A Deep Learning Approach to Personalized Exercise Recommendations in Stimuler

Shobhit Shukla
Github LinkedIn Mail

Abstract

This report presents the design and implementation of a personalized recommendation system for language learning, aimed at providing tailored practice exercises based on individual learner profiles and past interactions. The proposed system leverages a two-stage approach involving candidate generation and ranking using deep learning models. A recurrent neural network (RNN) is employed in the candidate generation phase to model the learner's interaction history and proficiency. For ranking, a deep Multi-Layer Perceptron (MLP) assigns relevancy scores to the exercises based on user-specific features, enabling a highly personalized learning experience. This approach enables dynamic adaptation to users' evolving language skills and preferences, resulting in an effective and engaging language learning tool.

Introduction

The objective of this report is to design a robust exercise recommendation system that adapts to each learner's unique profile and historical interactions. The system should dynamically suggest practice exercises that address users' frequent mistakes while considering their demographic and contextual information. It should not only cater to the specific learning needs of each user but also align with their preferences and interests to maintain engagement.

We have the following principal question that we need to solve, as derived from the given problem statement:

"For any learner, how can we **recommend** the most **relevant** practice exercise that adapts to their **past mistakes** and **current interests**, while dynamically **adjusting over time** to improve their language skills across grammar, vocab, pronunciation and fluency/content"

Data Format Assumptions

I have assumed the following data format which includes 1.Time-series data of user interactions, including utterances, errors, and feedback 2.User profile data with contextual information. 3.Exercise database with metadata (type, difficulty, topic, etc.)

1. User Profile

This section stores static information about each user:

```
"user_profile": {
  "user_id": "string",
  "age": "integer",
  "location": "string",
  "native_language": "string",
  "proficiency_level": "string",
  "interests": ["string"],
  "preferred_learning_style": "string"
}
```

- **user_id**: A unique identifier for each user.
- **age**: The user's age, useful for tailoring content appropriately.
- **location**: The user's geographical location, useful for cultural context.
- **native_language**: The user's first language, helpful for understanding potential areas of difficulty.
- **proficiency_level**: The user's current English proficiency (e.g., beginner, intermediate, advanced).
- **interests**: An array of topics the user is interested in for personalizing content.
- **preferred_learning_style**: The user's preferred learning method (e.g., visual, auditory, kinesthetic).

2. User Utterances

This section captures the data from each interaction in form of utterances a user has with the system:

```
"user_utterances": {
  "interaction_id": "string",
  "user_id": "string",
  "timestamp": "datetime",
  "utterance": "string",
  "feedback": {
    "grammar": {
      "score": "float",
      "errors": [
        {
          "error_type": "string",
          "description": "string",
          "suggestion": "string"
        }
      ]
    },
    "vocabulary": {
      "score": "float",
      "errors": [
        {
          "error_type": "string",
          "word": "string",
          "suggestion": "string"
        }
      ]
    },
    "pronunciation": {
      "score": "float",
      "errors": [
        {
          "phoneme": "string",
          "correct_pronunciation": "string",
          "user_pronunciation": "string"
        }
      ]
    },
    "fluency_content": {
      "score": "float",
      "feedback": "string"
    }
  }
}
```

- **interaction_id**: A unique identifier for each interaction.
- **user_id**: Links the interaction to a specific user.
- **timestamp**: The time when the interaction occurred.
- **utterance**: The actual text or speech input from the user.
- **feedback**: Detailed analysis of the user's performance, broken down into four categories:
  - **Grammar**: Overall score and specific errors with descriptions and suggestions.
  - **Vocabulary**: Score and errors related to word usage.
  - **Pronunciation**: Score and specific phoneme errors.
  - **Fluency/Content**: Overall score and feedback on fluency and content.

3. Exercise

This section defines the structure of exercises in the exercises repository:

```
"exercise": {
  "exercise_id": "string",
  "type": "string",
  "category": "string",
  "difficulty": "float",
  "content": "string",
  "metadata": {
    "topic": "string",
    "cultural_context": "string",
    "target_error_types": ["string"]
  }
}
```

- **exercise_id**: Unique identifier for each exercise.
- **type**: The type of exercise (e.g., multiple choice, fill-in-the-blank, speaking).
- **category**: The language skill category (e.g., grammar, vocabulary, pronunciation).
- **difficulty**: A numerical representation of the exercise's difficulty.
- **content**: The actual content of the exercise.
- **metadata**: Additional information about the exercise:
  - **topic**: The subject matter of the exercise.
  - **cultural_context**: Any cultural references or context in the exercise.
  - **target_error_types**: Specific types of errors the exercise addresses.

