Relation Aware Self-attention Knowledge Tracing

Description of how correlation between problems embeddings and performance is defined

Exercises are represented using a function f: M -> R^d.
Each word of the exercise is first translated into unique embedding, and then exercize is represented as weighted combination of embeddings of all the words present using Smooth Inverse Frequence:

"a" is a trainable parameter

$$\mathbf{E}_i = \frac{1}{|s_i|} \sum_{w \in s_i} \frac{a}{a + p(w)} f(w),$$

After being described as weighted combination of 50 ("d" in general) most important words (embedded), we need to define correlation between problems.
This correlation is defined as SUM of two components: a collaborative filter, calculating correlation between performance of users for same two problems, and a content filter, calcualting the similarity between two problems with cosine similarity of the embedded words.

		exercise i incorrect	exercise i correct	total
exercise j	incorrect	n_{00}	n_{01}	n_{0*}
	correct	n_{10}	n_{11}	n_{1*}
	total	n_{*0}	n_{*1}	n

$$\phi_{i,j} = \frac{n_{11}n_{00} - n_{01}n_{10}}{\sqrt{n_{1*}n_{0*}n_{*1}n_{*0}}}.$$

$$\mathrm{sim}_{i,j} = \frac{\mathbf{E}_i \mathbf{E}_j}{||\mathbf{E}_i||_2 ||\mathbf{E}_j||_2}$$

$$\mathbf{A}_{i,j} = \begin{cases} \phi_{i,j} + \mathrm{sim}_{i,j}, & \text{if } \mathrm{sim}_{i,j} + \phi_{i,j} > \theta \\ 0, & \text{otherwise.} \end{cases}$$

IDEA: we can try to change this suj with a WEIGHTED SUM or mantain the two matrices separated.
Or we could try using a more complex model for this system, which is similar to recommender systems..

$$\hat{\mathbf{x}}_j = [\mathbf{E}_{e_j} \oplus \mathbf{r}_j] + \mathbf{P}_j$$

Inputs are represented as using triples of (embedding, correctnes value, time). We need to include a positional encoding too to allow discrimination and this positional encoding is done using P, a positional encoding matrix of dimension l x 2d, where l is maximum allowed sequence lenght and d is number of embedding words used to represent an exercise, so the lenght of Eej.

matrix of embeddings

matrix of forgetting factors

$$\mathbf{R} = \mathrm{softmax}(\mathbf{R}^E + \mathbf{R}^T),$$

We could change softmax with a sort of KNN

Description of how forgetting factor is modeled

If we consider the current exercise E, for any previous exercise Ej at time Tj, we define deltaJ = Ti -Tj and we modeled forgetting factor as:

$$\mathbf{R}^T = [\exp(-\Delta_1/S_u), \exp(-\Delta_2/S_u), \ldots, \exp(-\Delta_{n-1}/S_u)],$$

where Su is the forgetting factor which can be learnt for every user and represent the tendence to forget topics.

Description of how attention weights are managed

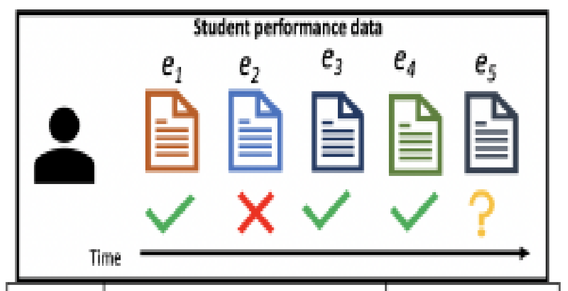
Attention mechanism is the core and main difference introduced by RKT.
Coeficients learnt from attentio mechanism are equal to Alpha:

$$\alpha_j = \frac{\exp(e_j)}{\sum_{k=1}^{n-1} \exp(e_k)}, e_j = \frac{\mathbf{E}_{e_n} \mathbf{W}^Q (\hat{\mathbf{x}}_j \mathbf{W}^K)^T}{\sqrt{d}},$$

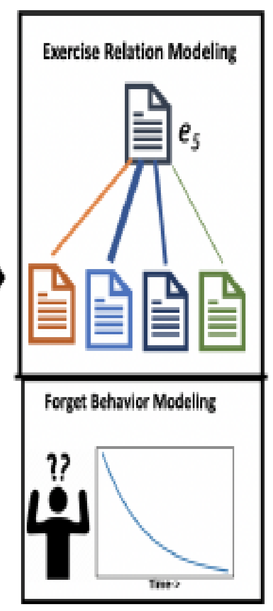
where WQ and WK are respectively query and key vectors used to represent learn attentions on exercises.

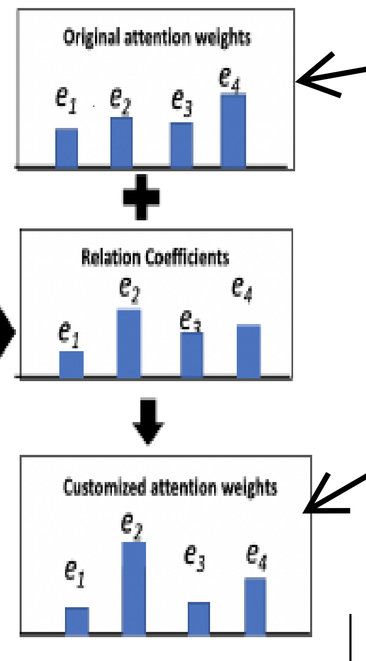
Finally we sum weights from correlation and forgetting factor with wwights from attention mechanism with a WEIGHTED sum:

