Abstract

A solution for the Netflix Prize was developed based on back propagation neural networks. The solution is different than most other Collaborative Filtering techniques in that rather than perform a global dimensionality reduction, this method focuses on each desired prediction by creating an entirely new neural network for each prediction. The implementation was parallelized using MPI, achieving speedups of varying quality depending on the parameters of the Neural Network. The most recent results are comparable with a benign solution to the Netflix problem, however with more parameter adjustments, it is expected to be possible to achieve results better than the benign solution.
1 Background

Netflix, Inc. has recently started a competition to improve movie rating feedback. Netflix is a company that provides online movie rentals. Users choose which movies they would like to view, placing them in a queue. The user is mailed each movie sequentially. Once a movie is mailed back, another is sent to replace it.

An important part of the online system involves recommending to users which movies they might like to see. Users are asked to rate various movies that they have seen. Using this information and information from other users, Netflix attempts to predict how each user rates movies for which they have not given a rating. This problem falls under the general category of Collaborative Filtering.

2 Problem Statement

The Netflix Prize competition attempts to formalize this rating prediction system for the sake of improving the quality of predictions. Netflix has provided a set of real-world ratings from thousands of customers and movies.

Informally, the problem statement for the contest is, given many previous ratings, the dates of each rating, and a list of movies and users, guess the ratings the various users would have given those movies on a given date\(^1\). The running time of the system is not taken into account. Once predictions have been performed, the results are submitted to Netflix online. An unseen oracle at Netflix calculates the Root Mean Square Error (RMSE) between the given ratings and the actual ratings. If a ten percent improvement is made over Netflix’s current algorithm, the submitter will receive one million dollars. As of this writing, no one has attained such a score, but they have done significantly better than Netflix’s current algorithm, with 6.1% improvement.

We can formalize the problem in the following manner. We are given a set of movies \(M\) and a set of customers \(C\). A rating \(r\) is an integer such that \(1 \leq r \leq 5\). A predicted rating is allowed to be a real number, rather than an integer, giving more gradation to the predictions. The training set data for the movie \(m\) rated by customers in the set \(C_m \subseteq C\) is given by

\[
T_m = \{r_{t,c} : \forall c \in C_m, r \in \mathbb{N}, 1 \leq r \leq 5, \ c \ gave \ m \ rating \ r\}.
\]

The entire training set, therefore, can be described as

\[
T = \{T_m : \forall m \in M\}.
\]

We can describe the qualifying data (the desired ratings) as

\[
Q_m \subseteq C
\]

\[
Q = \{Q_m : \forall m \in M\}.
\]

The qualifying data gives no rating data, rather it only provides a set of clients for each movie.

\(^1\)The implementation described by this paper ignores date information.
The predictions created by the collaborative filtering algorithm are described by

\[ P_m = \{ r_{p,c} : c \in Q_m, r_p \in \mathbb{R}, 1 \leq r_p \leq 5, \text{ guess } c \text{ gave rating } r_p \text{ to } m \} \]

\[ P = \{ P_m : \forall m \in M \} \]

Note that \( r_p \) need not be an integer.

Finally, the metric of quality is the RMSE between the actual values, \( A_m \), and the prediction.

\[ A_m = \{ r_{a,c} : c \in Q_m, r \in \mathbb{N}, 1 \leq r \leq 5, \text{ c actually chose rating } r_{a,c} \} \]

\[ RMSE = \sqrt{\frac{\sum_{m \in M} \sum_{r_{p,c} \in P_m} (r_{p,c} - r_{a,c})^2}{\sum_{r_{a,c} \in A_m}}} \]

### 3 Approach

In many Collaborative Filtering algorithms of this nature, a global dimensionality reduction is performance on the entire training set. Other participants in the Netflix competition have used this method with a high degree of success. However, this paper proposes a different system.

The approach described here is designed to take advantage of the unlimited timing constraints given in the problem statement. There is no need to perform global operations if more refined results can be produced by only looking at each prediction in turn.

Each desired value in the qualifying set \( Q \) can be taken independent of all other predictions. Therefore we will look at only one movie and client at a time. Using a neural network, we can predict \( r_p \) for \( c \in Q_m \). That is to say, we will guess how client \( c \) rated movie \( m \).

We have access to all movies client \( c \) has rated:

\[ M_c = \{ m : \forall m \in M, r_{t,c} \in T_m \} \]

The size of \( M_c \) varies considerably between users. For the sake of speed, we only look at a small random subset of \( M_c \) called \( M_{\text{rand},c} \).

We also can find those clients who have also rated movie \( m \):

\[ C_m = \{ c : \forall r_{t,c} \in T_m \} \]

Again, for the sake of speed we take only a random subset of these values, \( C_{\text{rand},m} \).

Next, we assume that the rating \( c \) would have given \( m \) is a function of the ratings it has already performed on other movies, so we create a back propagation neural network that can attempt to learn that function. The neural network \( N_{m,c} \) is specific to client \( c \) rating movie \( m \). \( N_{m,c} \) has \(|M_{\text{rand},c}|\) inputs and 1 outputs. The training vectors for the neural network are taken from the ratings performed by other clients using the function \( \phi \). When a rating is not available for a given client and movie, the global average for that movies is used instead.

\[ \phi(m,k) = \begin{cases} \frac{r_{t,k}}{|T_m|} & \text{if } r_{t,k} \in T_m \\ \frac{\sum_{r_{t,c} \in T_m} r_{t,c}}{|T_m|} & \text{if } r_{t,k} \notin T_m \end{cases} \]
A single training vector based on client $k$,

$$S_k = [\phi(m, k) : m \in M_{\text{rand}, k}],$$

has a corresponding output value $r_{t,k} \in T_m$, where $m$ is the movie we are trying to rate. All of the training data for $N_{m,c}$ is therefore

$$D_{m,c} = \{(S_k, r_{t,k}) : k \in C_{\text{rand}, m}, r_{t,k} \in T_m\}$$

The training data $D_{m,c}$ is presented to the neural network, which learns the given function using back propagation. Finally the corresponding ratings given by client $c$ are presented to the neural network and the output gives $r_{p,c}$ for movie $m$.

## 4 Example

To motivate the description above, we will look at an example. Figure 1 shows three users and five movies and various ratings. Of course in the real data set there are about 100 million such ratings. In the example, the predicting client $c$ is Carl. Notice that Carl has rated all four movies, Psycho, Manos: The Hands of Fate, Vertigo and Jaws, so these four movies will make up $M_{\text{rand}, c}$.

Create a neural network with $|M_{\text{rand}, c}| = 4$ inputs and 1 output. Each input corresponds to one of the four movies as shown in Figure 2.

The training data $D_{m,c}$ is composed of ratings from the other users, Ann and Bob:

$$D_{m,c} = \{([4, 1, 2.5, 5]^T, 5), ([2, 5, 1, 5]^T, 2)\}$$

The bold values are absent from the original training data, so $\phi$ has replaced the missing data with the average rating for the given movie.

The next step is to train the neural network on $D_{m,c}$. The vectors are presented to $N_{m,c}$, and learning can take place using RPROP. RPROP is used for its speed of convergence. It
also conveniently provides dynamic learning rates, putting less burden on the implementor to determine a single adequate learning rate. Training halts when (a) all the ratings are correctly classified, (b) the MSE of the output goes below some threshold, or (c) the maximum number of training epochs is reached. Due to timing constraints, typically the latter is the limiting factor.

After the network has been trained, Carl’s own rating vector \([4, 1, 4, 5]^T\) is fed into the neural network and the result added to the list of results. As we can see, Ann’s ratings follow much more closely to Carl’s than Bob’s, so we expect the neural network to predict Carl rated *Night of the Living Dead* around 5.

5 Implementation

A solution to the Netflix Prize problem has been created in C++ and MPI based on the method described in the previous sections.

5.1 Neural Network

The neural network used by this project is a slightly modified version of the neural network created by the author in Neural Networks class under the direction of Professor Robert Keller. The network is placed in RPROP mode and trained in the usual manner.

5.2 Data and Indexing

One of the key challenges with this process is the sheer volume of data and predictions required. The system has half a million users, just less than eighteen thousand movies, and 100 million training ratings. The qualifying set \(Q\) asks to product 2.8 million ratings. All of this data is provided by Netflix in separate ANSI files, one for each movie.
In order to efficiently create $M_{\text{rand},c}$ and $C_{\text{rand},m}$ for each prediction, it is necessary to have random access to all the training data. Creating $M_{\text{rand},c}$ requires finding all the movies rated by client $c$. This has been made possible with an index that holds all the movies and ratings each client has made. Likewise, the creation of $C_{\text{rand},m}$ requires an index mapping in the opposite direction. The second index provides fast access to the various ratings a given movie has received.

Low level memory management features in C++ were used to assure that all of this data would fit on one machine. Using this strategy, both indices together require about 800 megabytes of memory. The indices and the raw data were made to be memory mapped so all the data could be loaded directly from disk into memory. After all of this optimization, startup time improved from 15 minutes to as few as 8 seconds, when disk caching comes into play.

### 5.3 Parallelization

Several constraints dictated the method of parallelization of this problem. The primary challenge was memory availability. It is not necessarily the case that each machine has 800 megabytes of memory available. Each task, however, must have random access to the memory in order to generate the needed training data. Since random access on a distributed memory system is very slow, it also does not make sense to distribute pieces of the dataset across the cluster.

Instead, a master-slave system was created. A single master node holds all training data. The master compiles together all the sample data and sends it to the slave nodes for processing. One such submission will be known as a task. The slave nodes’ sole responsibility lies in training a neural network and generating the predicted rating.

### 5.4 Load Balancing and Checkpointing

The benign strategy for distributing the load in this problem is to simply give each of the $p$ processors $|Q|/p$ tasks. There is, however, potentially huge variability in the time required to compute any two given predictions.

Instead, the tasks are assigned only as clients are ready to perform them. As a client finishes a task, it asks for a new one. The master creates the task (or ideally already has a task ready to send), gives the task a sequentially numbered id, and sends it to the client. Once the client completes the task, it sends back the predicted rating and task id.

In order to provide crash recovery, the master saves these results to disk as they are received. The results may not be received in order, however, so a priority queue assures order is kept.

### 5.5 Speedup

As can be seen in Figure 3, the speedup of the system is dependent on the relative size of the tasks. The master node and communication is the limiting factor. The first chart shows nearly linear speedup; however, the full computation for that graph takes a week. An overbearing amount of work is performed in the neural network itself. On the other hand,
Figure 3: The first graph shows the speedup on the system with very large computation relative to the amount of time needed to actually find data. The second graph shows the speedup computation time is comparable to the data retrieval and communication time.
Table: Various submissions made to Netflix

| RMSE  | Time to Run | Network Structure          | Epochs | |M_{rand,c}| |C_{rand,m}|
|-------|-------------|----------------------------|--------|-----------------|-----------------|
| 1.235 | 36 hours    | 7 hyp - 1 lin              | 300    | 200             | 200             |
| 1.3954| 36+ hours   | 30 hyp - 8 hyp - 1 lin     | 100    | 70              | 200             |
| 1.3245| 1 week      | 70 log - 50 hyp - 1 lin    | 150    | 200             | 400             |
| 1.0932| 10 hours    | 20 log - 20 log - 20 hyp - 1 lin | 20 | 20 | 20 |
| 1.2524| 10 hours    | 20 log - 20 log - 20 hyp - 1 lin | 40 | 20 | 20 |
| 1.5335| 10 hours    | 20 log - 20 log - 20 hyp - 1 lin | 120 | 20 | 20 |
| 1.0853| 10 hours    | 5 log - 5 log - 5 hyp - 1 lin | 20 | 15 | 30 |
| 1.1010| 10 hours    | 4 log - 8 hyp - 1 lin      | 20 | 15 | 60 |

Figure 4: A listing of the various submissions made to Netflix. The network structure column shows the various layers of the neural network used along with the type and number of neurons in each layer. Here hyp means a hypertangent sigmoid function, log means a logsig function, and lin means a linear function.

the full computation in the second graph only takes ten hours. The master and the network have become the bottleneck in that case. Hence the poor speedup curve.

6 Results

Netflix restricts the number of submissions to one per 24 hour period to avoid learning the results off the oracle itself. The implementation this paper describes has been submitted eight times with various parameter adjustments made during each submission. Figure 4 shows the various RMSE and parameters for these submissions. For comparison, here are a few RMSE for other algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>1.0540</td>
</tr>
<tr>
<td>Netflix’s Solution</td>
<td>0.9514</td>
</tr>
<tr>
<td>Contest leader (as of 12/8/06)</td>
<td>0.8933</td>
</tr>
<tr>
<td>Million dollar goal</td>
<td>0.8563</td>
</tr>
</tbody>
</table>

Strangely, the results showed improvement as the network was allowed to perform more and more iterations (epochs). This can be seen in runs 4 through 6. The only modification to parameters given in each of these runs was the number of epochs; however, as the number of epochs increased, the value of the RMSE became significantly worse. One possible explanation for this behavior is the neural network over-learning the data. This would indicate that the network is too powerful for the given problem size. In an attempt to counterbalance this, recent efforts in simplifying the network structure show good progress.

7 Future Work

One of the biggest downfalls with the described collaborative filtering method is how it handles missing data. Currently, the algorithm simply uses the global average for a given
movie to replace missing ratings. There are several other possible solutions, however. For example, it may be possible to change how the neural network learns. Those values that are not present could be made to not have an effect on the learning process. This may, however, lead to imbalanced results. Another promising strategy is to apply other dimensionality reductions prior to using rating-specific neural networks. This may involve, for example, performing the Singular Value Decomposition on the data and then using neural networks to refine preferences between similar clients and movies.

Another possible strategy for improving the results would involve creating multiple neural networks per task. In such an strategy, when there is high variability between predictions, then the algorithm could simply give up and return the global average, but if there is a consistent rating, then the values could be averaged and returned.

Depending on the implementation, other more effective methods could be used to parallelize the algorithm. The Singular Value Decomposition, for example, already has existing parallel implementations. Other methods for improving the current system could include creating multiple master nodes, distributing all of the data to each node (if hardware allows), etc.

In any case, competitors in the Netflix competition are well on their way to success, and the remainder of the competition looks to be productive.