4. User Exercise Interaction

This section tracks how users interacted with past exercises:

```
"user_exercise_interaction": {
  "interaction_id": "string",
  "user_id": "string",
  "exercise_id": "string",
  "timestamp": "datetime",
  "performance": "float",
  "time_spent": "float",
  "user_response": "string",
  "user_feedback": "string"
}
```

- **interaction_id**: Unique identifier for each exercise interaction.
- **user_id** and **exercise_id**: Links the interaction to a specific user and exercise.
- **timestamp**: The time when the interaction occurred.
- **performance**: A score representing how well the user performed.
- **time_spent**: The time spent on the exercise.
- **user_response**: The user's response to the exercise.

Model Conceptualization

The overall structure of Stimuler's recommendation system is illustrated in Figure 1. The system comprises two neural networks: one for candidate generation and one for ranking. The candidate generation network takes sequential user interaction data (User past utterances, past user exercise interactions) and user contextual information and retrieves a subset of practice exercises from a large corpus of exercises. These candidates are relevant to the user's competency level and exercise complexity along with aligning well with user's interests.

The ranking network refines these suggestions, distinguishing the relative importance of each candidate with high recall. It assigns a relevancy score to each exercise based on features describing the user's current proficiency, user's preferences (e.g., topic interests) along with features describing the exercises. This score measures the probability of success on the candidate exercises. This two-stage approach allows us to retrieve relevant exercises from a vast collection while ensuring that the exercises delivered to the user are personalized and engaging.

During development, we rely heavily on offline metrics (precision, recall, ranking loss, etc.) to improve the system iteratively. However, the final evaluation of the effectiveness of any algorithm or model is determined through A/B testing in live experiments. These live experiments help measure subtle shifts in user engagement metrics such as practice completion rates, improvement in language proficiency, and session length, which may not always align with offline experiments.

The detail implementation of the model is described below.

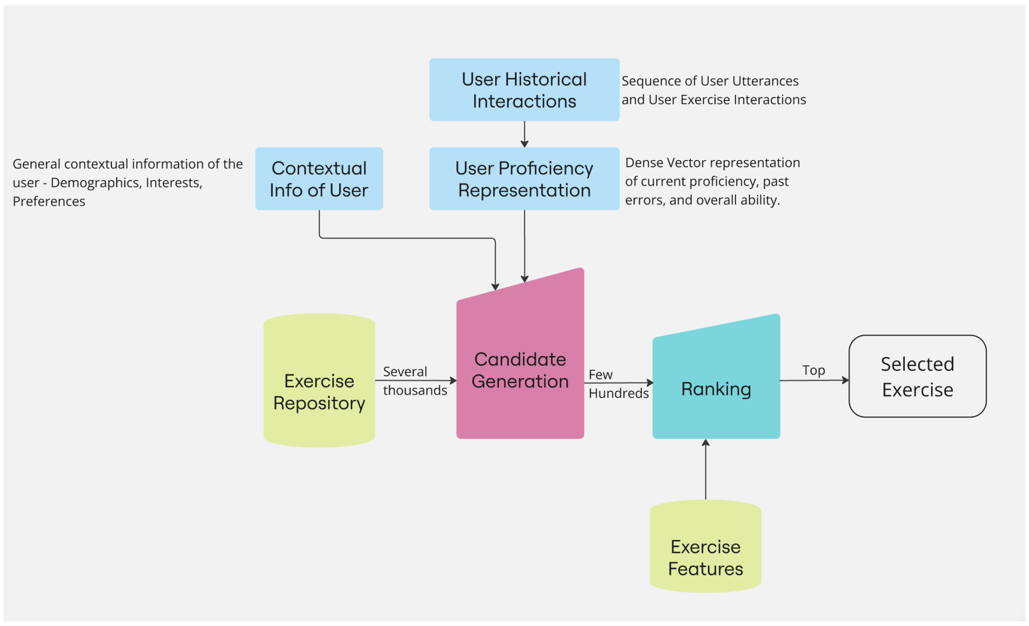
User Historical Interactions
Sequence of User Utterances and User Exercise Interactions
General contextual information of the user - Demographics, Interests, Preferences
Contextual Info of User
User Proficiency Representation
Dense Vector representation of current proficiency, past errors, and overall ability.
Exercise Repository
Several thousands
Candidate Generation
Few Hundreds
Ranking
Top
Selected Exercise
Exercise Features

