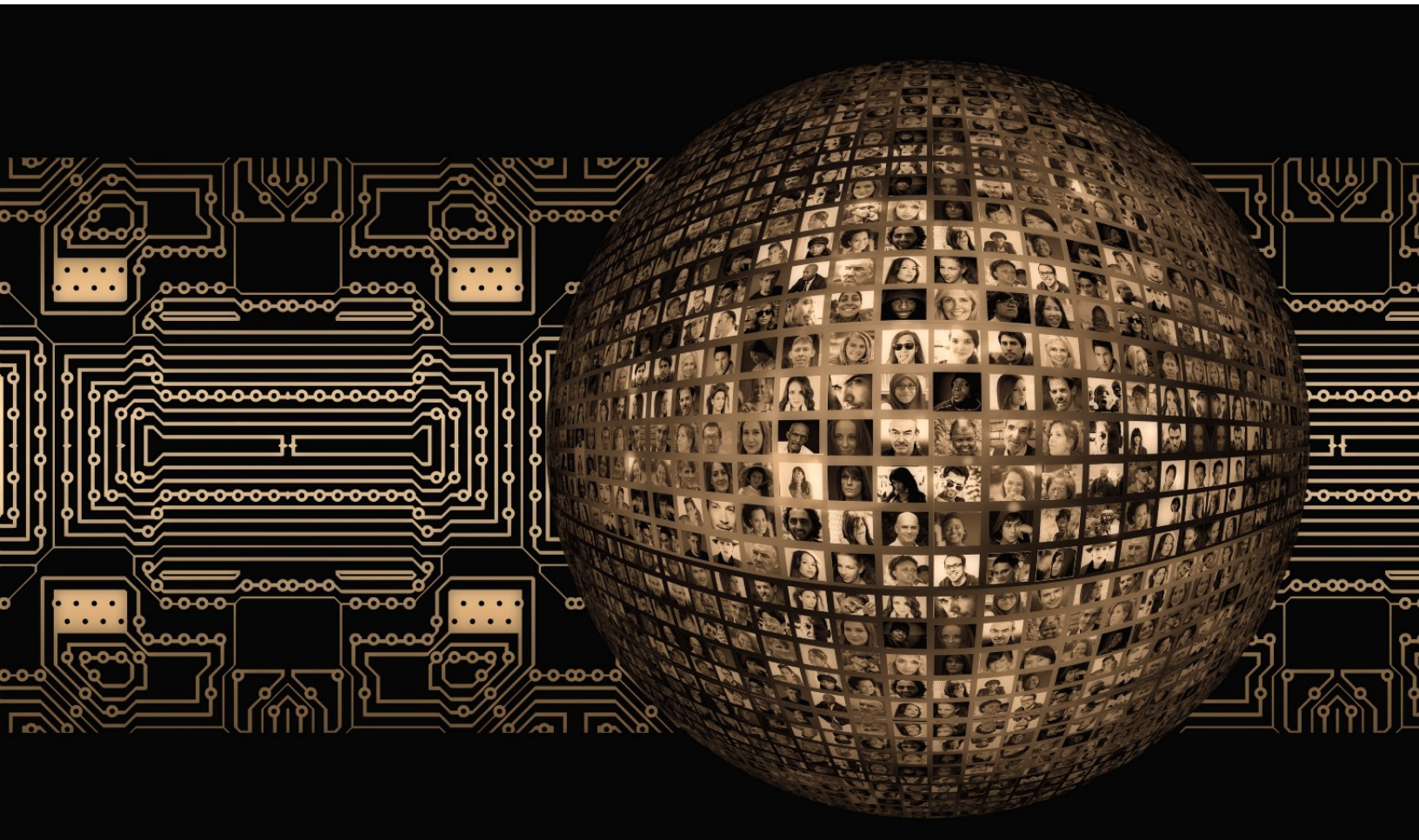
KVPY summer project - Auto-encoder for Online Recommendation using Implicit Feedback

Kushagra Gupta, KVPY reg no. SX2016/X25130053



SUMMER RESEARCH INTERNSHIP, NEW YORK OFFICE, IIT KANPUR

GITHUB.COM/AVIK-PAL/HYBRID_*ecommender*

THIS PROJECT WAS DONE UNDER THE SUPERVISION OF DR. MANINDRA AGARWAL WITHIN A TOTAL OF 8 WEEKS, FROM MAY 14TH TO JULY 14TH OF 2018.

