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
- Among the most popular Quasi Newton methods
- It stores the approximation to the Hessian matrix.
- If the dimension of the x vector in n, that takes O(n^2) memory.

# L-BFGS

- Limited-memory (L) Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
- L-BFGS is a BFGS version that does not require to explicitly store the approximation to the Hessian
    - So, it needs less memory