Figure 1: Proposed overall structure of Stimuler's recommendation system

Candidate Generation

During candidate generation stage, the vast collection of language exercises is narrowed down to a few hundred that may be relevant to the user in context of user's proficiency level and their interests. In particular, proposed approach is backed by a recurrent neural-network model (specifically, a long short-term memory, or LSTM, model), which learns to compress a learner's history of interactions with the system in form of utterances or previously attempted exercises along with their interests into a D dimensional vector. Every time a learner completes another exercise, we will update this vector based on its prior state, the exercise that the learner has completed, and whether they got it right. It is this vector, rather than a single value, that represents a learner's ability along with their interests.

This model can analyze and track individual learning trajectories facilitating highly personalized lesson planning tailored to the unique progress and challenges of each user.

Recommendation as Classification

We frame recommendation as an extreme multiclass classification problem, where the goal is to accurately classify an exercise e_t for a user U at time t among thousands of possible exercises i from a corpus E, given the user and context C,

$$P(e_t = i|U, C) = \frac{e^{v_i u}}{\sum_{j \in E} e^{v_j u}}$$

where $u \in R^N$ represents a high-dimensional representation of the user's proficiency level and context, and $v_j \in R^N$ represents embeddings of each candidate exercise. The task of the neural network is to learn these user representation as a function of the user's conversation history and context, and to use them for classification via a softmax classifier.

Although explicit feedback mechanisms could be employed in Stimuler (e.g., user rating of exercises), we primarily use implicit feedback, such as exercise completion and performance, category feedbacks etc. to train the model. This approach is preferred due to the larger volume of implicit feedback available, enabling us to recommend more targeted exercises in cases where explicit feedback is sparse.

Challenges with Large Exercise Corpora

When the number of exercises E is very large, computing the softmax over all exercises can be computationally expensive. Calculating the denominator of the softmax function involves summing over all exercises, making it infeasible for real-time systems. Therefore, optimizations like negative sampling is used.

Efficient Extreme Multiclass

To efficiently train the model with thousands of classes (i.e. exercises), we utilize negative sampling techniques, where negative examples (exercises which the user has not interacted with yet) are sampled, and importance weighting is applied to correct for this sampling. For each example, we minimize the cross-entropy loss for both the true label (positive sample) and the sampled negative exercises. In practice, we sample several hundreds negatives per training example, which results in a more than 10-fold speedup compared to traditional softmax considering we are having a corpus of several thousands of exercises. At serving time, we need to compute the most likely N exercises to rank in the next stage. Scoring thousands of exercises under strict latency constraints requires an approximate scoring scheme. We can use a nearest-neighbor search in the dot product space to rank exercises, rather than relying on likelihoods from softmax outputs.

Model Architecture

We propose a two-tower Deep Neural Network architecture for training the candidate generation model. In a two-tower DNN:

- **User Tower:** It maps user features and sequential history of user interactions to a *User Ability Vector.* Here's a breakdown of the whole architecture - The interactions include sequence of user utterances and interaction with previous exercises. These sequences are embedded firstly and then passed through an LSTM network. The last hidden state of the LSTM that encodes the sequence of user utterances is concatenated with the last hidden state of a second LSTM that encodes the sequence of previous exercise interactions the user encountered. Along with this, relevant contextual information such as demographics, interests and preferences which is represented by user feature vector are also embedded and are concatenated. This final user-specific vector, referred to as the *User Ability Vector*, encodes the user's current proficiency across various learning dimensions as well as user's contextual information.
- **Exercise Tower:** It maps exercise features to an exercise embedding which is further encoded by a MLP to form the final *Exercise Representation Vector* having same dimensions as that of *User Ability Vector.*
- **Scoring:** The similarity between *User Ability Vector* and *Exercise Representation Vector*(e.g., dot product or cosine similarity) is used to predict relevance.

