Attentive Neural Collaborative Filtering

Clemens Hutter, Rhea Sukthanker, Georgios Vasilakopoulos, Nina Wiedemann

group: Gryffindor, Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Many platforms aim to personalize their content, such that their users receive suggestions and offers inline with their preferences. A prominent example is the case of streaming platforms, that try to recommend movies to a user based on his ratings of other movies. State-of-the-art methods often embed user and movie into a vector representation, that is usually a look-up in a learned matrix. We firstly propose to initialize this Matrix more efficiently using user and movie representations obtained from the latent space of an auto encoder. Secondly we have developed a novel method to embed users based on other movies they have rated, which is superior to other models not only in accuracy, but also with respect to parameter efficiency and explainability. We show that this model together with different prediction models outperforms previous methods.**

I. Introduction

One of the most scary and yet the most promising advances in Machine Learning is the possibility to learn about people's feelings and preferences, and provide personalized content. As the well-known historian Yuval Noah Harari puts it, algorithms might soon "get to know you better than you know yourself"[1]. In the entertainment industry, we can already feel this development, when Spotify suggests a song one really likes or Netflix recommends movies according to your taste. High performing recommender-systems can provide better user experience, saving the user the time needed to dig through the large amount of available movies themselves.

Common approaches to this problem include K-Nearest Neighbors filtering, Singular Value Decomposition([1]–[3]) and Matrix Factorization methods [4], [5]. Recently different deep learning architectures have also been proposed targeting this problem as effectively summarized in [6]. Often, the item or user is embedded into a lower dimensional feature space (an embedding), which captures only the most relevant features for user preference prediction. An autoencoder (AE) for example can learn such embeddings, when trained to reconstruct the data based on the information preserved in a bottleneck layer. Autoencoder was first proposed for the use in recommender systems by [7] and multiple variants including sparse AEs [8], variational AEs [9] and denoising AEs exist today.

Though the autoencoder outperforms most of the traditional models it is less interpretable. We desire that the output (a movie recommendation) is more intuitive and explainable. For example we want to interpret how a higher rating for one movie by a user implies a higher rating for an other. To provide such information, attention-based deep learning models have been used extensively for Speech Recognition [10] and Machine Translation [11]. Inspired by [12] we have developed an approach that weights the influence of other movies and thereby constructs an attention-embedding.

In addition, we discovered that instead of using one of these embedding models on its own, performance can be significantly improved when we incorporate the pre-embeddings into a different and more powerful prediction architecture, namely a deep Feed Forward Neural Network (FFNN). While in some previous work such as [13] the rating is predicted on the raw (sparse) information for a user-item pair, others learn low dimensional embeddings for users and items first, and use effective feature composition techniques to get a feature representation which is then fed to a FFNN [14]. Our prediction model is loosely based on this architecture, but the user and movie embeddings are obtained differently, i.e. pretrained with AE or attention based. In the following sections we will dive into the details of our prediction and embedding models. We conclude that amongst all the models we have tried the combination of pre-trained attention embeddings and MLP2 (described later) achieves the best performance.

II. Methods

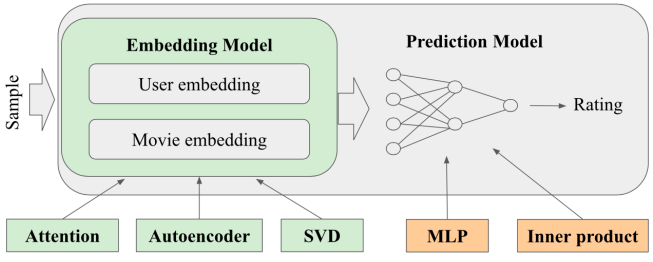

Figure 1. General framework structure: An embedding model serves as the input layer of a prediction model, for example an MLP, that infers the rating from user and item embedding. Here, we propose to use a pretrained Autoencoder or our novel Attention embedding.