$$\beta_j = \lambda \alpha_j + (1 - \lambda)\mathbf{R}_j,$$



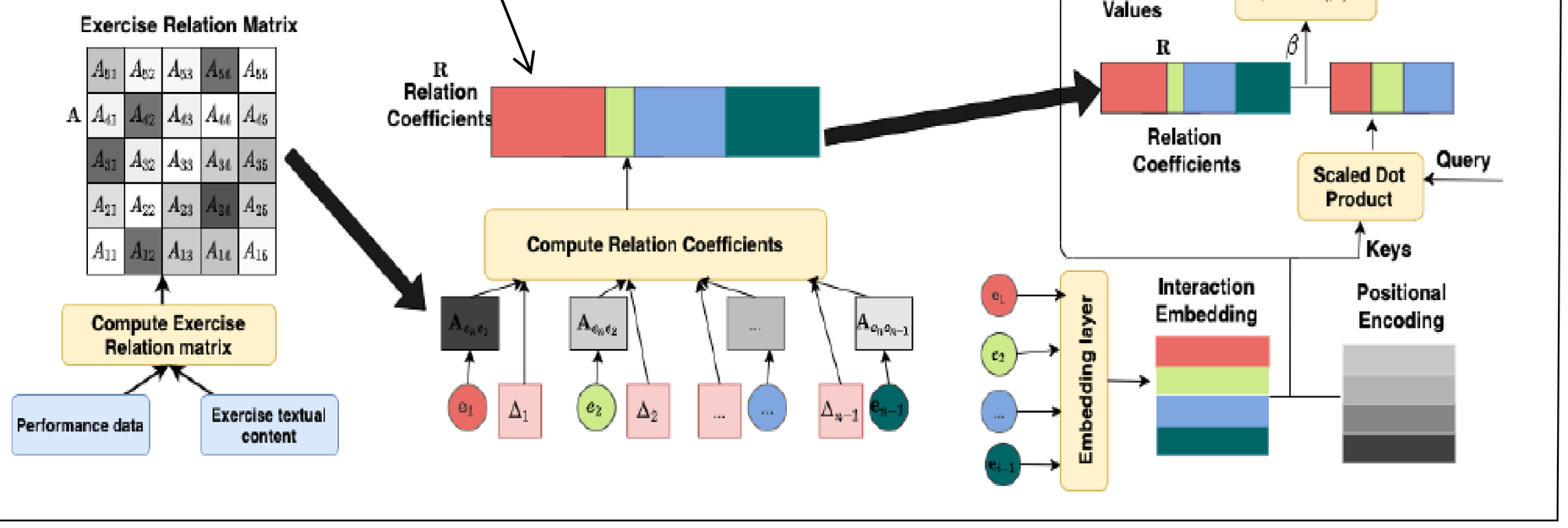
Exercise	Exercise Text	KC
e_1	Solve x: $2\|x+6\|+7 = 4x+6+4$	Linear Equations
e_2	Solve for x , $x^2 + 3x - 28 = 0$	Quadratic Equations
e_3	Evaluate $\sqrt{28}$	Square Root
e_4	What are factors of 5?	Division
e_5	Where will $f(x)$ intersect the x axis? $f(x) = (x+9)^2$	Quadratic Equations





More in deep

Each exercise is seen as combination of other exercise, this coeficients are leart from performance data and textual content



In the end we obtain outputs from weights as:

$$\mathbf{o} = \sum_{j=1}^{n-1} \beta_j \hat{\mathbf{x}}_j \mathbf{W}^V,$$

We add Always residual network and apply Normalization and Dropout

And this outputs are then given as input to a Feed Forward Neural Network, to insert possibility of relations between latent dimensiones, introducing non-linearities:

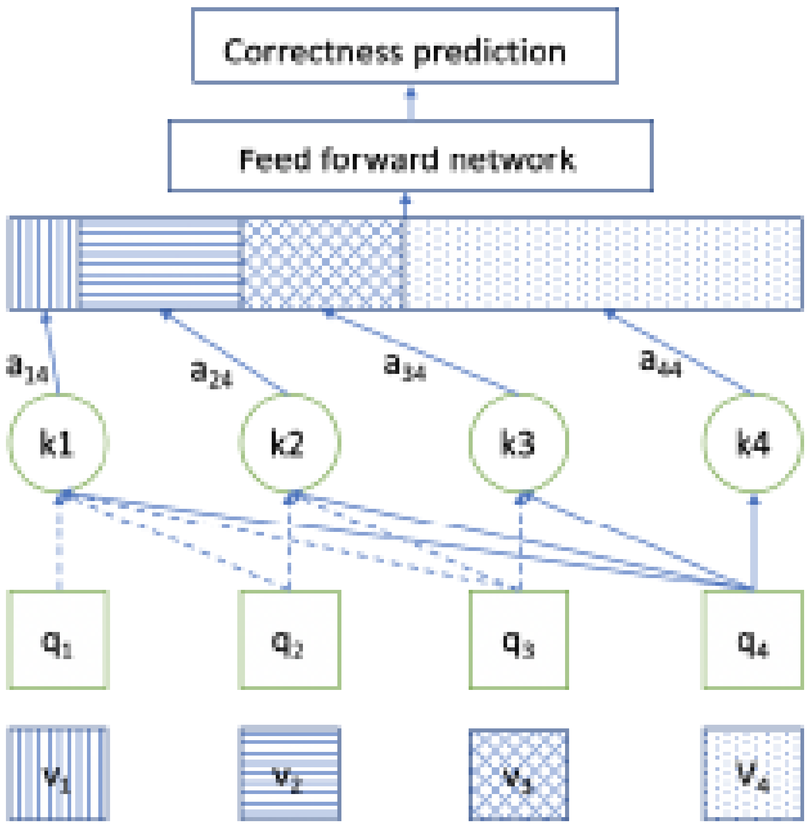
$$\mathbf{F} = \mathrm{ReLU}(\mathbf{o}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)},$$

where W are weight matrices and b1 and b2 are Bias vectors.
We add Always residual network and apply Normalization and Dropout

In the end we apply a sigmoid function to a linear layer of F matrix obtained to generate probability of answering correctly:

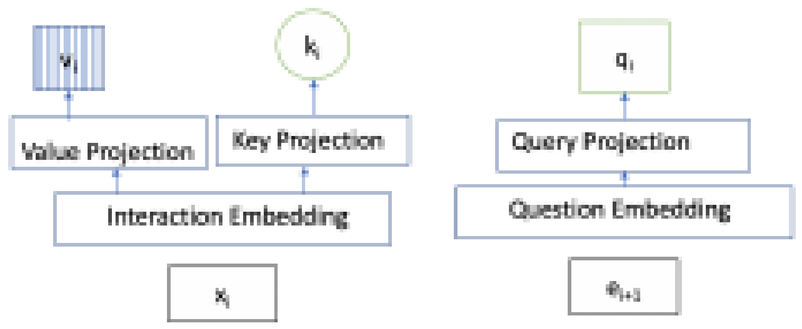
$$p = \sigma(\mathbf{F}\mathbf{W} + \mathbf{b}),$$

SAKT- Self-Attentive model for Knowledge Tracing



In the end we take outputs of Feed Forward network to calcuate prediciton as:

$$p_i = \text{Sigmoid}(\mathbf{F}_i\mathbf{w} + \mathbf{b}),$$



Input Xi is composed of a couple (exercise, result) and we have a sequence of inputs Xi which constitutes the rel input of the system and the new exercise we want to predict the result Ei+1.