Contents

1 Introduction 5
1.1 Motivation 5
1.2 A bit of context 5

2 Model 7
2.1 Loss function 7
2.2 Dense re-feeding 8

3 Experiments and Results 9
3.1 Experiment Setup 9
3.2 Effects of Activation Types 9
3.3 Over-fitting the data 9
3.4 Going deeper 12
3.5 dropout 12
3.6 Comparison with other methods 12

4 Conclusion 17
4.1 Submission 18

1. Introduction

1.1 Motivation

Sites like Amazon, Netflix and Spotify use recommender systems to suggest items to users. Recommender systems can be divided into two categories: context-based and personalized recommendations.
Context based recommendations take into account contextual factors such as location, date and time . Personalized recommendations typically suggest items to users using the collaborative filtering (CF) approach. In this approach the user's interests are predicted based on the analysis of tastes and preference of other users in the system and implicitly inferring "similarity" between them. The underlying assumption is that two people who have similar tastes, have a higher likelihood of having the same opinion on an item than two randomly chosen people.
In designing recommender systems, the goal is to improve the accuracy of predictions.
Structure of a classic CF problem: Infer the missing entries in an m*n matrix, R , whose (i , j) entry describes the ratings given by the i th user to the j th item. The performance is then measured using Root Mean Squared Error (RMSE).

1.2 A bit of context

Deep learning has led to breakthroughs in image recognition, natural language understanding, and reinforcement learning. Naturally, these successes fuel an interest for using deep learning in recommender systems. First attempts at using deep learning for recommender systems involved restricted Boltzman machines (RBM). Several recent approaches use autoencoders, feed- forward neural networks and recurrent recommender networks . Many popular matrix factorization techniques can be thought of as a form of dimensionality reduction. It is, therefore, natural to adapt deep autoencoders for this task as well. I-AutoRec (item- based auto-encoder) and U-AutoRec (user-based auto encoder) are first successful attempts to do so.
there are many non deep learning types of approaches to collaborative filtering (CF). Matrix factorization techniques, such as alternating least squares (ALS) and probabilistic matrix factorization are particularly popular. Several classic CF approaches has been extended to incorporate temporal

information such as TimeSVD++, as well as more recent RNN-based techniques such as recurrent recommender networks.

2. Model

The model I was working on was inspired by U-AutoRec approach with several important distinctions.I worked with much deeper models.To enable this without any pretraining, I: a) used “scaled exponential linear units” (SELUs), b) used high dropout rates, and d) used iterative output re-feeding during training. An autoencoder is a network which implements two transformations - encoder encode (x) : R n -> R d and decoder (z) : R d -> R n . the “goal” of autoencoder is to obtain d dimensional representation of data such that an error measure between x and f (x) = decode (encode (x)) is minimized. If noise is added to the data during encoding step, the autoencoder is called de-noising . Autoencoder is an excel- lent tool for dimensionality reduction and can be thought of as a strict generalization of principle component analysis (PCA). An autoencoder without non-linear activations and only with “code” layer should be able to learn PCA transformation in the encoder if trained to optimize mean squared error (MSE) loss. In the model, both encoder and decoder parts of the autoencoder consisted of feed-forward neural networks with classical fully connected layers computing l = f (W x + b) , where f is some non-linear activation function. If range of the activation function is smaller than that of data, the last layer of the decoder should be kept linear.It was very important for activation function f in hidden layers to contain non-zero negative part, and using SELU units in most of the experiments. If decoder mirrors encoder architecture, then one can constrain decoder’s weights W l d to be equal to trans- posed encoder weights W l e from the corresponding layer l . Such autoencoder is called constrained or tied and has almost two times less free parameters than unconstrained one.Forward pass and inference . During forward pass (and inference) the model takes user represented by his vector of ratings from the training set x ε R n , where n is number of items. Note that x is very sparse, while the output of the decoder, f (x)ε R n is dense and contains rating predictions for all items in the corpus.

2.1 Loss function

Since it doesn’t make sense to predict zeros in user’s representation vector x, I followed the approach from and optimize Masked Mean Squared Error loss: where r i is actual rating, y i is reconstructed, or predicted rating, and m i is a mask function such that m i = 1 if r i , 0 else m i = 0. Note that there

$$MMSE = \frac{m_i * (r_i - y_i)^2}{\sum_{i=0}^{i=n} m_i}$$

Figure 2.1: masked mean square

is a straightforward relation between RMSE score and MMSE score: RMSE = $\sqrt{}$ MMSE .

