Netflix Official Final Report

Members: Sha Sha, Yu Wu, Diyi Liu, Haotian Sheng, Wen Gu

Overview

- Final Performance
- Restricted Boltzmann Machine
- Singular Value Decomposition
- K-Nearest Neighbor
- Blending
- Division of Labor

Final Performance

- Test Performance:
  - RMSE: 0.88136
  - Above water: 7.3618%
- Quiz Performance:
  - RMSE: 0.88051
  - Above water: 7.4511% (Indicating overfitting on quiz)
- Novelty Score:
  - 0.4294

Models

Basic Restricted Boltzmann Machine

- Two RBM models:
    - Model 1 trained with base, validation, hidden
    - Model 2 trained with base, validation, hidden, probe
- Quiz Performance:
    - Model 1: 3.02%
    - Model 2: 3.52%
    - Indicates **shift between training set and test set**
- Model Detail:
    - 88,850 visible units (known), 100 hidden units(need to be tuned)
    - Total number of parameters: 8,973,950

Basic Restricted Boltzmann Machine -- Hyperparameter Tuning

- Number of hidden units:
  - We also tried 150, 200 hidden units, the performance improved is very tiny.
  - Linear relationship between number of hidden units and training time.
- Contrastive Divergence (CD):
  - Adaptive Gibbs Sampling Steps (T)
  - Epoch 0: T = 1, Epoch 19: T = 3, Epoch 28: T = 5, Epoch 43: T = 9
  - Linear relationship between steps and training time.
- Adaptive Learning Rate:
  - Start with 2e-5/user, weight decay = 0.999, slow convergence
  - At epoch 46, increase the learning rate to 3e-5/user, weight decay = 0.999
- Epochs: 150