The past interactions are embedded using a trained interaction embedding matrix M of dimensions 2E*d , to express each exercise as a weighted combination of some terms (embedded concepts). Exercise to predict instead will be embedded with a trained exercise matrix of dimensions E*d. This will lead to a matrix M and a vector E, we can use them as inputs to a SELF ATTENTIVE layer.

$$\mathbf{Q} = \hat{\mathbf{E}}\mathbf{W}^Q, \mathbf{K} = \hat{\mathbf{M}}\mathbf{W}^K, \mathbf{V} = \hat{\mathbf{M}}\mathbf{W}^V,$$

Where Wq, Wk and Wv are respectively attention matrices for query, key and value and are learnt during training.

In the end we use Q, K and V as input to calculate attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}.$$

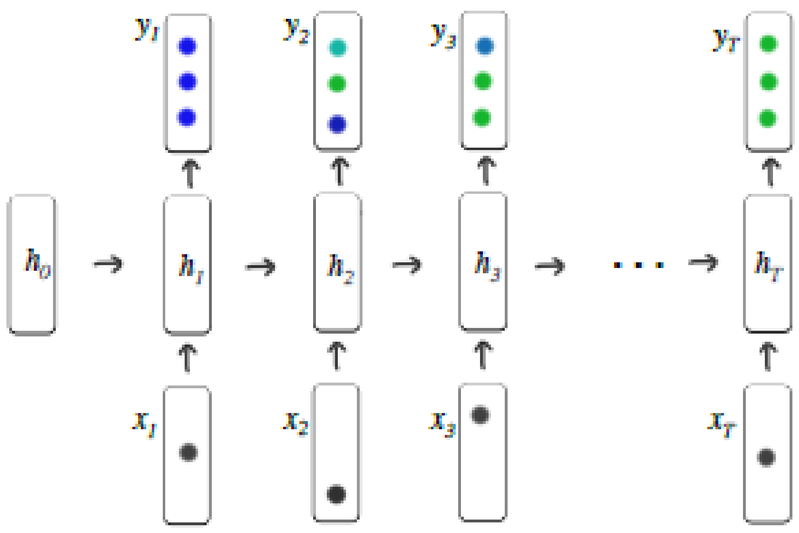
We use Multi Head Attention too, meaning that we calculate different Wq, Wk, Wv to allow detection of different templates.

In the end we take the outputs from different attention heads and use as inputs for a Feed Forward Neural Network to allow non linearities and relations between different heads.

$$\mathbf{F} = \text{FFN}(\mathbf{S}) = \text{ReLU}(\mathbf{S}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)},$$

We use Normalization and add Residual Connections too.

DKT: Deep Learning for Knowledge Tracing



Vanilla RNN model

we will apply two different types of RNNs – a vanilla RNN model with sigmoid units and a Long Short Term Memory (LSTM) model.

Inputs must be sequences of fixed lenght input vectors Xt, where we distinguish:
1. with small number M of unique exercises, Xt will be an one-hot encoding of interactions {Qt, At}
It will have dimension: 2M, where M is number of unique exercises. So we are describing each exercize as correct or not or not done.
2. with big number M of unique exercises, Xt will be impossible to encode this way, so we will represent each input tuple or with a random vector Nqa, with dimension N << M motivated by compressed sensing, which affirms that a K-sparse signal can be recovered from Klog(d) random linear projections.

Outputs instead will be a sequence of values representing probabilities of answering correctly to the corresponding problem.

Optimization objective:

The objective to minimize is the Negative Log Likelihood of observed sequence.equal to:

$$L = \sum_t \ell(\mathbf{y}^T \delta(q_{t+1}), a_{t+1})$$

It is minimized using Stohastich Gradient Descent on minibatches
Dropout is applied to avoid overfitting ONLY for output computation, not for hidden state
Exploiding gradient is avoid by truncating the lenght of gradients with norm over a threshold

We used HIDDEN DIMENSIONALITY of 200, and mini batch-size of 100.

Discovering exercise relationships:

We can use this to discover relationships between exercises (finding knowledge concepts) by assigning a infuence J to every directed pair of exercises:

$$J_{ij} = \frac{y(j|i)}{\sum_k y(j|k)},$$

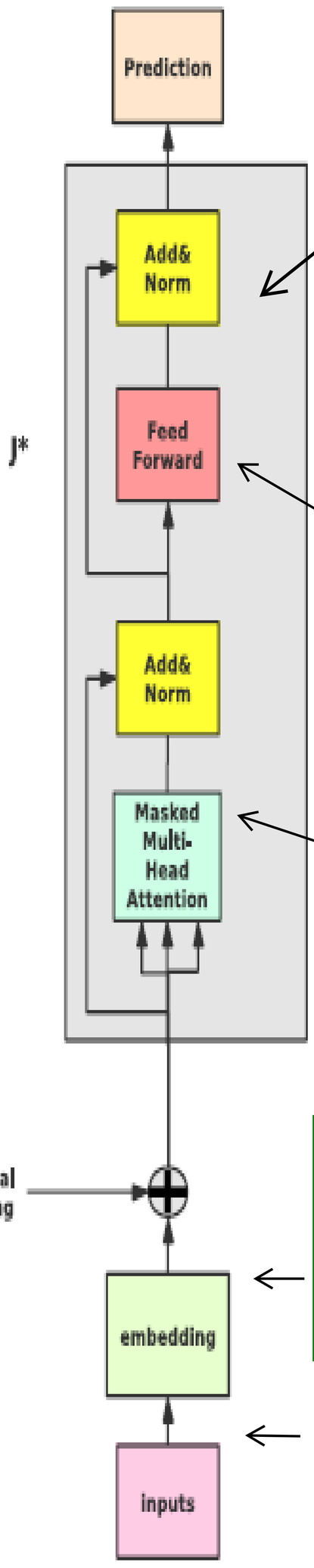
KT-XL: A Knowledge Tracing Model for Predicting Learning Performance Based on Transformer-XL

Prediction Layer

$$\mathbf{T}_i = \frac{\sum_0^{i-1} \mathbf{F}_x}{i-1}$$ ← Normalized

$$y_i = \text{Sigmoid}((\mathbf{T}_i, \mathbf{F}_i)\mathbf{W}_3 + b_3)$$ ← Function for objective result prediction (0 or 1)

$$y_i' = (\mathbf{T}_i, \mathbf{F}_i)\mathbf{W}_3 + b_3$$ ← Function for subjective result prediction