2.2 Dense re-feeding

During training and inference, an input x R n is very sparse because no user can realistically rate but a tiny fractions of all items. On the other hand, autoencoder's output f (x) is dense. If we consider an idealized scenario with a perfect f . Then f (x) i = x i , i : x i , 0 and f (x) i accurately predicts all user's future ratings for items i : x i = 0. This means that if user rates new item k (thereby creating a new vector x) then f (x) k = x k and f (x) = f (x) . Hence, in this idealized scenario, y = f (x) should be a fixed point of a well trained autoencoder: f (y) = y . To explicitly enforce Fixed-point constraint and to be able to perform dense training updates, I had to augmented every optimization iteration with an iterative dense re-feeding steps (3 and 4 below) as follows:

1. Given sparse x , compute dense f (x) and loss using equation 1 (forward pass)
2. Compute gradients and perform weight update (backward pass)
3. Treat f (x) as a new example and compute f (f (x)) . Now both f (x) and f (f (x)) are dense and the loss from equation 1 has all m as non-zeros. (second forward pass)
4. Compute gradients and perform weight update (second backward pass).

Steps (3) and (4) can be also performed more than once for every iteration.

3. Experiments and Results

3.1 Experiment Setup

For the rating prediction task, it is often most relevant to predict future ratings given the past ones instead of predicting ratings missing at random. Training interval contains ratings which came in earlier than the ones from testing interval. Testing interval is then randomly split into Test and Validation subsets so that each rating from testing interval has a .5 probability chance of appearing in either subset. Users and items that do not appear in the training set are removed from both test and validation subsets. For most of our experiments I used a batch size of 128, trained using SGD with momentum of 0.9 and learning rate of 0.001. I used xavier initialization to initialize parameters. I did not use any layer-wise pre-training.

3.2 Effects of Activation Types

To explore the effects of using different activation functions, I tested some of the most popular choices in deep learning : sigmoid, "rectified linear units" (RELU), max (relu (x) , 6) or RELU6, hyperbolic tangent (TANH), "exponential linear units" (ELU), leaky relu (LRELU) [20] , and "scaled exponential linear units" (SELU) on the 4 layer autoencoder with 128 units in each hidden layer. Because ratings are on the scale from 1 to 5, I kept last layer of the decoder linear for sigmoid and tanh-based models. In all other models activation function is applied in all layers. I found that on this task ELU, SELU and LRELU perform much better than SIGMOID, RELU, RELU6 and TANH. There are two properties which separate activations which perform well from those which do not: a) non-zero negative part and b) unbounded positive part. Hence, in this setting these properties are important for successful training. Thus, I used SELU activation units and tune SELU-based networks for performance

3.3 Over-fitting the data

The largest data set used for training, "Netflix Full" contains 98M ratings given by 477K users. Number of movies (e.g. items) in this set is n = 17 , 768. Therefore, the first layer of encoder had

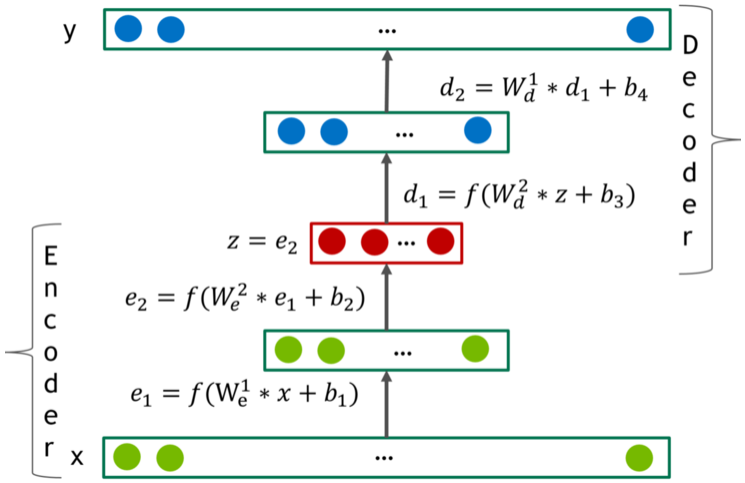


Figure 3.1: AutoEncoder consists of two neural networks, en- coder and decoder, fused together on the “representation” layer z . Encoder has 2 layers e 1 and e 2 and decoder has 2 layers d 1 and d 2 . Dropout may be applied to coding layer z .