Epoch 1

Epoch 46

Epoch 150

Random Initialized RBM Model

Learning Rate 2e-5/user, weight decay = 0.999

Training Dataset:
base data

Pretrained RBM Model

Learning Rate 3e-5/user

weight decay = 0.999

Dataset:
base,
validation,
hidden

RBM Model 1

Used for blending training

Dataset:
base,
validation,
hidden,
probe

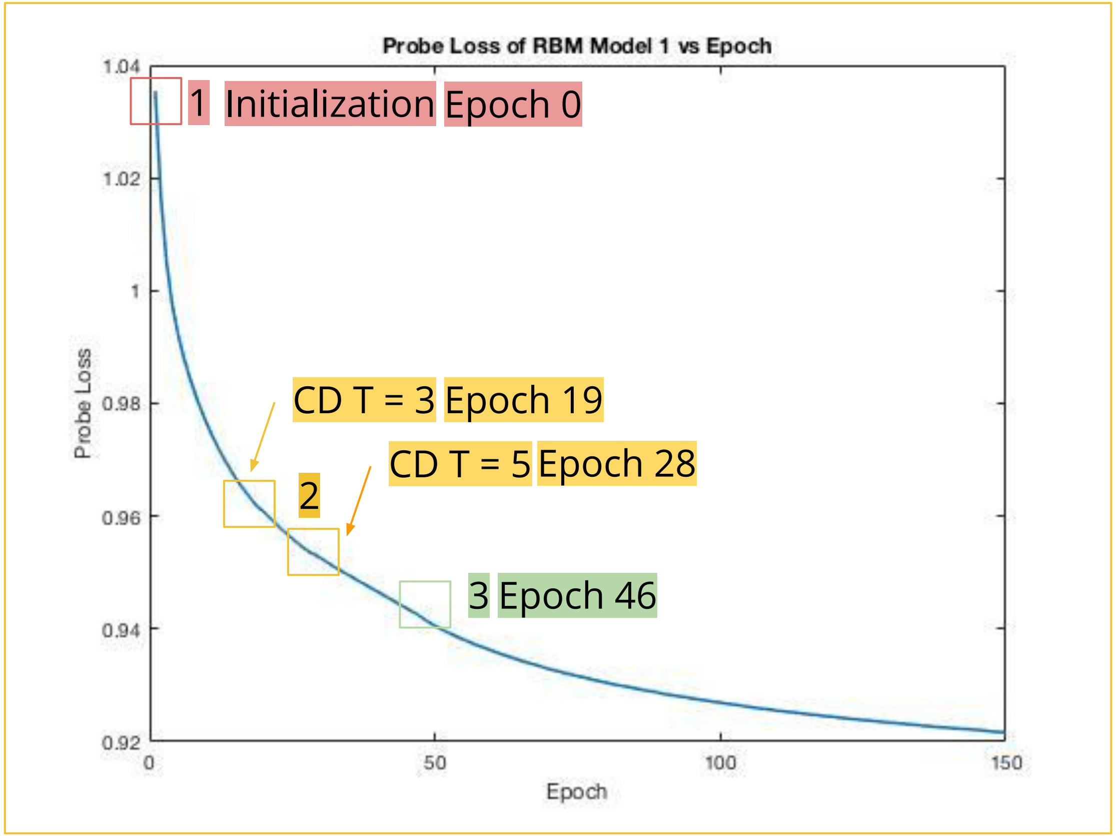
RBM Model 2

For final blending

Training Setup:

AWS c5n.2xlarge, 3.0 Ghz Intel Xeon Platinum 8000 series CPU, 21GiB RAM (single thread)

Basic Restricted Boltzmann Machine -- Probe Loss Function



1 : Initialization does matter

- Weight / Hidden Units bias ~ Normal（0, 0.01）
- Visible Units bias log(p/(1-p))
- First epoch RMSE loss decrease by 0.03

2: Increasing CD steps in training helps faster convergence

3: Feed in more data (Base, Hidden, Validation); Adjust Learning rate.

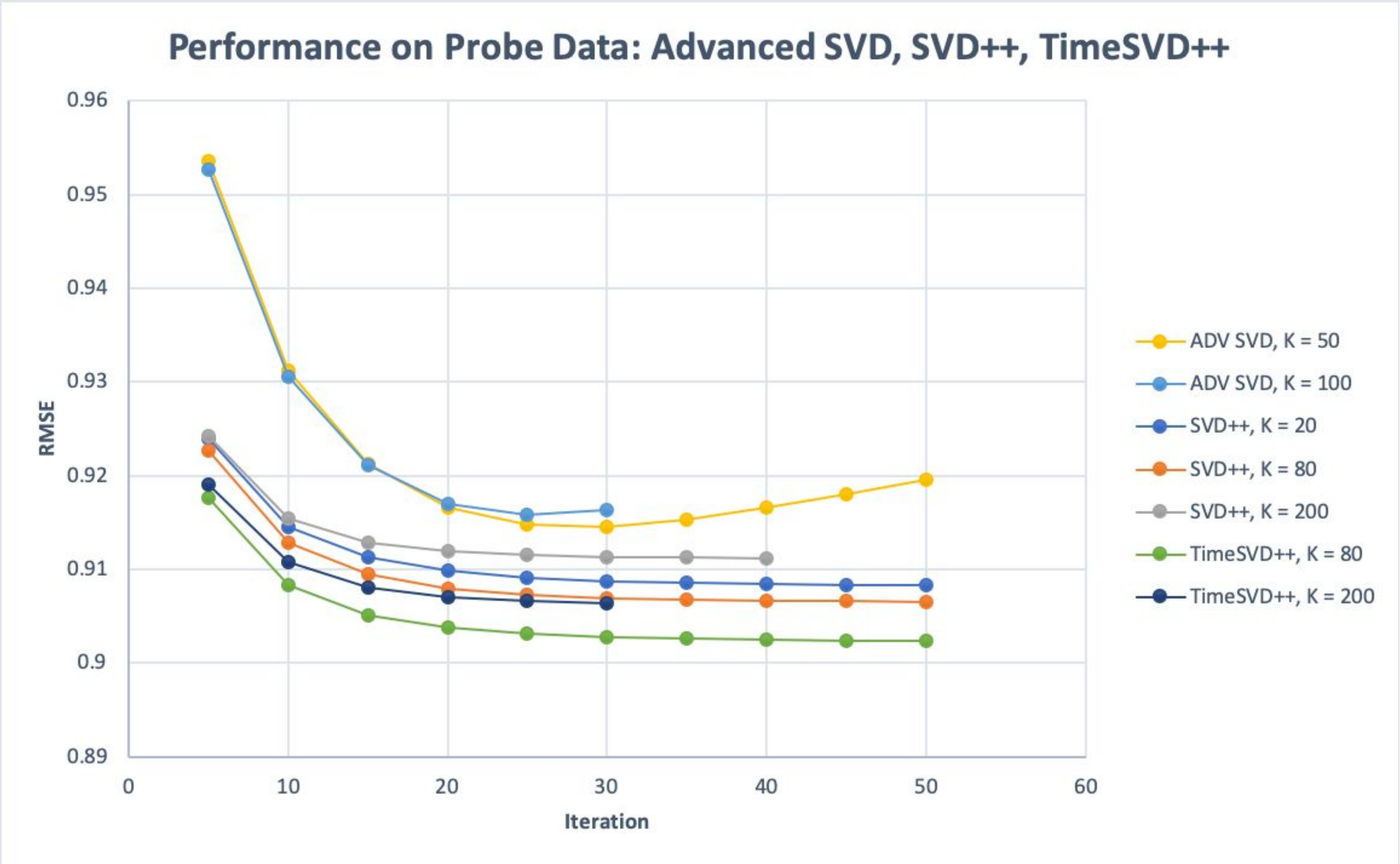
Basic Restricted Boltzmann Machine -- Optimization