Recurrence Mechanism

Extended state — CONCATENATION

$$\tilde{\mathbf{h}}_{t+1}^{j-1} = \left[\text{SG}(\mathbf{h}_t^{j-1}) \circ \mathbf{h}_{t+1}^{j-1}\right],$$
$$\mathbf{q}'^{j}_{t+1}, \mathbf{k}'^{j}_{t+1}, \mathbf{v}'^{j}_{t+1} = \mathbf{h}_{t+1}^{j-1}\mathbf{W}_q^T, \tilde{\mathbf{h}}_{t+1}^{j-1}\mathbf{W}_k^T, \tilde{\mathbf{h}}_{t+1}^{j-1}\mathbf{W}_v^T,$$
$$\mathbf{h}_{t+1}^{j} = \text{Layer}(\mathbf{q}'^{j}_{t+1}, \mathbf{k}'^{j}_{t+1}, \mathbf{v}'^{j}_{t+1})$$

Hj at time t means the j-th layer of the hidden state at time t, while SG mean Stop-Gradient

Layer is Self attention layer and Feed Forward Layer and we start from initial hidden state equal to the row of B corresponding to exercise at time t

Attention Mechanism applied with input the state for Quey (exercise to predict result) and extended hidden state for Keys (previous exercises).

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

We implement multiple head attention, by training different matrices Wq, Wk, Wv and then we use them to obtain different Qi, Ki, Vi from B and E.

We get $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$ by the following equations:

$$\mathbf{q}_i = \hat{\mathbf{E}}_{i+1}\mathbf{W}_i^{\mathrm{Q}},\ \mathbf{k}_i = \hat{\mathbf{B}}_i\mathbf{W}_i^{\mathrm{K}},\ \mathbf{v}_i = \hat{\mathbf{B}}_i\mathbf{W}_i^{\mathrm{V}} \quad (2)$$

where $\mathbf{W}^{\mathrm{Q}}, \mathbf{W}^{\mathrm{K}}, \mathbf{W}^{\mathrm{V}} \in \mathbb{R}^{d\times d}$ are the query, key and value projection matrices.

In the end we compute Attention functions as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}$$

We need to MASK keys Kj for queries Qi such that: j > i
This avoids considering future interactions in the prediction.

$$\hat{\mathbf{B}} = \begin{bmatrix} \mathbf{B}_{x_1} + \mathbf{P}_1 \\ \mathbf{B}_{x_2} + \mathbf{P}_2 \\ \dots \\ \mathbf{B}_{x_n} + \mathbf{P}_l \end{bmatrix}, \hat{\mathbf{E}} = \begin{bmatrix} \mathbf{E}_{x_1} \\ \mathbf{E}_{x_2} \\ \dots \\ \mathbf{E}_{x_l} \end{bmatrix}$$

B has dimension Lu * d
E has dimension L * d

1. We split each sequence in smaller sequences of fixed lenght L
2. We train interaction embedding matrix B, where d is Hidden dimension.
3. We train Exercise embedding matrix E to embed current exercise we want to predict score. Bias is not needed, because can be inferred later in attention layer.
4. We create a positional encoding matrix P where i-th row represents relative distance between two positions and we sum it to B.

Inputs are sequences of already done exercises X1, X2,.... Xi, each composed by a couple (exercise ID, result). We conveniently modify classical input Xi in a single value:

$$Ui = Ei * (Amax + 1) - Ai$$

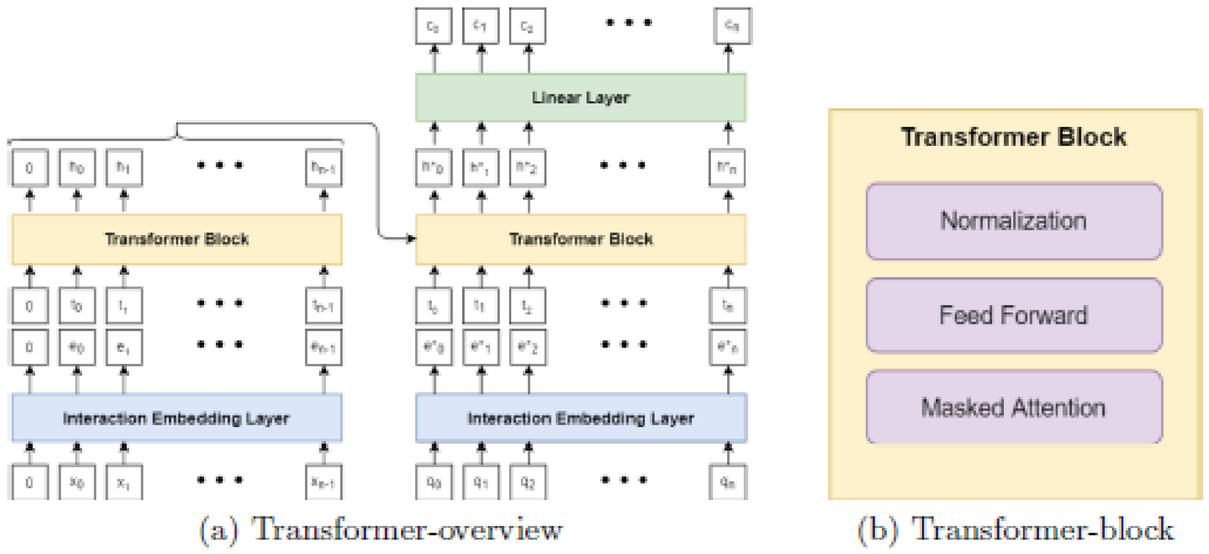
Where Ei is exercise ID and Ai result, while Amax is the maximum score of the exercise
So it can say that Ui can have a number of different values equal to:

$$Lu = (Amax+1) * L$$

where L is number of values of Ei, meaning the number of exercises

Figure 1: The architecture of KT-XL

DKT_Transformer: Deep Knowledge Tracing with Transformers



(a) Transformer-overview (b) Transformer-block

Fig. 1. Transformer architecture

Inputs:

Input are sequence tuples of Xi = { Qi, Ci } and timestamp Ti, where Qi represents an univoque exercise, while Ci the correctness of the answer.