A significant advantage of using deep neural networks is the ability to incorporate arbitrary continuous and categorical features. In Stimuler, the user's demographic data (e.g., location, age, language proficiency) is embedded similarly to their conversation history. Other continuous features, such as the user's age and logged-in status, are input directly into the network as normalized real values between [0, 1].

Training Procedure

1. Input Preparation

- For each training instance, extract sequential features of user's past L interactions and user's contextual information features
- Prepare the **positive label** as the target item — this is an exercise that the user interacted with at timestep t i.e. the next timestep
- Sample a few **negative items** that the user did not interact with and/or performed poorly on, using negative sampling.

2. Forward Pass

- The User Tower processes user features to generate a user ability vector, $\mathbf{u}$.
- The Exercise Tower processes exercise features to generate an exercise representation, $\mathbf{e}_j$ for each exercise e.
- Compute the similarity score between the user representation and exercise representation using a dot product:

$$\text{score} = \mathbf{u} \cdot \mathbf{e}_j$$

3. Softmax Cross-Entropy Loss

- For each training instance, compute the softmax score over the target (positive exercise) and sampled negative Exercises:

$$P(j|x) = \frac{e^{\mathbf{u}\cdot\mathbf{e}_j}}{\sum_{k=1}^{N} e^{\mathbf{u}\cdot\mathbf{e}_k}}$$

  where N is the number of positive and negative Exercises in the instance.

4. Loss Function

- Use a cross-entropy loss between the true label (target exercise) and the predicted probabilities over the positive and negative exercises:

$$\mathcal{L} = -\sum_j y_j \log(P(j|x))$$

  where y_j is the ground truth label, which is 1 for positive exercises and 0 for negative exercises.