- Computational Challenge
  - Large number of parameters embedded in the model
- Sparse Matrix
  - Missing ratings do not need to be considered in the sampling and update
  - Employ Unordered_map to lock on to the weights of presenting ratings
  - By omitting the missing weights, we reduce the computation by 5x
- Eigen Linear Solver
  - The Eigen Linear solver is 1.5x slower
- Other tricks
  - Using g++ complier optimization (g++ -O2) reduce running time by 7x
  - -O2 is faster than -O1, but almost the same as -O3
- Training Time/ Epoch : ~43mins, CD step = 1

Singular Value Decomposition

- Best Performance of Single Model:
  - More advanced SVD: (~2mins/epoch)
    - $\underset{U,V,a,b}{\operatorname{argmin}} \frac{\lambda}{2}\left(\|U\|^2+\|V\|^2+\|a\|^2+\|b\|^2\right)+\sum_{(i,j)\in S}\left(\left(Y_{i,j}-\mu\right)-\left(u_i^T v_j+a_i+b_j\right)\right)^2$
    - 4.33% (K = 20)
  - SVD ++ (~15mins/epoch)
    - 5.36% (K = 80)
  - Time SVD ++ (~20mins/epoch)
    - 6.0563% (K= 80)

Singular Value Decomposition



Singular Value Decomposition

- Implementation & Optimization
  - parameters referred to Koren's paper: *"Advances in Collaborative Filtering"*
  - learning rate decay per iteration (**0.9*)
  - update y per user to speed up (tradeoff between accuracy and speed)
- SVD++

for $iter \leftarrow 0$ to $maxIter$
  for $data\_point$ in $training\_dataset$
    if new_user then
      $curr\_user \leftarrow new\_user$
      update $sumYj$ for $curr\_user$
    perform updates for Bu, Bi, Pu, Qi
    (save $tempYPart$ for the update of y)
    if ($last\_data\_point$ of $curr\_user$) then
      update y with $tempYPart$
      update $sumYj$ for $curr\_user$

per data point

per user
(approximation to speed up)

0.007 0.005

$$b_u \leftarrow b_u + \boxed{\gamma} \cdot (e_{ui} - \boxed{\lambda_5} \cdot b_u)$$

$$b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i)$$

sumYj

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \boxed{\textstyle\sum_{j \in \mathrm{R}(u)} y_j}) - \lambda_6 \cdot q_i)$$

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_6 \cdot p_u)$$

$$\forall j \in \mathrm{R}(u):$$

tempYPart 0.015

$$y_j \leftarrow y_j + \gamma \cdot (\boxed{e_{ui} \cdot |\mathrm{R}(u)|^{-\frac{1}{2}} \cdot q_i} - \boxed{\lambda_6} \cdot y_j)$$

Singular Value Decomposition

- Time SVD++

for $iter \leftarrow 0$ to $maxIter$
  for $data\_point$ in $training\_dataset$
    if new_user then
      $curr\_user \leftarrow new\_user$
      update $sumYj$ for $curr\_user$
      perform updates for $Bu, Bi, B_{u,t}, Pu, Qi, Bi_{Bin}, \alpha_u$
      (save $tempYPart$ for the update of y)
      if ($last\_data\_point$ of $curr\_user$) then
        update y with $tempYPart$
        update $sumYj$ for $curr\_user$

per data point

per user (approximation to speed up)

Differences from SVD++ in Time SVD++

0.007 0.01

$$b_u \leftarrow b_u + \boxed{\gamma} \cdot (e_{ui} - \boxed{\lambda_5} \cdot b_u)$$

$$b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i)$$

$$\boxed{b_{u,t} \leftarrow b_{u,t} + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_{u,t})}$$

$sumYj$

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + |R(u)|^{-\frac{1}{2}} \boxed{\textstyle\sum_{j \in R(u)} y_j}) - \lambda_6 \cdot q_i)$$

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_6 \cdot p_u)$$

$$b_{i,Bin(t_{ui})} \leftarrow b_{i,Bin(t_{ui})} + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_{i,\boxed{Bin}(t_{ui})})$$

$$\alpha_u \leftarrow \alpha_u + \boxed{\gamma_2} * (e_{ui} * dev_u(t_{ui}) - \lambda_5 * \alpha_u)$$ 30

$\forall j \in R(u)$: 0.0001 $tempYPart$ 0.015

$$y_j \leftarrow y_j + \gamma \cdot (\boxed{e_{ui} \cdot |R(u)|^{-\frac{1}{2}} \cdot q_i} - \boxed{\lambda_6} \cdot y_j)$$

Other Speed Up:

$$dev_u(t) = sign(t - t_u) \cdot |t - t_u|^{\beta}$$

- Calculate t_u for each user in initialization for $dev_u\ (t_{ui})$
- Calculate $dev_u\ (t_{ui})$ in advance, save in $Dev[user][date]$ matrix for future indexing

K-Nearest Neighbor

- Implementation
  - Pearson Calculation:
  - Prediction: weighted average rating

Struct: *Pearson Intermediates* Array [# movies]

$$r_{xy} = \frac{n\boxed{\sum x_i y_i} - \boxed{\sum x_i}\boxed{\sum y_i}}{\sqrt{n\boxed{\sum x_i^2} - (\sum x_i)^2}\ \sqrt{n\boxed{\sum y_i^2} - (\sum y_i)^2}}.$$

top k movies similar to movie i rated by user u

movie j rating given by user u

average rating of movie j given by all users

$$r_{ui} = \frac{\sum_{j=1}^{k} p_{ij} \cdot \left(r_{uj} - \overline{r}_j\right)}{\sum_{j=1}^{k} \left|p_{ij}\right|} + \overline{r}_i$$