GOAL: Predict

$$P(c_{i+1} = 1 | x_0, .., x_i, t_0, ..., t_i, q_{i+1}).$$

Embedding Layers:

Xi will be encoded as an INDEX Di and passed to an interaction mapping layer:

$$e_i = softmax(W_{d_i.})S$$

where:
W is matrix representing each row an exercise and each coloumn the weight of a latent knowldege concept for the exercise

weighted sum of all underlying latent skills

S is vector representation of Skills (Knowledge Conecpts)

Transformers Block - Masked Attention:

← This block is repeated in stack

$$q_i = Qe_i, k_i = Ke_i, v_i = Ve_i$$

Masked Attention Layer extracts query qi, Key ki and value vi from encoding ei using learnt matrices Q, K, V

$$A_{ij} = \frac{q_i k_j + b(\Delta t_{i-j})}{\sqrt{d_k}}, \forall j \leq i$$

For each previous interaction ej, from which we extracted key Kj, we calculate attention Aij for question ei to previous question ej.
It is computed as sum of two parts:
First part consider the degree of overlapping between latent description of Kj and ei.
Second part is TIME GAP BIAS

$$h_i = \sum_{j \leq i} softmax(A_{ij}) v_j$$

Hidden representation of exercise qi is computed as weighted sum of past value representations of ej multiplied for attention computed before Aij.

Then a FeedForward and a normalization layer are used and residual connections are added.

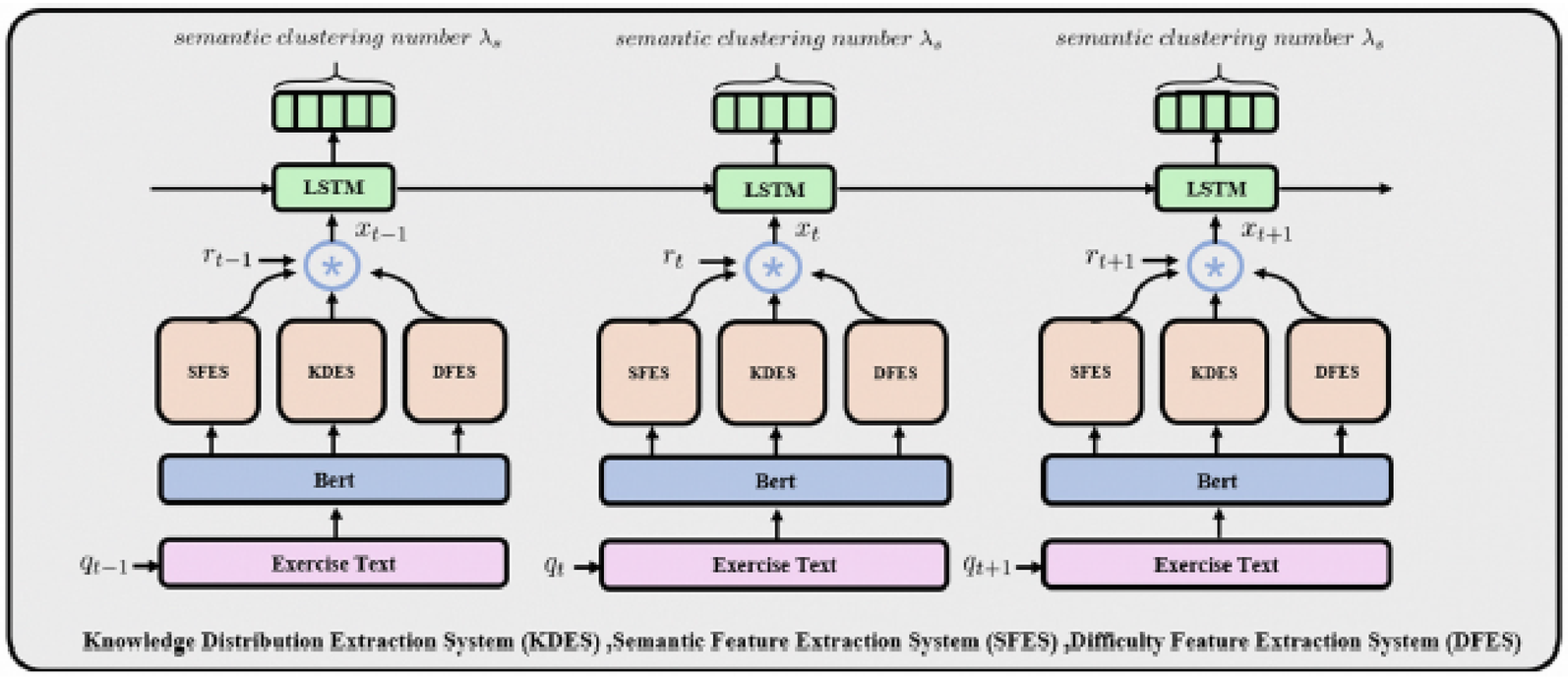
Linear Layer + Loss:

After a stack of transfomers block, output is fed to the LINEAR LAYER before calculating this LOSS function:

$$p_{i+1} = \frac{exp(h_i e_{i+1})}{exp(h_i e_{i+1}) + exp(h_i e^*_{i+1})}$$

$$Loss = -\sum_i c_{i+1} log(p_{i+1}) + (1 - c_{i+1}) log(1 - p_{i+1})$$

Exercise Hierarchical Feature Enhanced Knowledge Tracing



Input encoding:

Exercise recording of a student is a sequence of couples (Qi, Ri), where Qi represent the exerciseID and Ri the correctness, and we have to predict future result Rt for exercise Qt.

The exericise text of Qt is encoded using BERT into embedding vector Vb.
This vector is then feed to 3 different systems:

One for producing Knowledge Distribution Vt with dimension K. ⟶ KDES

One for producing question difficulty Dt. ⟶ DFES

One for producing SEMANTIC FEATURES St, represented by the one-hot encoding PHI(St). ⟶ SFES

The tuple of (Vt, PHI(St), Dt, Rt) will be the input tto our sequence model.

Subsystems description:

KDES and DFES are both text classification systems designed to predict respectively Knowledge Distribution and Diffuculty of exercise. They are predicted using TextCNN and serve as ground truth.
In KDES we use softmax results, while in DFES we use a neural network to predict difficulty to solve COLD START problem.
Semantic Features Extracion System (SFES) can be considered as an unsupervised clustering problem, solved by using Hiearchical Clustering method calculating COSINE DISTANCE between semantic vectors.