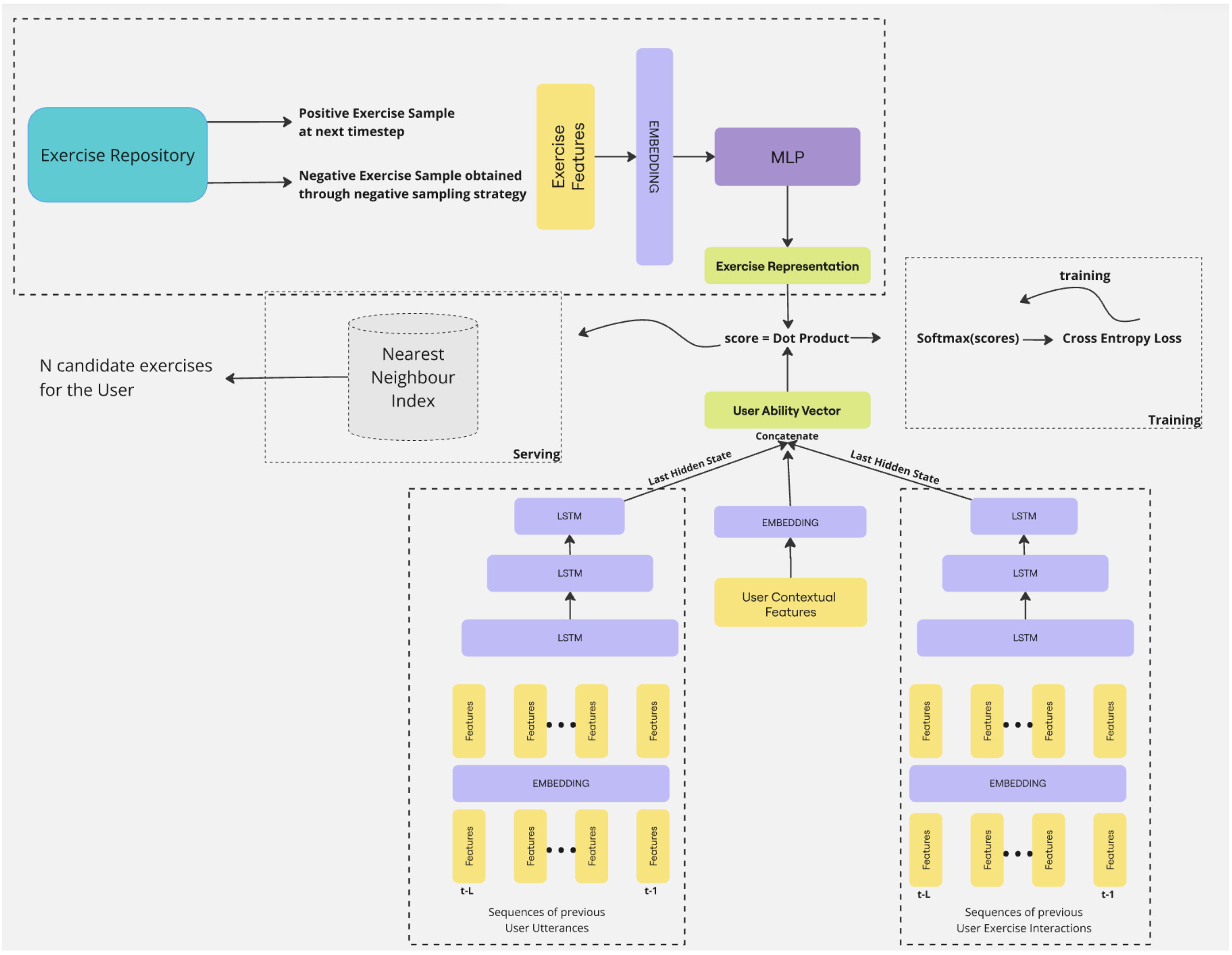

Figure 2: Overview of the Candidate Generation Model. The model utilizes user utterance sequences and exercise interaction sequences as input, processes them through separate LSTM layers, and concatenates the last hidden states to form a user ability vector. The ability vector is used to score and rank candidate exercises based on their representations.

5. Backpropagation and Parameter Update

- Perform backpropagation and update the weights of both towers using stochastic gradient descent (SGD) or a variant such as Adam optimizer.

By structuring each training instance with these features and labels, the two-tower model can be effectively train for sequential recommendations, capturing both the user's learning trajectory and contextual information for generating personalized exercises.

Candidate Ranking

Candidate Ranking model ranks the candidate exercises based on Relevancy score of a candidate exercise $e_{candidate}$ for a user u. The proposed Candidate ranking model is trained using a Bayesian Personalised Ranking (BPR) loss. This model returns a relevancy score for a specific user and a specific exercise. This relevancy score is more representative of the relevancy to the user than the simple similarity score as used by the candidate generation model because the model is trained on a deep MLP which learns complex interactions between the user and exercise features instead of a simple dot product.

BPR Loss is a widely used pairwise loss function designed for optimizing recommendation systems. It works by optimizing the relative ranking of items for each user. When using BPR loss, we focus on learning a model that ranks the relevant (positive) exercises higher than the irrelevant (negative) exercises. It's particularly suitable for implicit feedback scenarios, where we only have information about what users engaged with, but not what they explicitly disliked.

Dataset Structure

The dataset is structured in the form of triplets (u, i, j), where:

- u represents the user.
- i is a positively engaged exercise (e.g., an exercise that the user performed well on).
- j is a negatively engaged exercise (e.g., an exercise that the user did not attempt or performed poorly on).

Each row in the dataset contains the following components:

- **User Features:** User ability vector at the timestep when the user interacted with positive exercise i.
- **Exercise Features:** Exercise representation vector as generated previously for both execise i and execise j.
- **Explicit Features:** Other Explicit feedback of user. For example: Number of exercises successfully completed etc.

Model Architecture

The candidate ranking model is based on a deep MLP. The User, exercise and explicit features are concatenated for the positive and negative exercise and is passed through a MLP to finally generate a relevancy score. The model optimizes for ranking pairs of exercises such that, given a user u, the model assigns a higher relevancy score to exercises that the user has interacted with or engaged in, compared to those they have not.