	Full	3 months	6 months	1 year
Training	12/99-11/05	09/05-11/05	06/05-11/05	06/04-05/05
Users	477,412	311,315	390,795	345,855
Ratings	98,074,901	13,675,402	29,179,009	41,451,832
Testing	12/05	12/05	12/05	06/05
Users	173,482	160,906	169,541	197,951
Ratings	2,250,481	2,082,559	2,175,535	3,888,684

Figure 3.2: Subsets of Netflix Prize training set used in the experiments.

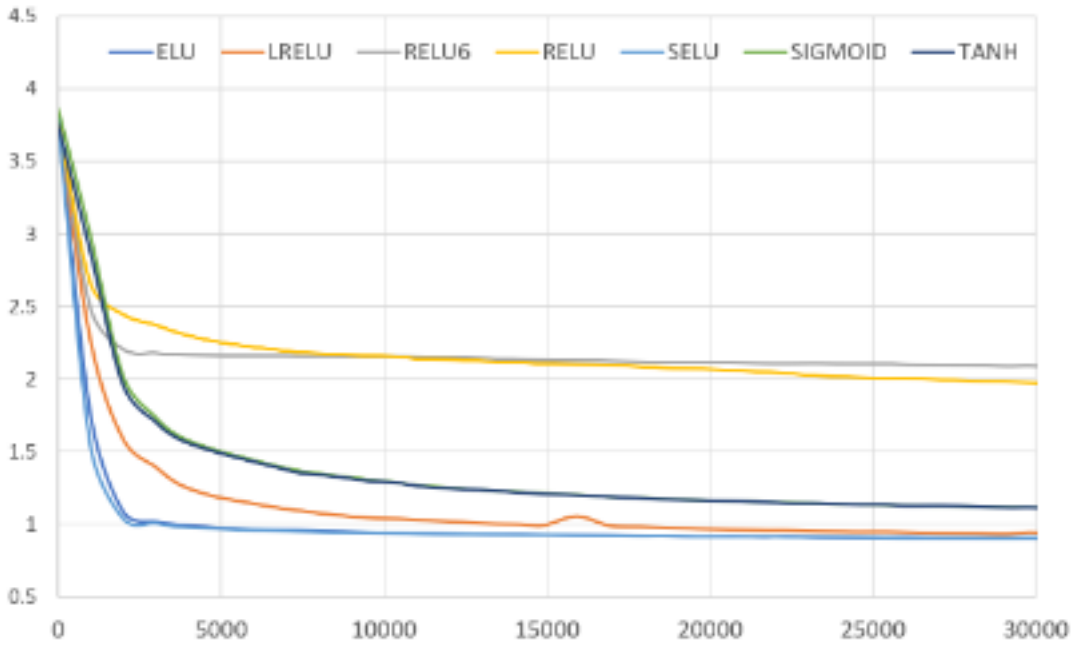


Figure 3.3: Training RMSE per mini-batch. All lines correspond to 4-layers autoencoder (2 layer encoder and 2 layer decoder) with hidden unit dimensions of 128. Different line colors correspond to diferent activation functions. TANH and SIGMOID lines are very similar as well as lines for ELU and SELU.

d n + d weights, where d is number of units in the layer. For modern deep learning algorithms and hardware this is relatively small task. If we start with single layer encoders and decoders we can quickly over fit to the training data even for d as small as 512. Switching from unconstrained autoencoder to constrained reduces over-fitting, but does not completely solve the problem.

3.4 Going deeper

While making layers wider helps bring training loss down, adding more layers is often correlated with a network's ability to generalize. In this set of experiments, this is indeed the case. By choosing small enough dimensionality (d = 128) for all hidden layers, it became easy to avoid over-fitting and start adding more layers. There is a positive correlation between the number of layers and the evaluation accuracy. Going from one layer in encoder and decoder to three layers in both provided good improvement in evaluation RMSE (from 1.146 to 0.9378). After that, blindly adding more layers did help, however it provided diminishing returns. Note that the model with single d = 256 layer in encoder and decoder had 9,115,240 parameters which is almost two times more than any of these deep models while having much worse evaluation RMSE (above 1.0).