Modeling Process:

We use a RNN (in particular LSTM to preserve long-term dependency) to calculate subsequent hidden state Ht from previous one Ht-1 and current state Xt equal to tuple { Vt, PHI(St), Dt, Rt }.
TETAt is trainable parameter vector. ⟶

$$h_t, c_t = LSTM(x_t, h_{t-1}, c_{t-1}; \theta_t) \quad (1)$$

Then we use Ht as input to a sigmoid function with trained matrix and bias. ⟶

$$y_t = \sigma(W_{yh} \cdot h_t + b_y) \quad (2)$$

GOAL: Minimize Negative Log Likelihood of observed sequence of student responses ⟶

$$loss = -\sum_t (r_{t+1} * log(y_i^T \cdot \varphi(s_{t+1})) + (1 - r_{t+1}) * log(1 - y_i^T \cdot \varphi(s_{t+1}))) \quad (3)$$

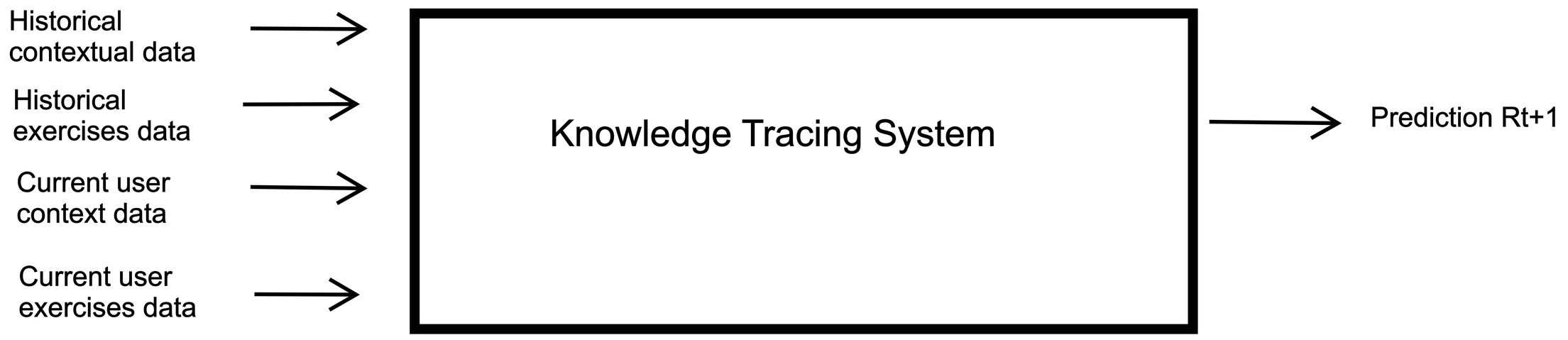
Results:

Adding hierarchical features can better represent questions; the reason is that exercises in same cluster have similar semantics, difficulty and knowledge representation
While we have shown the instability of knowledge tracing implicitely due to unpredictability of Difficulty fo exercises.

Personal ideas/suggestions:

Modular Generalization of Knowledge Tracing:

We can try define a general model to represent the common characteristics a Knowledge Tracing system should have in order to provide a structure for the models allowing to reduce the correctness prediction problem to a compostion of "smaller" problems, according to design pattern "DIvide Et Impera":



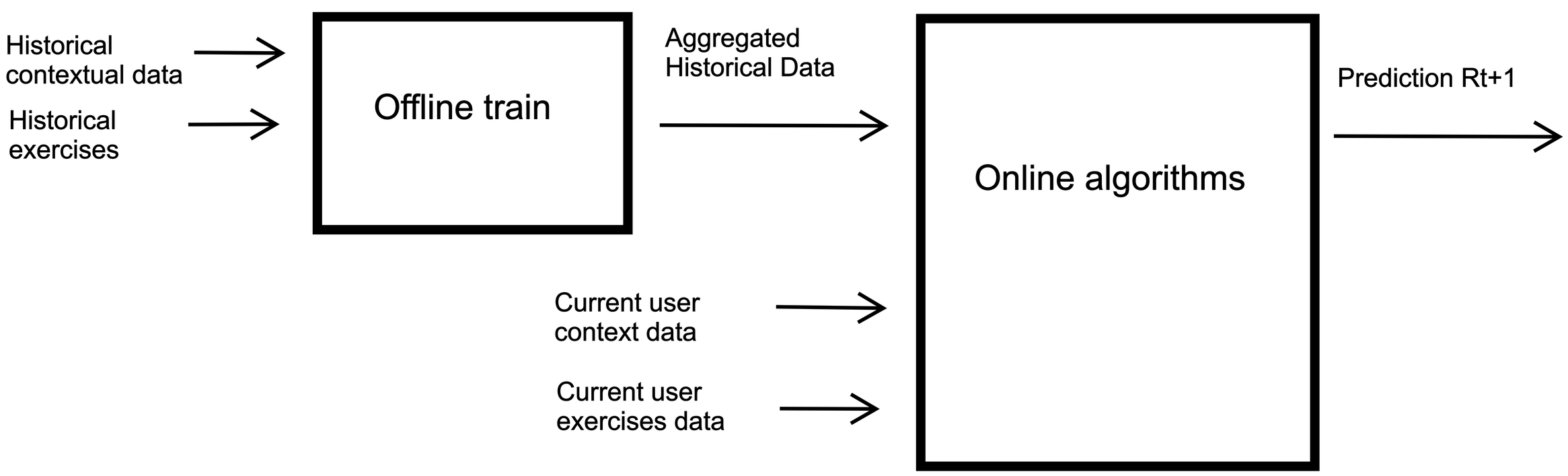
A generic model for KT has different inputs, which can be divided in four subsets:

1) Historical exercises data, which in general represent the total amount of available data about the interaction of users and exercises in the past and their characteristics. This is a dataset to be mantained offline.

2) Historical contextual data, which represent the total amount of available data about the context of past data. It can contain aggregated extracted information, computed offline. This is a dataset to be mantained offline.

3) Current user exercise data, they represent previous interactions between user and exercises. They need at least to contain correctness value and possibly timestamp, but they can contain many other useful data.

4) Current user contextual data, which contain any data known about the user, such as age, field of study, and hypotethically should contain Knowledge state estimation.



We have divided the algorithms in two main classes:

1) Offline algorithms, which take as input Historical data (both generic and user specific), and return in general aggregated data which will be used by online algorithms. They are used to train paramters and models to be used online.
These algorithms can be performed only periodically and not each time an user answer to an exercise, due to the memory and time performances.
They contain aggregated information learnt from past and should try to detect patterns and elements to help online algorithms.

2) Online algorithms, which should be evaluated each time the Current user answer a new exercise; they are personalized algorithms, which use as input both personalisez Current user data and ggregated data from offline algorithms.
In general they should be less computational and memory expensive, because they should be evaluated fast to allow online decisions for the application.

Offline algorithms:

The mojority of algorithms always require an offline computation phase.
From literature we have many examples of offline algorithms already efficiently developed:

1) Exercise Relation Modeling from RKT is a clear example of offline algorithms in which we use as input both Historical Exercise Data (called in the paper performance data, and which are simply the tuples {Exercise, Correctness, Timestamp}) and Historical Context Data (in particular they use exercise textual content).
The outputs are the coeficients Aij representing correlation between exercises i and j.
Actually this method can be seen as an hybrid combination (linear combination of coefficients) of 2 different methods: a collaborative filter using the Phi coefficient between exercise performances, and a content filter using cosine similarity to calculate content similarity.
In the end these two methods are simply summed but in reality they could produce two different Exercise Relation Matrices.

