Predictive Model: Unraveling Movie Ratings with RBM-Based Collaborative Filtering

Introduction/Project goals:

Netflix needs to recommend movies based on a user's viewing history and preferences, and to solve this problem, Netflix launched a contest to encourage data scientists to develop advanced algorithms to predict user preferences and improve movie recommendations. In the context of recommender systems, linear regression can be used to predict user ratings of movies based on various features. Restricted Boltzmann Machine (RBM) is a type of artificial neural network that excels in capturing complex patterns and dependencies in data.RBM excels in collaborative filtering, which can be used to make analysis by analyzing the preferences of similar users. Project goals: The main goal of this project is to train a model for movie recommendation through collaborative filtering using a recommender system that focuses on linear regression and restricted Boltzmann machines. Collaborative filtering involves predicting user preferences based on the preferences of other users with similar tastes.

Linear Regression:

Mathematical view

$Main\ objective\ from\ a\ math\ perspective :: \min\|Ab - c\|_2^2$
$Where\ A\ is\ a\ sparse\ matrix\ similar\ to\ a\ one\ hot\ encoding$
$each\ row\ of\ A\ is\ corresponds\ to\ a\ movie\ review$
$each\ row\ has\ 2\ non\text{-}negative\ numbers, their\ indices\ correspond\ to$
$-the\ rated\ movie\ -\ the\ user\ who\ rated\ the\ movie$
$b\ is\ the\ biases\ we\ are\ looking\ for$
$c\ is\ the\ result\ we\ check\ our\ loss\ on\ but\ we\ normalize\ it$

Cell with rating →

	b_1	b_2	b_3	b_4	b_A	b_B	b_C	b_D
	1	0	0	0	1	0	0	0
	1	0	0	0	0	0	0	1
	0	1	0	0	0	1	0	0
# of ratings ↕	0	1	0	0	0	0	1	0
	0	0	1	0	1	0	0	0
	0	0	1	0	0	0	1	0
	0	0	0	1	0	1	0	0
	0	0	0	1	0	0	0	1

$\cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_A \\ b_B \\ b_C \\ b_D \end{pmatrix} - \begin{pmatrix} 5-3.3 \\ 3-3.3 \\ 5-3.3 \\ 1-3.3 \\ 5-3.3 \\ 4-3.3 \\ 3-3.3 \\ 5-3.3 \end{pmatrix} = Ab - c$

$\left||Ab - c|\right|_2^2 = (Ab - c)^T(Ab - c)$
$= \left(b^T A^T - c^T\right)(Ab - c)$
$= b^T A^T Ab - b^T A^T c - c^T Ab + c^T c$
$we\ take\ the\ derivate\ with\ respect\ to\ b$
$\Rightarrow 2A^T Ab - 2A^T c$
$we\ want\ the\ derivate\ to\ be\ zero\ so\ we\ solve\ for:$
$A^T Ab = A^T c$
$when\ we\ introduce\ a\ regularization\ term :$
$we\ want\ to\ find\ the\ derivative\ of\ \|Ab - c\|_2^2 + \lambda\|b\|_2^2$
$\|Ab - c\|_2^2 + \|b\|_2^2 = b^T A^T Ab - b^T A^T c - c^T Ab + c^T c + \lambda b^T b$
$we\ take\ the\ derivate\ with\ respect\ to\ b$
$\Rightarrow 2A^T Ab - 2A^T c + 2\lambda b$
$we\ want\ the\ derivate\ to\ be\ zero\ so\ we\ solve\ for:$
$\left(A^T A + \lambda I\right)b = A^T c$