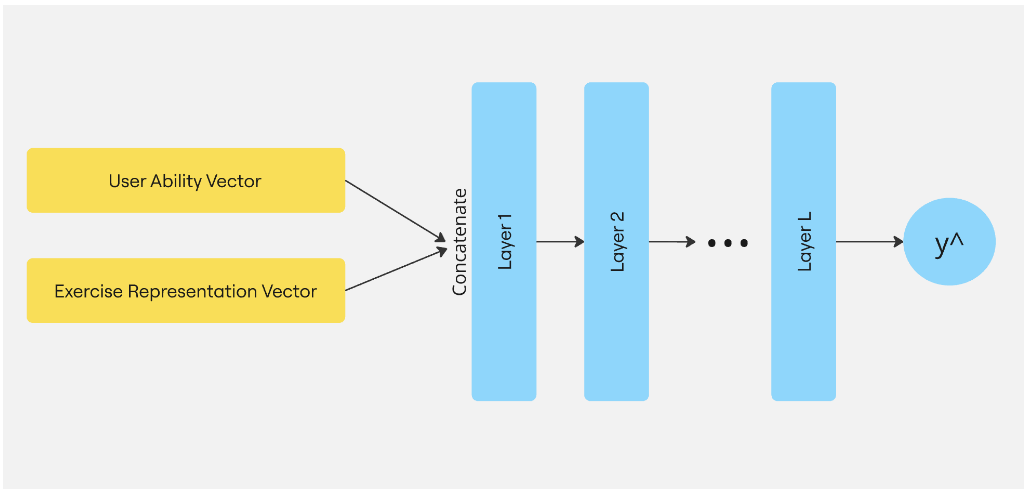

Figure 3: Overview of the Candidate Ranking Model

Training Procedure

The model is trained using the BPR loss function, defined as:

$$\mathcal{L}_{BPR} = -\sum_{(u,i,j)\in D} \ln(\sigma(\hat{y}_{uij})) + \lambda\|\Theta\|^2 \tag{1}$$

where:

- $\hat{y}_{uij} = \hat{y}_{ui} - \hat{y}_{uj}$ is the difference in predicted scores for positive exercise i and negative exercise j.
- $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.
- Θ represents the model parameters, and λ is the regularization term.

The training procedure involves the following steps:

1. **Input Preparation:** Generate triplets (u, i, j) for training, where i is a positive exercise and j is a negative exercise for user u.
2. **Forward Pass:** Compute the scores $\hat{y}_{ui}$ and $\hat{y}_{uj}$ for the positive and negative exercises, respectively.
3. **Compute BPR Loss:** Calculate the BPR loss using the difference in scores between i and j.
4. **Backpropagation:** Update model parameters using backpropagation and an optimizer like Adam.
5. **Regularization:** Add a regularization term to prevent overfitting.

Evaluation Metrics

The performance of the ranking model can be evaluated using the following metrics:

- **Area Under the ROC Curve (AUC):** Measures the model's ability to rank positive exercises higher than negative ones.
- **Precision at K:** Computes the proportion of relevant exercises in the top K recommendations.
- **Normalized Discounted Cumulative Gain (NDCG):** Evaluates the ranking quality by considering the position of relevant exercises in the ranked list.

Addressing the Cold Start Problem

When a new user joins the system, we lack the historical interaction data necessary to generate personalized recommendations effectively. This section proposes an extension to our existing two-tower Deep Neural Network architecture to address this issue.

Adaptive User Tower with Auxiliary Network

We propose an enhancement to the User Tower in our candidate generation model by introducing an auxiliary network specifically designed for new users. This approach allows us to generate a meaningful initial User Ability Vector based on limited available information, enabling relevant recommendations from the outset.

Auxiliary Input Features

The auxiliary network utilizes the following input features available for new users:

- User profile data: age, location, native language, and stated proficiency level
- Initial assessment results: scores from a quick placement test administered during onboarding
- Stated interests: encoded as a multi-hot vector representing topics of interest

Auxiliary Network Architecture

The auxiliary network is designed as follows:

- A separate neural network integrated within the User Tower
- Implemented as a Multi-Layer Perceptron (MLP), simpler than the main LSTM-based User Tower
- Processes the initial user features to generate an approximate User Ability Vector

Integration with Main User Tower

The auxiliary network is integrated with the main User Tower as follows:

- For new users, the output of the auxiliary network serves as the initial User Ability Vector
- As the user interacts with the system, we gradually blend the auxiliary output with the LSTM-based User Ability Vector
- The blending is weighted based on the number of user interactions, transitioning smoothly from the initial approximation to a more personalized representation

Training Procedure

The training of the auxiliary network involves the following steps:

1. Select a subset of users who have both initial profile data and a substantial history of interactions
2. For each selected user, use their final User Ability Vector (after several interactions) as the target for the auxiliary network's output
3. Train the auxiliary network to predict this target vector based only on the initial user data
4. This approach teaches the auxiliary network to estimate a user's eventual ability vector using only the information available at sign-up

Deployment Strategy

The deployment of this enhanced User Tower follows these steps:

1. For a new user, initially use only the auxiliary network's output as the User Ability Vector
2. As the user completes exercises, begin incorporating the LSTM-based vector
3. Weight the contribution of each vector based on the number of user interactions
4. Gradually phase out the auxiliary network's contribution as more user-specific data becomes available

By implementing this enhanced User Tower, we can significantly mitigate the cold start problem in Stimuler, providing new users with relevant and engaging content from their very first interaction while maintaining the sophisticated personalization capabilities of our existing recommendation system.

Final Workflow of the Stimuler Recommendation System

This section presents the comprehensive workflow of the Stimuler recommendation system, incorporating all components including the candidate generation model, ranking model, and the solution for cold start scenarios.

Workflow Description

The Stimuler recommendation system operates through the following steps:

1. **User Input**: The process begins with user input, which can be in the form of utterances, exercise interactions, or initial profile information for new users.
2. **User Profile**: The system maintains and updates a user profile based on ongoing interactions and explicitly provided information.
3. **Cold Start Solution**: For new users, the cold start solution is activated:
   - The auxiliary network processes initial user data to generate an approximate User Ability Vector.
   - This vector is used in place of the history-based vector for new users.
4. **Candidate Generation**: The two-tower deep neural network generates candidate exercises:
   - The User Tower processes user data (including cold start data for new users) to create a User Ability Vector.
   - The Exercise Tower processes exercise features from the exercise corpus.
   - Candidate exercises are selected based on the similarity between user and exercise vectors.
5. **Candidate Ranking**: The selected candidate exercises undergo a ranking process:
   - A deep MLP model computes relevancy scores for each candidate exercise.
   - The model considers user features, exercise features, and explicit feedback.
6. **Final Recommendations**: The system produces a final list of recommended exercises, ordered by their relevancy scores.
7. **Feedback Loop**: User interactions with the recommended exercises feed back into the system, updating the user profile and improving future recommendations.

References

[1] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay, *Deep Learning based Recommender System: A Survey and New Perspectives*, ACM Computing Surveys, vol. 52, no. 1, pp. 1–38, 2019.

[2] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha, *Sequential Recommendation with User Memory Networks*, In Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM), pp. 108–116, 2018.

[3] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang, *BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer*, In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM), pp. 1441–1450, 2019.

[4] Paul Covington, Jay Adams, and Emre Sargin, *Deep Neural Networks for YouTube Recommendations*, In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys), pp. 191–198, 2016.

[5] Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk, *Session-based Recommendations with Recurrent Neural Networks*, In Proceedings of the International Conference on Learning Representations (ICLR), 2016.

[6] IEEE Spectrum, *How Duolingo's AI Learns What You Need to Learn*, Available at: https://spectrum.ieee.org/duolingo, 2021.

[7] Duolingo Blog, *Learning How to Help You Learn: Introducing Birdbrain*, Available at: https://blog.duolingo.com/learning-how-to-help-you-learn-introducing-birdbrain, 2021.

[8] Duolingo Blog, *How We Learn How You Learn*, Available at: https://blog.duolingo.com/how-we-learn-how-you-learn, 2021.

[9] Google Developers, *Recommendation Systems*, Available at: https://developers.google.com/machine-learning/recommendation, 2021.

[10] Jiaxi Tang and Ke Wang, *Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding*, In Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM), pp. 565–573, 2018.

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua, *Neural Collaborative Filtering*, In Proceedings of the 26th International Conference on World Wide Web (WWW), pp. 173–182, 2017.