Mainly, the proposed model consists of two parts, as depicted in in Fig.1. The prediction model computes an

[1]https://towardsdatascience.com/yuval-noah-harari-and-fei-fei-li-on-ai-90d9a8686cc5

estimate $\widehat{r}_{ki}$ of the rating r_{ki} of user k for movie i, based on the embeddings u_k and m_i which are inferred from the embedding model. Both will be explained in detail in the following section.

A. Embedding Models

1) Attentive User/Item Embedding

We propose a novel way to calculate a user embedding, which can later be used in combination with any embedding based prediction model. The approach is motivated by the assumption that users have different preference structures for different genres or types of movies. For example, when predicting a user's rating for a horror movie, it is arguably not relevant what kind of romantic comedies she prefers. Thus it makes sense to construct an adhoc user embedding based on those movies she interacted with, that are similar to the target movie for which we are trying to predict the rating.

To describe the approach formally, let $E \in \mathbb{N}$ be the embedding size. Suppose for a target movie $i \in \{1 \dots N_{\text{movies}}\}$ and a user $k \in \{1 \dots N_{\text{user}}\}$ we want an embedding, that can be used to describe how much the user k likes the movie i. Let $C(k) \subseteq \{1 \dots N_m\}$ be the set of other movies that user k has rated and let $r_{hj} \in \{1, 2, 3, 4, 5\}$ be the rating the user h gave to movie j. Further we define two trainable embedding matrices $M \in \mathbb{R}^{N_{\text{movies}} \times E}$ and $T \in \mathbb{R}^{N_{\text{movies}} \times E}$ where each row (denoted by m_i and t_j) is an embedding vector corresponding to a movie. Semantically, m_i represents the structure of the movie i while t_j represents what kind of users like movie j (i.e. what liking movie j says about a user).

Now we construct the user embedding u_{ki} for the movie/user pair (i, k) according to

$$u_{ki} = \sum_{j \in C(k)} att(i, j) \cdot \widetilde{r_{kj}} \cdot t_j \tag{1}$$

where $\widetilde{r_{kj}} = (r_{kj} - 3)/2$ (i.e. the rating rescaled to $[-1, 1]$). $att(i, j)$ is a function that computes how much attention should be given to the context movie j given the target movie i.

Intuitively, t_j can be seen as a vector encoding the personality traits of users that like movie j. If user k liked the movie j we therefore add this vector to the embedding for user k with magnitude proportional to the degree to which she liked it (i.e. higher rating above 3 stars). If user k disliked movie j we subtract the vector with the magnitude proportional to the degree to which she disliked the movie (i.e. lower rating below 3 stars). As a result the ad-hoc user embedding u_{ki} will be similar to the t_j of movies she liked and dissimilar to the t_j of movies she did not like.
This, however, is done such that more weight (or attention) is given to context movies j that are similar to the target movie i, for which we are computing a prediction.

While many choices for the attention function $att(i, j)$ are possible, we decided for 'scaled-dot product attention' [15], with the corresponding movie structure embeddings m_i and m_j to represent the query and the keys respectively:

$$a_{ij} = \frac{m_i^T m_j}{\sqrt{E}} \quad \text{for } j \in C(k)$$
$$att(i, j) = \frac{e^{a_{ij}}}{\sum_{j' \in C(k)} e^{a_{ij'}}}$$

In practice it is not feasible to compute the sum in Equation 1 for the whole set $C(k)$ of other movies a user has watched, as this might be to big. Thus we replace $C(k)$ by a subset $\widehat{C}(k)$ which contains N_{context} randomly sampled movies (without replacement) out of $C(k)$. We set N_{context} to 64 during training and to 128 during testing. If $|C(k)| \leq N_{\text{context}}$ then $\widehat{C}(k) = C(k)$. If a user has not rated any other movie yet (i.e. $C(k) = \emptyset$) we replace u_{ki} from (1) with a learned default user embedding $u_{\text{default}} \in \mathbb{R}^E$. That is, all users for which we do not know other ratings will have the same embedding vector u_{default}.

The novel user embedding proposed here is somewhat similar to [12], where user embeddings are also constructed based on an attention distribution over other items a user has interacted with. However, the attention mechanism in [12] does not depend on the target item i, for which the embedding is going to be used and will thus be constant for a user. Further they rely on a $N_{\text{user}} \times E$ user embedding matrix.

Thus our approach has the following advantages:

- *Explainability:* Suppose a new movie is recommended to a user, based on a high predicted rating. This prediction - specifically the user embedding that was used to generate it - is associated with an attention distribution over other movies the user has watched before. Thus the recommendation can be explained to the user, by listing those previously watched movies that had high attention values.
- *Data Augmentation:* A user's embedding will be different every time it is computed for a user and movie pair, firstly because it depends on the target movie, and secondly because it is computed based on only a random subset of other movies the user has watched. Thus our attentive embedding mechanism has built in data augmentation which can guard against overfitting.
- *Parameter Efficiency:* Traditional look up embbeddings need an $N_{\text{movies}} \times E$ movie embedding matrix and a $N_{\text{user}} \times E$ user embedding matrix. Our approach requires two $N_{\text{movies}} \times E$ matrices, which results in much fewer learnable parameters as N_{movies} is usually much smaller than N_{user}. For the data set used here with $N_{\text{user}} = 10,000$ and $N_{\text{movies}} = 1,000$ with $E = 64$ this is a significant reduction from $10,000 \cdot 64 + 1,000 \cdot 64 = 704,000$ to $2 \cdot 1,000 \cdot 64 = 128,000$ parameters.

- *Adding new users:* In our approach the user embedding is computed based on known ratings of the user for other movies. If a new user signs up and we happen to know some ratings of the user already (e.g. through an initial questionnaire) we can directly obtain ad hoc embeddings for the new user without any expensive retraining of the network. This posses an advantage over a user identity based embedding (e.g. [12]), which would require retraining the network with an additional row in the user embedding matrix for the new user.

As movie embedding for the movie/user pair i/k we simply use a traditional look-up in the movie structure embedding matrix M, that is m_i.

2) Autoencoder

An AE is a bottleneck architecture that is trained to encode the input in a lower dimensional representation, while optimizing the reconstruction quality. In the following, only the user-AE for collaborative filtering will be explained, but the movie-AE is similar by symmetry. A single data sample for training a user-AE corresponds to the data of a single user v_k, which is a vector of ratings for each movie ($v_{kj} = 0$ if user k has not rated movie j yet). Then, a function f encodes v_k in a latent space $z = f(v_k)$, from where it is decoded with another function g. This way, the input user vector v is reconstructed as $\hat{v} = g(z) = g(f(v_k))$.

In the process of training, the latent space z is optimized to capture as much information as possible, by minimizing the reconstruction loss. However, in the special case of recommender systems, this loss can only be computed on the observed ratings of the user. With $C(k)$ again defined as the set of movies that user k has rated, the loss is defined as the RMSE over the observed ratings:

$$L(\hat{v_k}) = \sqrt{\frac{1}{|C(k)|} \sum_{j \in C(k)} (v_{kj} - \hat{v_{kj}})^2} \tag{2}$$

Here, deep AEs are used, such that f and g both consist of two feed forward layers. Particularly, the user AE is a four layered FFNN of sizes 512, 128, 512 and 1000 neurons respectively. All layers except for the output layer are followed by ReLU activation. Regarding the movie AE on the other hand, best performance could be achieved by a four layered network with the inner layers all containing 128 neurons. Sigmoid activation is added after each layer including the last one. In the beginning, the ratings are scaled to values between zero and one (rating 1 corresponds to 0.2, 2 to 0.4 ... 1 to 5) to provide the network with normalized inputs. Thus, the loss can simply be computed between these scaled inputs and the output of the sigmoid function.

B. Prediction Models

In this section we briefly discuss our Prediction models. Let $\mathcal{S} \subseteq \{1 \dots N_{\text{user}}\} \times \{1 \dots N_{\text{movies}}\}$ be the set of all training examples each consisting of a user k and a movie i. For a sample $(k, i) \in \mathcal{S}$ each Prediction model receives as input a movie embedding m_i and a user embedding u_k. These embeddings are computed according to one of the embedding methods detailed in section II-A. Note that u_k also depends on i if we use the *attentive embedding* from section II-A1.

We experiment with three prediction models. The first one is based on the inner product between the user and movie embeddings. The second two are MLP variants as displayed in Figure 2.

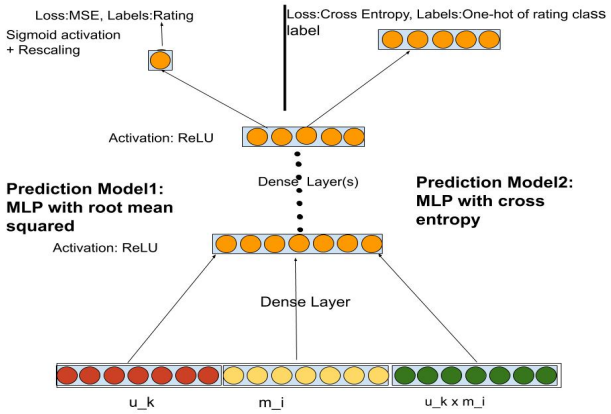

Figure 2. General structure of prediction models: The embeddings are concatenated with each other as well as their product. Two FFNN models are proposed, mainly differing in their loss function

1) Simple Inner Product Prediction

Given a user embedding u_k and a movie embedding m_i for a sample (k, i), this simple model computes the predicted rating according to

$$p_{ki} = \sigma(u_k^T m_i) \tag{3}$$

$$\hat{r}_{ki} = (p_{ki} \cdot 4) + 1 \tag{4}$$

where σ is the sigmoid function. p_{ki} can be interpreted as the probability that user k likes movie i, which is mapped to the range of the rating from 1 to 5. Then the per sample squared difference between predicted and actual rating is minimized using Adam [16].

2) MLP with RMSE loss: MLP1

Our first MLP constructs a feature composition $x_{ki,0}$ from the corresponding user embedding u_k and movie embedding m_i obtained from the Embedding models, as well as their element-wise product $u_k \cdot m_i$:

$$x_{ki,0} = [u_k, m_i, u_k \cdot m_i]$$

$x_{ki,0}$ is then passed through two feedforward layers with ReLU activation. The FFNN terminates into a single neuron which uses sigmoid activation with rescaling (similar to equation (3) and (4)) to get the desired predictions. We found the setting with two hidden layers of 200 and 100 neurons to be optimal.

Further we optimize over the Mean Squared Error of the predicted rating. With r_{kj} as the ground truth rating of user k for movie j, and $f(x_{ki})$ the prediction by the FFNN

$$Loss = \sum_{(k,i)\in\mathcal{S}} (r_{ki} - f(x_{ki}))^2$$

3) MLP with cross-entropy loss: MLP2

Last, another version of the MLP described above is obtained by inreasing the number of hidden layers and also changing the loss function and the predictions. A detailed architecture graph can be found in Appendix section. We mainly define the task as a classification problem with labels corresponding to the one-hot-encoded rating class $(1, 2, 3, 4, 5)$. We use the same input composition and projection layers, with the last layer consisting of 5 neurons and softmax activation. Thus, n_c, the value of neuron $c \in \{1, 2, 3, 4, 5\}$, can be seen as the probability that the rating is $r_{ki} = c$. The motivation behind this approach is to force the model to learn a better user/movie representation by treating the rating of the user k for the movie i as a specific category. Further in this model we minimize the cross entropy loss over the actual and the predicted ratings.

We predict the desired rating ($\widehat{r}_{ki}$) on the validation set as a combination of all the ratings weighted by the probability of the rating as estimated by the network:

$$\widehat{r}_{ki} = \sum_{c=1}^{5} n_c \cdot c$$

III. Results

A. Data Preparation

A very crucial aspect to take care of in recommender systems is that the validation set is "fair", i.e. it is representative of all the users in our training set. Thus relying on a random split for validation is very disadvantageous as we could end up excluding users with fewer ratings. Hence we generate each validation split to include 10% of the observed interactions for each user. Also we make sure that our pre-trained embedding models have never seen the validation set and the pre-training is solely based upon the train set.

B. Baselines v/s Embedding-Prediction Models

For comparison, different baselines were implemented. First in Table I we summarize cross-validation scores obtained using some algorithms implemented in the Surprise package [17] in Python, as well as a self-implemented SVD baseline.

We then compare our three embedding models in combination with our three different prediction models in table II. Each pair of embedding and prediction model is evaluated with 10-fold cross validation. For each epoch we first average the root mean squared error over all 10 folds. Then we report the error at epoch producing lowest error. We only report the best results obtained (using hyperparameters tuned specifically for each model).

Baseline Name	**10-fold CV score**
BaselineOnly-Surprise	0.998862
SlopeOne-Surprise	0.999499
SVD movie mean-imputed (self-implemented)	1.007455
KNNBaseline-Surpirse	1.008220
CoClustering-Surprise	1.009961
SVDpp-Surprise	1.013799
KNNWithMeans-Surprise	1.021425
KNNWithZScore-Surprise	1.023665
SVD-Surprise	1.023758
KNNBasic-Surprise	1.024921
NMF-Surprise	1.047238
NormalPredictor-Surprise	1.486223

Table I
PREDICTION RMSE SCORES FOR DIFFERENT BASELINE MODELS

Embedding Model	**Prediction Model**	**10-fold CV score**
simple	inner product	0.9923^*
simple	MLP1	$0.9957^\dagger$
simple	MLP2	$\mathbf{0.9885}^*$
autoencoder	inner product	$0.9885^\dagger$
autoencoder	MLP1	$\mathbf{0.9830}^\dagger$
autoencoder	MLP2	$0.9861^\dagger$
attentive	inner product	0.9761^*
attentive	MLP1	0.9761^*
attentive	MLP2	$\mathbf{0.9740}^*$

Table II
PERFORMANCE OF OUR MODELS.
$*$ 64 embedding size , † 128 embedding size

The *simple* embedding model corresponds to a traditional lookup of user and movie in two randomly initialised matrices. We can see that MLP2 is the best model here.

For the *autoencoder* embedding we initialise those two matrices with the user and movie representations obtained from the latent space z of a user/movie autoencoder. These are then further fine tuned. We can see from table II that all models benefited from this smart initialization, with the MLPs again outperforming the baseline *inner product* model.

Finally, we try the *attention* embedding, which is the novel embedding architecture detailed in section II-A1. All prediction models have the best performance in combination with this embedding method.

Thus we have successfully demonstrated the validity of our modular approach. Specifically, each prediction model benefits from replacing the embedding with a smarter one. Further we illustrated that our novel attention embedding helps all prediction models to achieve best performance.

The final submitted model computes the mean over our best submissions on Kaggle. This can be seen as a variant of the Bagging [18] approach.

IV. Discussion

We have presented a framework that combines optimal embeddings with high performing deep learning architectures. The advantage of such a procedure is firstly that the

network does not need to learn a good representation of the data from scratch. Secondly new users can be incorporated into the models (attention and autoencoder) without any additional training. The attention model provides all these benefits without loss of interpretability and with fewer parameters. Also we would like to point out that our prediction models are quite flexible and can in principle make use of any superior pre-trained embeddings.

Of course, the results are still far from perfect with an RMSE around 0.97, considering that the error is only 1.15 when overfitting on the mean completely, i.e. outputting the same number for all data. The methods developed in recent years have improved the score in very small steps, and we do not claim to a more radical achievement. However, our contribution is not the single best performing model, but the framework of embedding-prediction models, where results of later research can efficiently be incorporated. On the other hand we believe that the attention-embedding is a promising direction for further work in the field, which should be explored more in detail and tested on larger datasets.

References

[1] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," in *Proceedings of KDD cup and workshop*, vol. 2007, 2007, pp. 5–8.

[2] S. Zhang, W. Wang, J. Ford, F. Makedon, and J. Pearlman, "Using singular value decomposition approximation for collaborative filtering," in *Seventh IEEE International Conference on E-Commerce Technology (CEC'05)*. IEEE, 2005, pp. 257–264.

[3] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Incremental singular value decomposition algorithms for highly scalable recommender systems," in *Fifth international conference on computer and information science*, vol. 27. Citeseer, 2002, p. 28.

[4] S. K. Lam and J. Riedl, "Shilling recommender systems for fun and profit," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 393–402.

[5] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[6] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 5:1–5:38, 2019. [Online]. Available: https://doi.org/10.1145/3285029

[7] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, A. Gangemi, S. Leonardi, and A. Panconesi, Eds. ACM, 2015, pp. 111–112. [Online]. Available: https://doi.org/10.1145/2740908.2742726

[8] A. Makhzani and B. J. Frey, "k-sparse autoencoders," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.5663

[9] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, P. Champin, F. L. Gandon, M. Lalmas, and P. G. Ipeirotis, Eds. ACM, 2018, pp. 689–698. [Online]. Available: https://doi.org/10.1145/3178876.3186150

[10] A. Zeyer, K. Irie, R. Schlüter, and H. Ney, "Improved training of end-to-end attention models for speech recognition," *arXiv preprint arXiv:1805.03294*, 2018.

[11] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[12] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T.-S. Chua, "Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '17. New York, NY, USA: ACM, 2017, pp. 335–344. [Online]. Available: http://doi.acm.org/10.1145/3077136.3080797

[13] H. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, C. Sierra, Ed. ijcai.org, 2017, pp. 3203–3209. [Online]. Available: https://doi.org/10.24963/ijcai.2017/447

[14] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 173–182.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[16] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[17] N. Hug, "Surprise, a Python library for recommender systems," http://surpriselib.com, 2017.

[18] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

Appendix

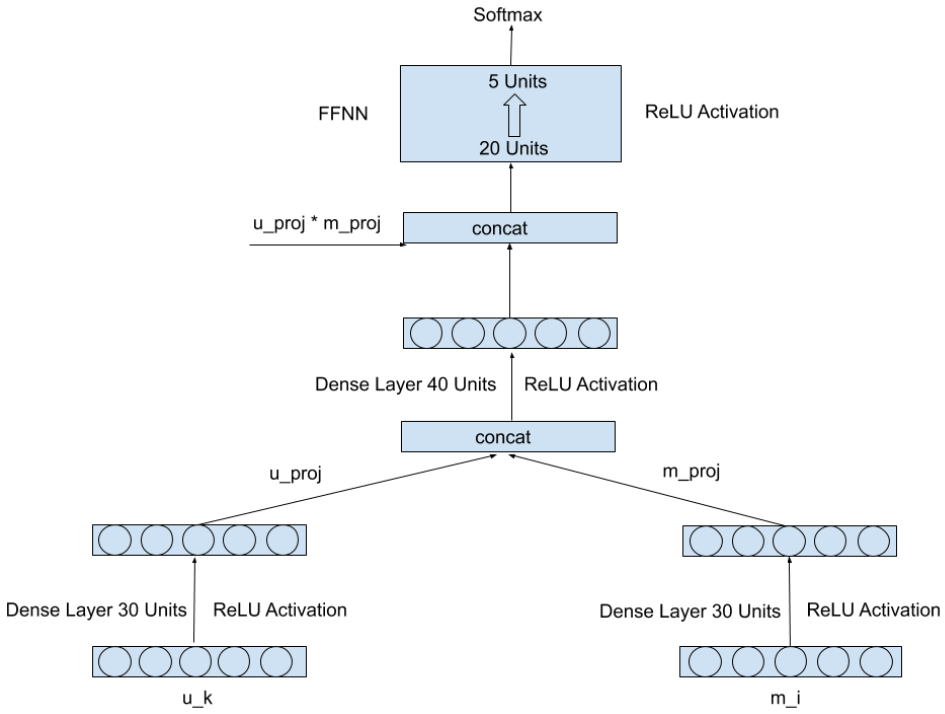

Figure 3. A detailed graph of MLP2 .