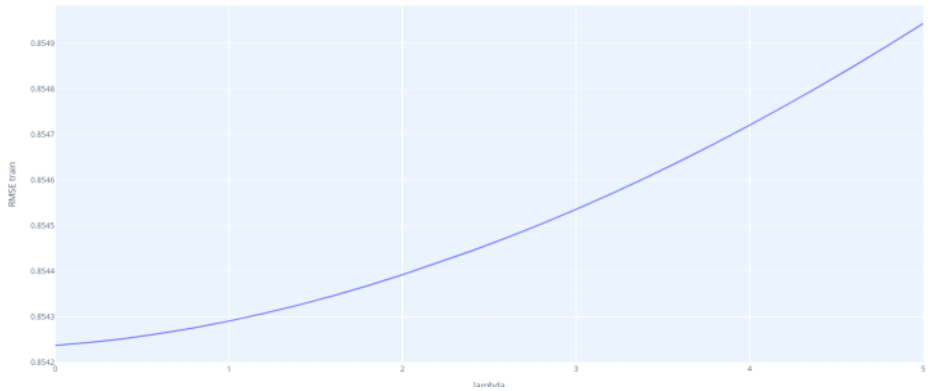
Implementation and Analysis

$The\ implementation\ of\ the\ math\ is\ very\ simple, for\ A\ the\ encoding\ was\ done\ row\ by\ row:$

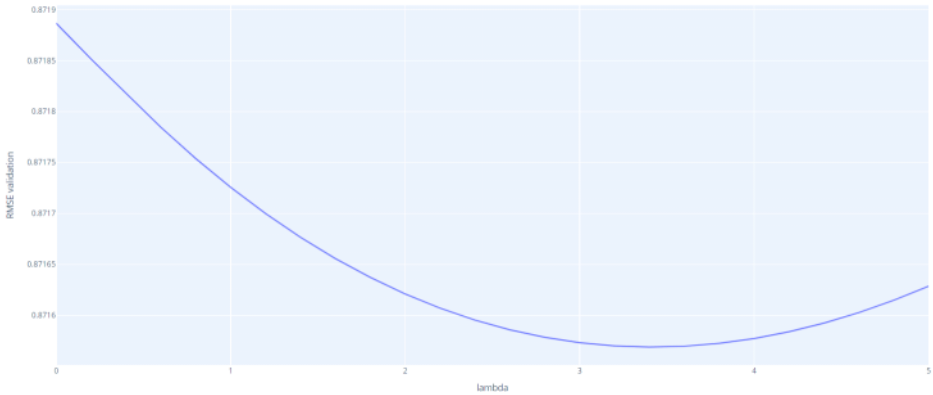
```python
def getA(training): #Edit by Alae
    A = np.zeros((trStats["n_ratings"], trStats["n_movies"] + trStats["n_users"]))
    for row in range(trStats["n_ratings"]):
        A[row][training[row][0]] = 1 #movie index
        A[row][training[row][1]+trStats["n_movies"]] = 1 #user index
    return A
```

The program works with no issues and we have only 1 *Hyperparameter to tune The regularization. So we try a bunch of values from* 0 *to* 5

Effect on Training Loss:



Effect on Validation Loss:



we see that the best value is reached at lambda $= 3.4$

Restricted BoltzMann Machine:

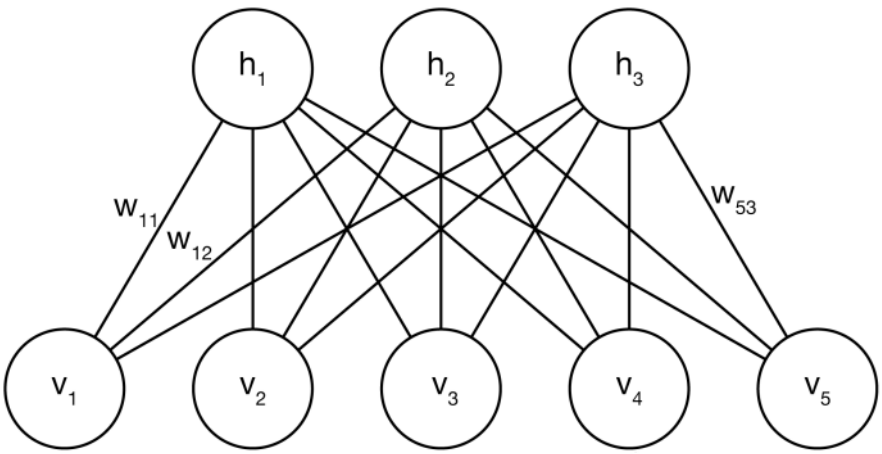
Mathematical view

it is a two layer network:

-the Visible Layer is the input layer

-the Hidden Layer receives a non-linear transformation of the input allowing it to make precise estimators

What we want to learn in this model: the weights $\boldsymbol{W}$ *connecting the two layers together*



How do we traverse the RBM in theory? Let's see the math

It all starts from the Visible Layer getting an input
then said input is passed through to the Hidden layer

$$\mathbb{P}(h_j = 1|\mathbf{v}) = \sigma\left(\sum_{k\in K}\sum_{i\in V} v_i^k W_{ij}^k\right)$$

$\sigma(x)$ is the sigmoid function
Bias Extension: *Adding biases is necessary to make model complete*

$$\mathbb{P}(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_{k\in K}\sum_{i\in V} v_i^k W_{ij}^k\right)$$

Next part is data ***generation***
we traverse the model backwards to reach from hidden to visible layer :

$$\mathbb{P}(v_i^k = 1|\mathbf{h}) = softmax\left(\sum_{j\in J} h_j W_{ij}^k\right)$$

Bias Extension:

$$\mathbb{P}(v_i^k = 1|\mathbf{h}) = softmax\left(b_i^k + \sum_{j\in J} h_j W_{ij}^k\right)$$

The next part is Learning the weights:
after going from visible to hidden
we compute $(PG)_{ijk} = \mathbb{P}(h_j = 1|\mathbf{v}) \cdot v_i^k$
traverse hidden to visible to get $\bar{v}$ *and then compute* :

$$(NG)_{i,j,k} = \mathbb{P}(h_j = 1|\overline{\mathbf{v}}) \cdot \overline{v}_i^k.$$

finally update W :

$$W \leftarrow W + \epsilon \cdot (PG - NG).$$

Extensions :

Regularization :
$W \leftarrow W + grad$
$grad = \epsilon \cdot [(PG - NG) - \lambda W]$

Momentum with bias:
$MomentumW = rate * momentumW + grad$
$Momentumbias = rate * Momentumbias + grad\ bias$
$W \leftarrow W + momentumW$
$bias \leftarrow bias + Momentumbias$

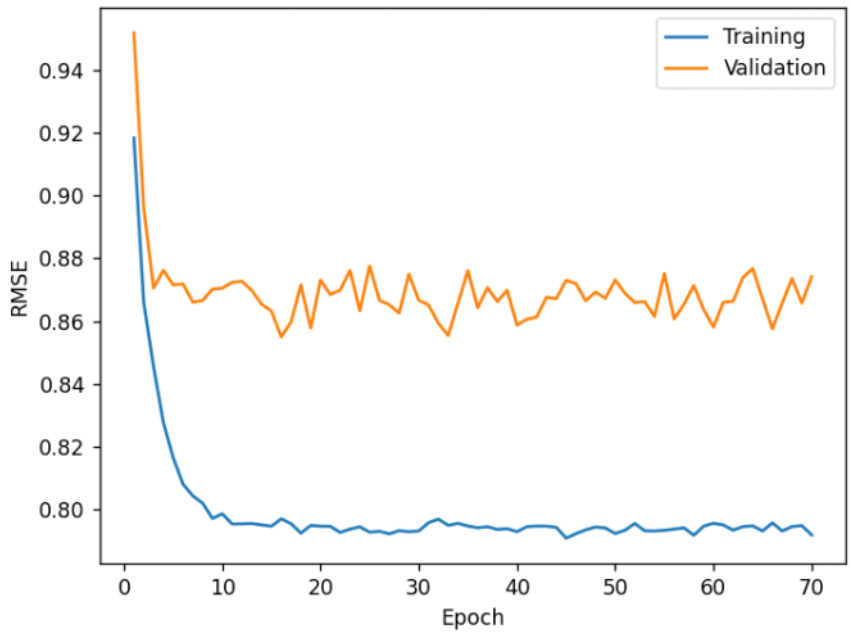
Implementation

Thankfully for this part we have been guided to first make a few basic sub-functions the source code implements all these sub functions and the additional extensions to be added
For example the functions traversing the neural net now also take the biases are parameters
These functions are then used to implement the theoretical process explained in the paragraph above
Some notable extensions are implemented through a coding solution rather than being introduced
into the math directly, for example our solution to early stopping is to keep track of the history of

$$\begin{aligned}&\text{the model throughout its execution epoch by epoch.}\\&\text{we keep a record of all the weight matrices } W\text{, all the losses for both train}\\&\text{and validation}\\&\text{in the end we can revisit the Weights that gave the best losses and neglect}\\&\text{the overfitting after it}\\&\text{The reason to not just totally stop is for analysis reasons, sometimes there}\\&\text{are interesting trends to see}\\&\text{As for the adaptive learning rate of the gradients, we use the same history}\\&\text{to check if the model}\\&\text{has been performing bad on the last few epochs, if there isnt much}\\&\text{improvement we lower the rate}\\&\text{In fact, we have also tested making the Regularization parameter}\\&\text{dynamic, increasing whenever}\\&\text{overfitting is detected}\begin{pmatrix}\text{good improvement on train but negative}\\\text{improvement on validation}\end{pmatrix}\end{aligned}$$

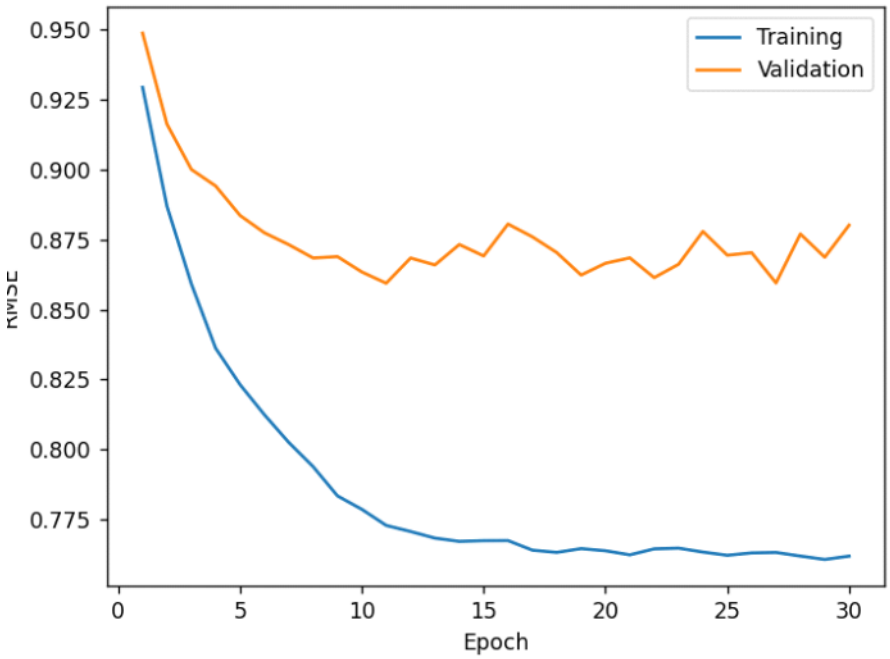
Results and Analysis

Now that the model is Implemented and working well, it's time to optimize the Hyper Parameters
Perhaps this is where the real work is done as a data scientist,
With the fact that there is some slight randomness in the model, it takes many runs to average out
And find the real trends behind the change of some parameters
First we must think how many epochs does the model need to converge on average ?



Here we see, on a non-overfitting model, going anything over 20-30 epochs is useless.
In fact after implementing the history and early stopping, we notice most of the times
The best value of the RMSE is found before 20 epochs
With that said, the gradient learning rate also plays a role on the convergence speed
An extremely slow rate is inefficient and hard to test as it takes forever to converge
But with higher rates (even 0.01) we can get into overfitting easily (depends on the complexity of the model with comes with higher number of hidden nodes)

Here we notice an overfitting model, even at 30 epochs it is still getting more used to the training dataset while the results on the validation only get worse. By implementing an adaptive learning rate with other variables kept intact we can see a large difference :



Here we see the validation keeps going down for longer until about 12 epochs where both the training and validation plateau so we know this is not overfit

Analysis on the number of hidden units:

Perhaps the most impactful parameter theoretically, the more hidden units we have the more the model becomes complex, this is a double edged sword, because it can understand more detailed nuances of the data but it also fails to generalize to unseen datasets. With out dataset not being too large, we have to be very careful of overfitting. So we try many values of F with other parameters fixed (in reality we should dynamically change some parameters like higher Regularization with higher F but the time is too short and the possibilities are limitless)

F (Hidden units)	10	20	25	30	40	80
Train RMSE	0.820	0.803	0.763	0.801	0.733	0.691
Valid RMSE	0.852	0.854	0.853	0.879	0.870	0.886

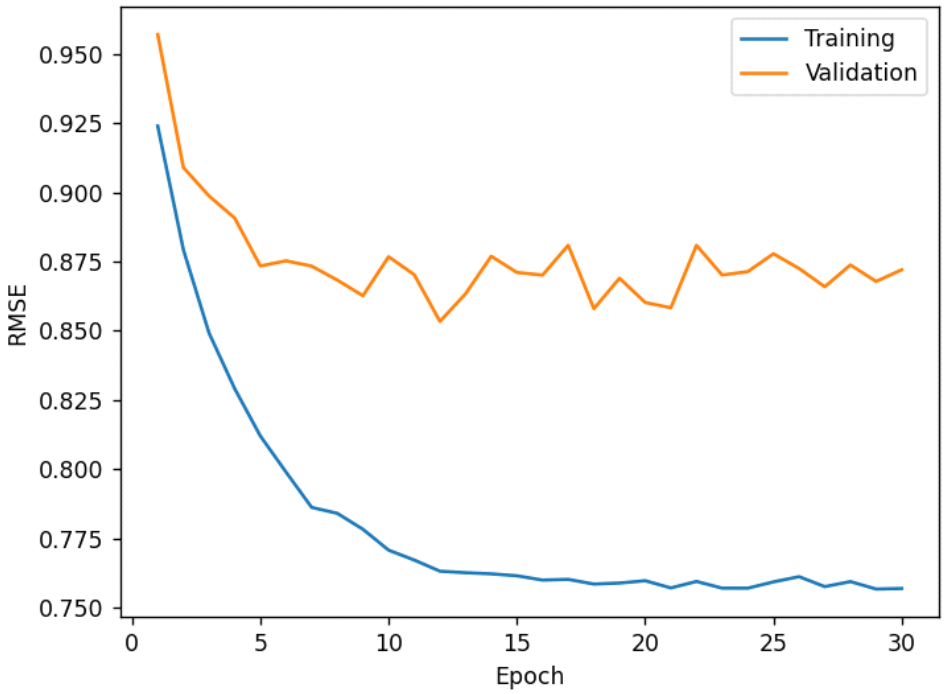
For the rest of the Analysis we fix F=25

Analysis on regularization coefficient:

Lambda	0	0.1	0.5	1	Adaptive (0.05 to 1 max)
Train RMSE	0.772	0.763	0.768	0.769	0.7790
Valid RMSE	0.856	0.853	0.867	0.859	0.8735

The adaptive lambda seems like a good idea but the execution is still a little flawed so I prefer to just Keep lambda at 0.1

The best model with all parameters taken into account :



Conclusion:

In conclusion, embarking on the journey of developing an RBM model for predicting movie ratings, akin to the challenges posed by the Netflix problem in collaborative filtering, has been both a rewarding and enlightening experience. This project not only allowed us to delve into the intricacies of a sophisticated model but also provided invaluable insights into the nuanced realm of mitigating overfitting challenges.

Beyond the technical aspects, the project has proven to be a valuable asset for our professional development. The complexities encountered mirror real-world scenarios, where the unpredictability of a hidden test set parallels the uncertainty one often faces when implementing solutions for genuine applications. Navigating through these challenges has not only honed our technical skills but also underscored the importance of adaptability and resilience in the face of ambiguity.

Undoubtedly, the successful execution of this project contributes significantly to our skill set and makes for a compelling addition to our curriculum vitae. The lessons learned extend far beyond the confines of this endeavor, providing a solid foundation for tackling more intricate problems in the exciting world of machine learning and predictive modeling. As we reflect on this endeavor, we can confidently assert that the knowledge gained and the experiences accrued will serve as guiding beacons in our future pursuits within the dynamic and ever-evolving field of data science.