2) Transformer attention mechanism from KT-XL, DKT_Transformers, RKT and SAKT consists of using train to obtain some embedding matrices to embed key, query and vector.
So they can be considered as offline algorithms which produce matrices which will be then used to embed the exercises (online algorithm).

3) From EHFKT we can notice the usage of different offline algorithms:

1) BERT pretrained model to embed textual content

2) KDES, responsible of estimating Knowledge Distribution of an exercise from textual information

3) DFES, responsible of estimating

4) SFES, responsible of Clustering exercises according to hierachical features

All of them are offline algorithms which take as input the text of an exercise and produce different outputs useful for online algorithms.

4) Obviosuly any other algorithms requiring a train phase on Historical data is considered as an offline algorithm

Online algorithms:

Instead online algorithms use Current user data and offline computed parameters/models to predict the output.
Online algorithm can use different models and in general should provide ways of managing all the input to predict the output.

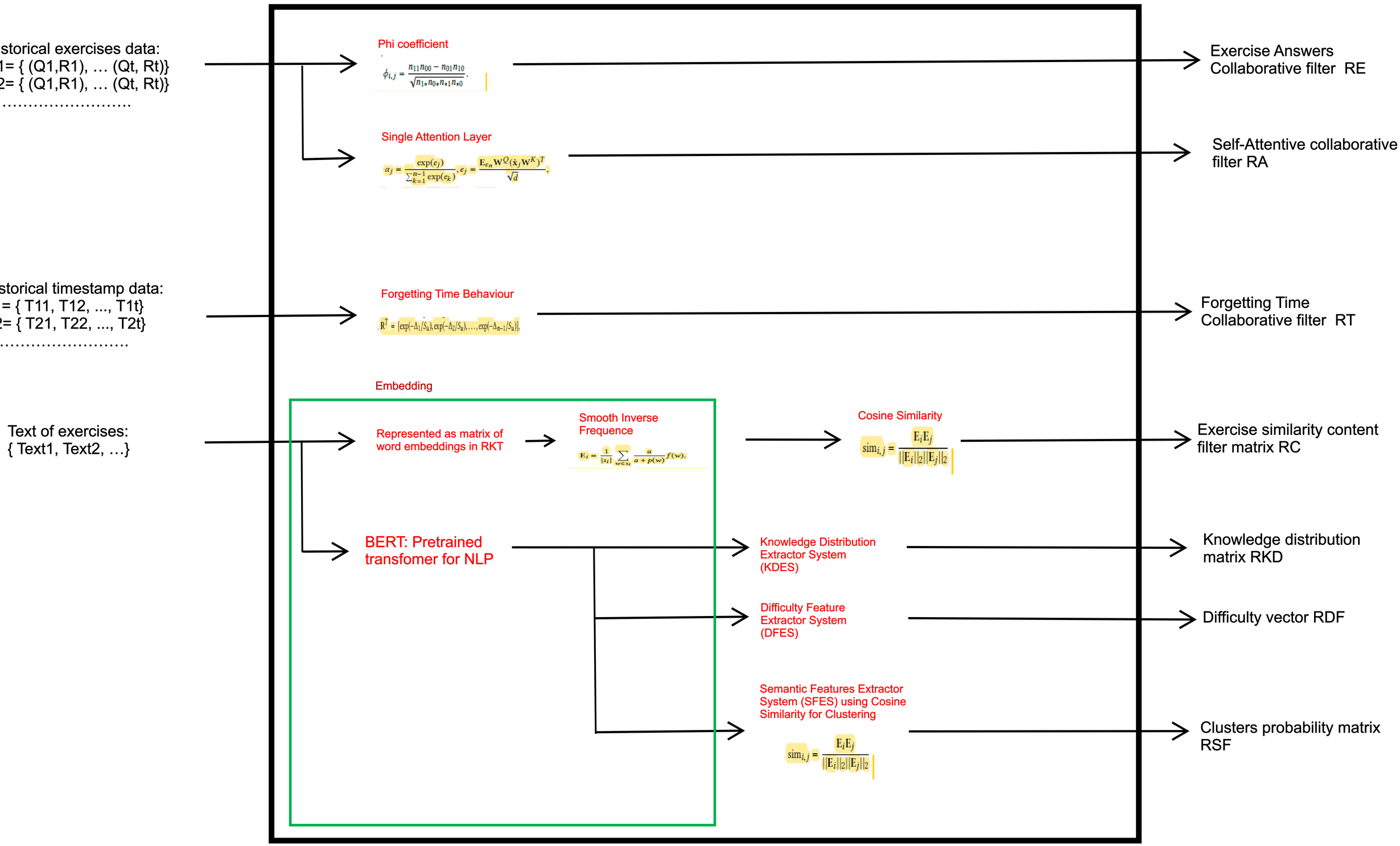
From literature we mainly see online algorithms only able to use pretrained paramters/models, as for example:

1) Attention algorithms use pretrained embedding matrcies and weights and just furnish Current user data as input to obtain embeddings and in generale values. These values are usually given to a Feed Forward Neural Network to allow non-linearities and in the end to a pre-trained Sigmoid function.

2) Same for RNNs and LSTM. In which input is used to calculate online hidden state and predicted output.

Forgetting time behaviour modelization is main example of using current user exercises data, to train a parameter to model user tendence to forget concepts.
In RKT this parameter is trained offline, but pheramps it could be possible to refine it better online each time a new exercise is inserted.

Offline algorithms:



The algorithm seen in literature can be all seen as hybrids of these procedures and can be combined to try new systems:

1) RKT computes Exercise Answers Collaboritive Filter and Exercise similarity content based filter matrices: RE and RC, and simply sum them obtaining:
R = RE+RC. Then it computes Forgetting Factor Colllaborative filter RT and sums, using a weight combination, it to R' obtaining matrix R''.
Then it applies softmax to R''.
In the end it computes attention weights, based on some trained matrices, and sums to R'', applying softmax too.
RKT should be considered as a weight combination of 4 different filters: Exercise Answers CF, Exercise similarity CBF, Self-Attentive Matrix and Forgetting Factor Collaborative filter.
So I propose to express it this way:

$$R = (Alpha*RE + Beta*RC + Gamma*RA + Delta*RT) /(Alpha+Beta+Gamma*Delta)$$

And try to train model this way to improve performance

2) We can easily see from previous figure that EHFKT can be considered as a substitute of Exercise Similarity Context Based filter matrix RC.
So we should try to compare the results obtained from these 2 methods, understand the best and eventually try unifying them with an HYBRID MODEL.
Thus hybrid can be created by:

1) Simplest solution would be to use all these data concatenated as hidden state of LSTM, but it would be impossible to adapt to RKT structure.