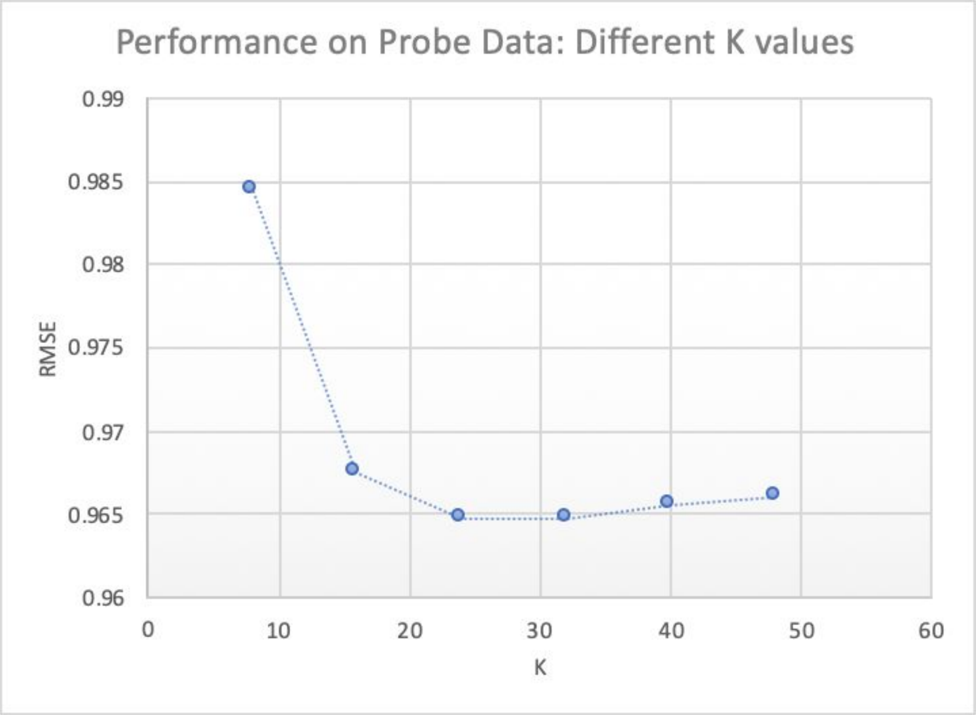
average rating of movie i given by all users

new test point: user u, movie i

Pearson correlation coefficient between movie i and j

K-Nearest Neighbor

- Performance
  - not very well (below water) as a single model
  - k = 8: RMSE = 0.9844
  - k = 16: RMSE = 0.967442
  - k = 24: RMSE = 0.964737
  - k = 32: RMSE = 0.964719
  - k = 40: RMSE = 0.965459
  - k = 48: RMSE = 0.966065



Blending

Blending

- Data
  - training data:
    - X_train = probe predictions (trained on training data)
    - y_train = probe true ratings
      - RBM
      - SVD (advanced SVD, SVD ++, Time SVD ++)
        - different number of factors (K = 20, 50, 80, 200, 300)
      - KNN
        - different number of nearest neighbors (K = 8, 16, 32)
  - test data:
    - X_test = qual predictions (trained on training data + probe data)

Blending

- Models
    - Referred to *"Combining Predictions for Accurate Recommender Systems"*
    - Generally, more models, better performance
    - Linear Regression:
        - Ridge Regression
            - regularization strength *alpha=5e-6*
            - Best performance: ~7.36%
    - XGBoost: (optimized distributed gradient boosting)
        - XGBoost Regressor
            - *max_depth=6, n_estimators=70,* other parameteres set default
            - Best performance: ~7.4%
    - Nerual Network:
            - *Two* dense hidden layers with *64* units, "relu" activation
            - Optimizer: Stochastic gradient descent (SGD), *lr=5e-4, decay=5e-7*
            - Best performance: ~7.4%

Blending

- Further combination: Mean
  - Calculate the mean predictions of the blended qual predictions
    - Ridge Regression
    - XGBoost Regressor
    - Nerual Network
  - Performance:
    - input: all three prediction data
      - 7.42%
    - input: exclude predictions from Ridge Regression
      - 7.45%

Division of Labor

- RBM: Wen Gu, Haotian Sheng
- SVD: Yu Wu, Sha Sha, Diyi Liu
- KNN: Yu Wu, Sha Sha, Diyi Liu
- Blending: Yu Wu, Sha Sha, Diyi Liu

Questions?

Thanks!