3.5 dropout

I concluded that adding too many small layers eventually hits diminishing returns. Thus, I began working with model 3 architecture and hyper-parameters more broadly. The most promising model had the following architecture: n , 512 , 512 , 1024 , 512 , 512 , n , which meant 3 layers in encoder (512,512,1024), coding layer of 1024 and 3 layers in decoder of size 512,512,n. This model, however, quickly over-fitted if trained with no regularization. To regularize it, I used several dropout values and, interestingly, very high values of drop probability (e.g. 0.8) turned out to be the best. I applied dropout on the encoder output only, e.g. f (x) = decode (dropout (encode (x))). I also tried applying dropout after every layer of the model but that stifled training convergence and did not improve generalization.

3.6 Comparison with other methods

I compared the best model with Recurrent Recommender Network from which it has been shown to outperform PMF,T-SVD and I/U-AR on the data I used. Note, that unlike T-SVD and RRN, the method does not explicitly take into account temporal dynamics of ratings.Yet, it is still capable of outperforming these methods on future rating prediction task. I trained each model using only the training set and compute evaluation RMSE for 100 epochs. Then the checkpoint with the highest evaluation RMSE was tested on the test set. "Netflix 3 months" had 7 times less training data compared to "Netflix full",therefore, the model's performance was significantly worse if trained on this data alone (0.9373 vs 0.9099). In fact, the model that performs best on "Netflix full" overfits on this set, and I had to reduce the model's complexity accordingly.

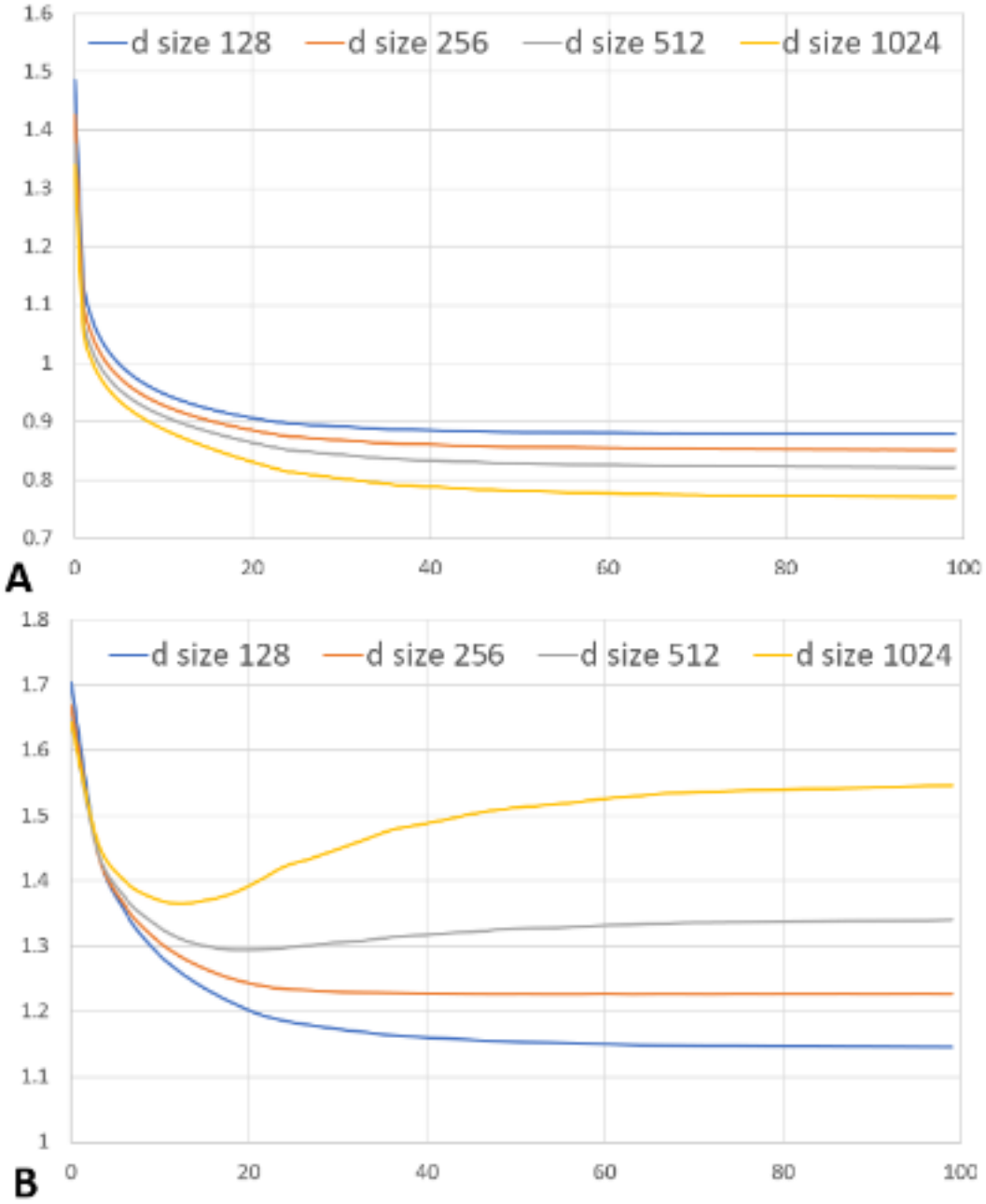


Figure 3.4: Single layer autoencoder with 128, 256, 512 and 1024 hidden units in the coding layer. A: training RMSE per epoch; B: evaluation RMSE per epoch

Number of layers	Evaluation RMSE	params
2	1.146	4,566,504
4	0.9615	4,599,528
6	0.9378	4,632,552
8	0.9364	4,665,576
10	0.9340	4,698,600
12	0.9328	4,731,624

Figure 3.5: Depth helps generalization. Evaluation RMSE of the models with different number of layers. In all cases the hidden layer dimension is 128.

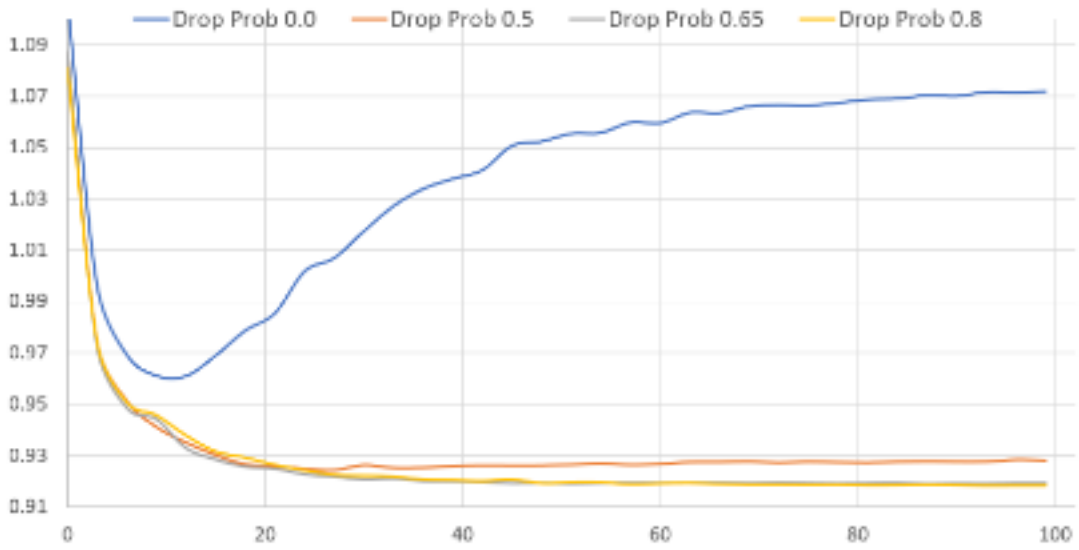


Figure 3.6: Effects of dropout. Y-axis: evaluation RMSE, X- axis: epoch number. Model with no dropout (Drop Prob 0.0) clearly over-fits. Model with drop probability of 0.5 over-fits as well (but much slowly). Models with drop probabilities of 0.65 and 0.8 result in RMSEs of 0.9192 and 0.9183 correspondingly.

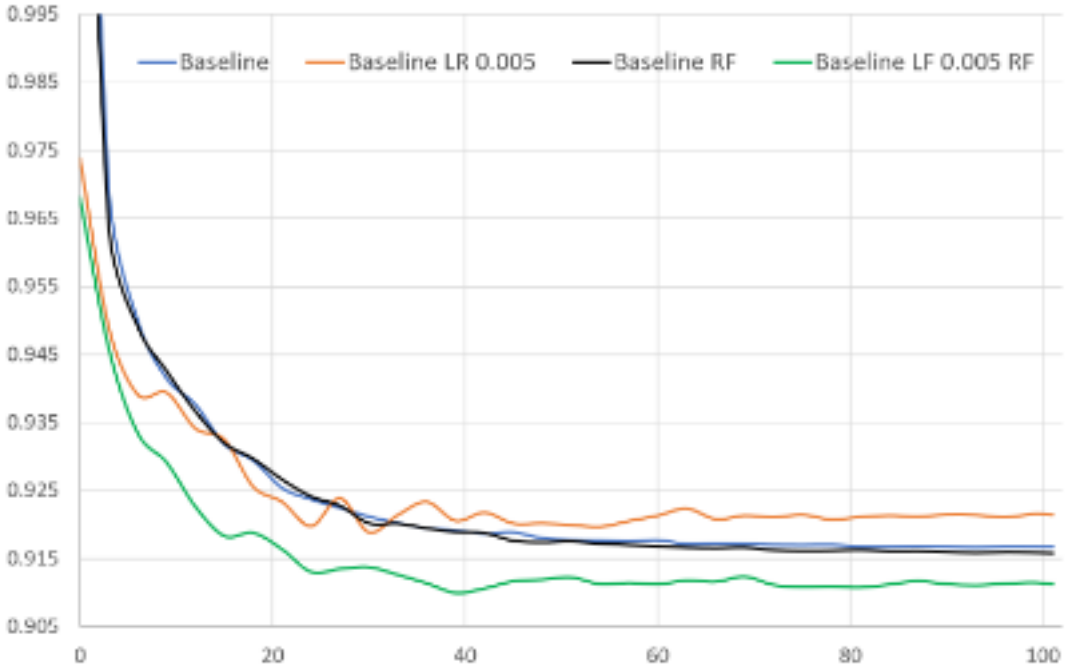


Figure 3.7: Effects of dense re-feeding. Y-axis: evaluation RMSE, X-axis: epoch number. Baseline model was trained with learning rate of 0.001. Applying re-feeding step with the same learning rate almost did not help (Baseline RF). Learning rate of 0.005 (Baseline LR 0.005) is too big for baseline model without re-feeding. However, increasing both learning rate and applying re-feeding step clearly helps (Baseline LR 0.005 RF).

DataSet	I-AR	U-AR	RRN	DeepRec
Netflix 3 months	0.9778	0.9836	0.9427	**0.9373**
Netfix Full	0.9364	0.9647	0.9224	**0.9099**

Figure 3.8: Test RMSE of different models.

4. Conclusion

Deep learning has revolutionized many areas of machine learning, and it is poised do so with recommender systems as well. Through replicating this experiment, I realised how very deep autoencoders can be successfully trained even on relatively small amounts of data by using both well established (dropout) and relatively recent ("scaled exponential linear units") deep learning techniques. Further, working on iterative output re-feeding - a technique which allowed me to perform dense updates in collaborative filtering, increase learning rate and further improve generalization performance of the model. On the task of future rating prediction, the model out- performed other approaches even without using additional temporal signals.
While the code supported item-based model (such as I-AutoRec) this approach is less practical than user-based model (U- AutoRec). This is because in real-world recommender systems, there are usually much more users then items. Finally, when building personalized recommender system and faced with scaling problems, it can be acceptable to sample items but not users.

DataSet	RMSE	Model Architecture
Netflix 3 months	**0.9373**	n, 128, 256, 256, $dp(0.65)$, 256, 128, n
Netflix 6 months	**0.9207**	n, 256, 256, 512, $dp(0.8)$, 256, 256, n
Netflix 1 year	**0.9225**	n, 256, 256, 512, $dp(0.8)$, 256, 256, n
Netfix Full	**0.9099**	n, 512, 512, 1024, $dp(0.8)$, 512, 512, n

Figure 4.1: Test RMSE achieved by Deep Rec on different Netflix subsets. All models are trained with one iterative output re- feeding step per each iteration.

4.1 Submission

Submitted by

Kushagra Gupta
KVPY Reg no. SX2016/X25130053
BS Mathematics, 2 year
170358, IIT Kanpur

Submitted to

Prof Manindra Agarwal
CSE, IIT Kanpur

Mentored by

Adarsh Jagannatha
CSE, IIT Kanpur