2) We could extend RKT by passing to attention mechanism the matrices RDF, RKD; RSF. This could be a simple but effective solution.

3) From DKT_Transfomer we could think of using this obtaned inputs (data, RE, RT, RDF, RKD, RSF) as inputs to a Transformer.

4) From KT_XL we could think of using hidden state to model knowledge from previous sequence for RKT.

5) Other ideas could be adding P3ALPHA or others graph based methods too.

6) In the end, but probably the most important, SAINT is a new architecture which applies a transfomer separating econdere and decoder layers. It could be much useful to be applied to already found inputs produced by other Offline algorithms.

Results:

1) RKT has results with dataset divided in train(0.6), validation(0.2), test (0.2) sets:

```
Step 4706, {'loss/train': 0.5666489096268716, 'auc/train': 0.7613245600527567, 'auc/val': 0.7485272460633234, 'acc/val': 0.6890745488837927}
{'loss/train': 0.5666489096268716, 'auc/train': 0.7613245600527567, 'auc/val': 0.748527246063234, 'acc/val': 0.6890745488837927}
```

```
(5830548,)
(5830548,)
(5830548,)
auc_test =  0.7568187349448214
```

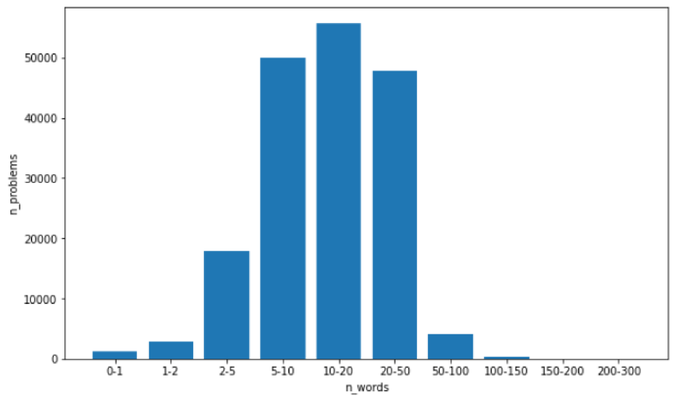
+Code +Markdown

Dataset characteristics and preparation:

1) Assistments Problems:

Problems we have textual information: 179950
Average number of words for problem: 17.28

Different words: ? , stopwords: ?, without stowpwords: 27742



Assistments 2012/13 users interactions:

Problems we have user interactions: 45759

Intersection with problems we know textual information too: 4048 --> not enough probably

WORD2VEC using Gensim:

Vocabulary: 13818 words
Time to train a model: around 1 minute

Hyp: epochs:10

vector_size: 60 -> Accuracy: 0.665156
vector_size: 120 -> Accuracy: 0.664149
vector_size: 180 -> Accuracy: 0.663645
vector_size: 240 -> Accuracy: 0.664652
vector_size: 300 -> Accuracy: 0.663645

We can understand that 60 vector_size is enough so we try with less:
vector_size: 5 -> Accuracy: 0.661631
vector_size: 10 -> Accuracy: 0.660120
vector_size: 15 -> Accuracy: 0.660121
vector_size: 20 -> Accuracy: 0.662638
vector_size: 25 -> Accuracy: 0.662134
vector_size: 30 -> Accuracy: 0.664149
vector_size: 35 -> Accuracy: 0.663645
vector_size: 40 -> Accuracy: 0.662638
vector_size: 45 -> Accuracy: 0.662134
vector_size: 50 -> Accuracy: 0.66364

We can see that vector_size 60 seems the best value.

Fixing 60 as vector size and changing number_of_epochs only decreases accuracy, so 10 epochs is enough with 60 as vector_size.

Now I try increasing vector size and finding which is

ZVocabulary: 13818 words
Time to train a model: around 1 minute

Hyp: vector_size: 180

epochs: 10 -> Accuracy: 0.666163
epochs: 20 -> Accuracy: 0.663141
epochs: 20 -> Accuracy: 0.663141

Hyp:
vector_size 180, epochs 40 -> 0.664149

We can see that augmenting vector size or number of epochs does not increase similarity anyway I will take best 3 models:

1) size: 60 epochs 10
2) size: 180 epochs: 10
3) size: 300 epochs: 40

TF_IDF using TF_vectorizer:

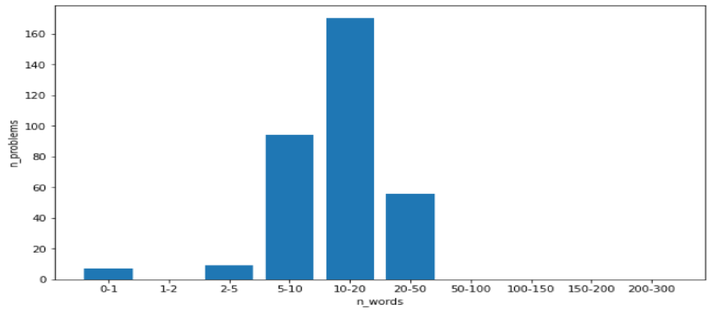
Vocabulary: 27742 words

Accuracy: 0.5795568982880162

1) Junyi Academy:

Problems we have textual information: 840
average words for problem: 5.5738

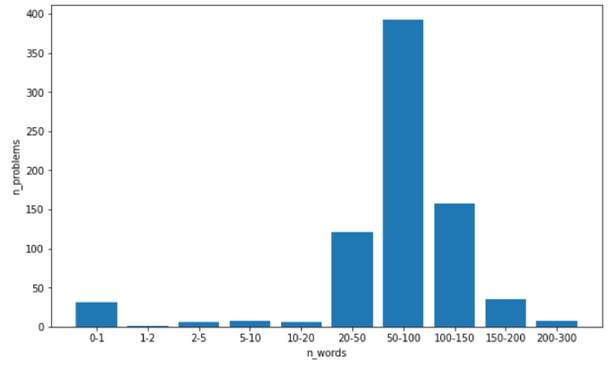
Different words: 1234 stopwords: 90, without stowpwords: 1144



1) POJ:

Problems we have textual information: 774
Average number of words for problem: 80.3410

Different words: 9655, stopwords: 142, without stowpwords: 9513



Cleaning techniques: