

# SLIM and factorization machines

**Vladislav Goncharenko**

Material by Andrey Zimovnov



MSU,  
spring 2024

# Lecture plan

- SLIM
- Factorization machines
- Content-based recommendations
- Neural network recommendations.



# Sparse Linear Methods

---

**girafe**  
**ai**

**01**

# SLIM – Sparse Linear Methods



- $\mathbf{A}$  is a binary matrix of  $\mathbf{M} \times \mathbf{N}$  user-item interactions
- We will define the  $\mathbf{a}_{ui}$  interaction as weighing events from the past:

$$\hat{a}_{ui} = \sum_{j=1}^N w_{ij} a_{uj}$$

- Weights  $\mathbf{w}_{ij} \geq 0$ , that is, the model takes into account similar items. For example, for a photo with a cat, it is much easier to say who looks the most like it than who looks the least like it
- Moreover,  $\mathbf{w}_{ii} = 0$  – allows you to explicitly avoid an elementary solution  $\mathbf{W} = \mathbf{I}_N$
- Thus,  $\mathbf{w}_{ij}$  is a similarity score  $\mathbf{j}$ -th item to  $\mathbf{i}$ -th



# SLIM – Sparse Linear Methods

- Optimized MSE loss with **L1** and **L2** regularizations:

$$\frac{1}{2} \sum_{u,i} \left( a_{ui} - \sum_j w_{ij} a_{uj} \right)^2 + \lambda \sum_{i,j} |w_{ij}| + \frac{\beta}{2} \sum_{i,j} (w_{ij})^2 \rightarrow \min_W$$

- Note that according to the lines **W****i**, the task is divided into m independent:

$$\frac{1}{2} \sum_u \left( a_{ui} - \sum_j w_{ij} a_{uj} \right)^2 + \lambda \sum_j |w_{ij}| + \frac{\beta}{2} \sum_j (w_{ij})^2 \rightarrow \min_{w_{i1}, \dots, w_{iN}} \quad (\forall i)$$

- We can solve each problem by coordinate descent:

1. fix all **W****i** except one coordinate **w<sub>ij</sub>**

2. go to the optimum by **w<sub>ij</sub>**

go to the next coordinate

4. repeat until convergence

3.

# SLIM – Sparse Linear Methods



- The process of building recommendations:

1. We take the vector of user interactions  $\mathbf{A}_u$

2. We count  $\hat{a}_{ui}$  for all unseen items

3. We sort the unseen items by  $\hat{a}_{ui}$  and take the top products with the highest value

- Due to the presence of **L1**-regularization, the matrix **W** will be sparse;
- The event matrix **A** is also sparse;
- This makes it possible to significantly speed up the asymptotics of the model application

$$\hat{a}_{ui} = \sum_{j=1}^N w_{ij} a_{uj}$$

# Factorization Machines

---

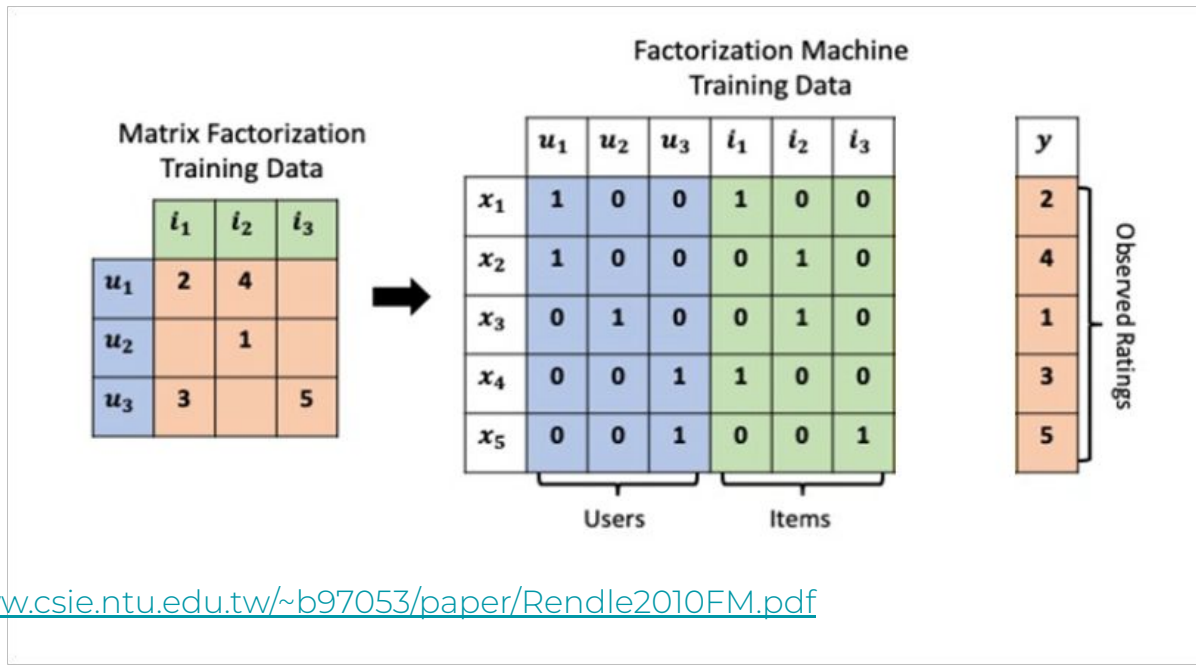
**girafe**  
**ai**

02



# Factorization Machines

- Let  $x \in \mathbb{R}^n$  — one-hot be the interaction vector of the user-item pair, where 1 stands in place of the corresponding user and product ( $n = |U| + |I|$ ):







# Factorization Machines

- Let's consider the regression model in this statement:

$$a(x) = w_0 + \sum_{i=1}^n w_i x_i$$

- We will add second-order interactions to our regression model, which will allow us to take into account more complex relationships between features:

$$a(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n w_{ij} x_i x_j$$

- In the resulting model,  $n(n+1)/2 + n + 1$  parameters;
- Since  $n = |I| + |U|$ , the model size becomes too large.



# Factorization Machines

$$a(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n w_{ij} x_i x_j$$

- Let's compare each feature with the  $\mathbf{x}_i$  vector  $v_i \in \mathbb{R}^k$  and present the model in the form:

$$a(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle v_i, v_j \rangle x_i x_j$$

- The number of model parameters decreased to  $\mathbf{nk + n + 1}$ ;
- The last term can be calculated as  $\mathbf{O(nk)}$ :

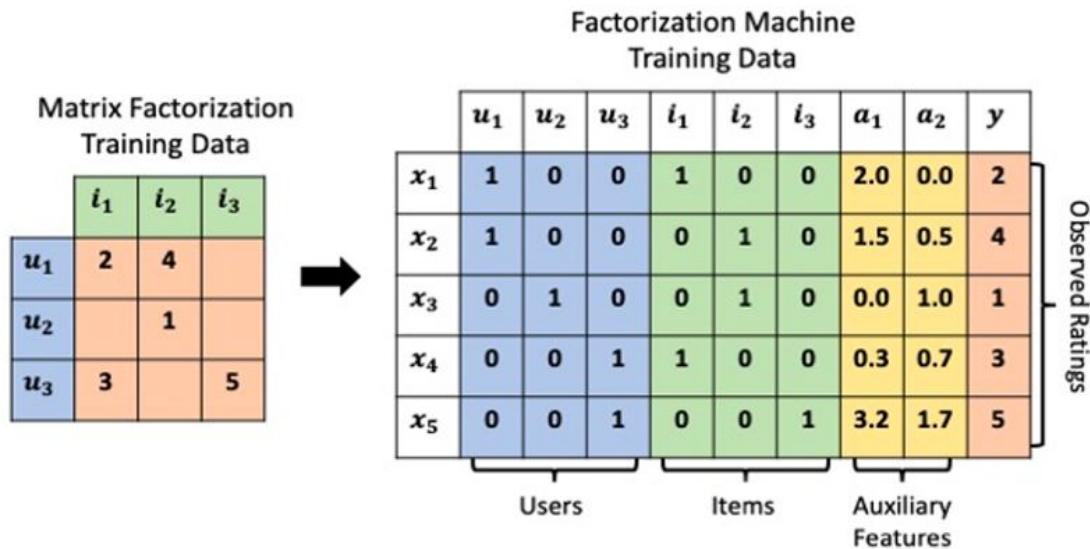
$$a(x) = w_0 + \sum_{i=1}^n w_i x_i + \frac{1}{2} \left\| \sum_{i=1}^n v_i x_i \right\|_2^2 - \frac{1}{2} \sum_{i=1}^n \|v_i\|_2^2 x_i^2$$

- Such a model is called a factorization machine.



# Factorization Machines

- In addition to one-hot coded interaction, you can add content attributes of a user or product to vector  $\mathbf{x}$ :





# Factorization Machines

$$a(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle v_i, v_j \rangle x_i x_j$$

- The model is trained by gradient descent;
- The main point of factorization machines is that the weights for paired interactions of features are factorized;
- In addition, the predictions of the model can clearly be considered linearly, which gives a good performance of the algorithm.

# Field-aware Factorization Machines

---

**girafe**  
**ai**

**03**

# Field-aware Factorization Machines



- Example: there are 3 signs that are different in nature: year of manufacture, color and model of the car;
- In the FM model, the same vector for the year is used to account for the year-color and year-brand interaction;
- But since these signs are different in meaning, the nature of their interaction may differ;
- Idea: use 2 different vectors for the “year of manufacture” attribute when taking into account the year-color and year-brand interactions;

Feature vector $\mathbf{x}$															Target $y$							
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0	0	0	1	...	13	0	0	0	0	...	5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	1	0	0	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0	1	0	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	1	0	0	...	5	0	0	0	0	...	4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	1	0	...	8	0	0	1	0	...	5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0	0	0	1	...	9	0	0	0	0	...	1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0	1	0	0	...	12	1	0	0	0	...	5	$y^{(6)}$

# Field-aware Factorization Machines



- Let's divide the signs into groups, let  $f_i$  be the index of the group  $i$  of that sign;
- Then the FFM model looks like:

$$a(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle v_{i,f_j}, v_{j,f_i} \rangle x_i x_j$$

- It is trained by gradient descent, similar to FM;
- Similarly, the quadratic sum can be calculated linearly by  $\mathbf{n}$ ;
- They work best with groups of the type “categorical sign of large cardinality”;

# Neural networks in recommendations

---

**girafe**  
**ai**

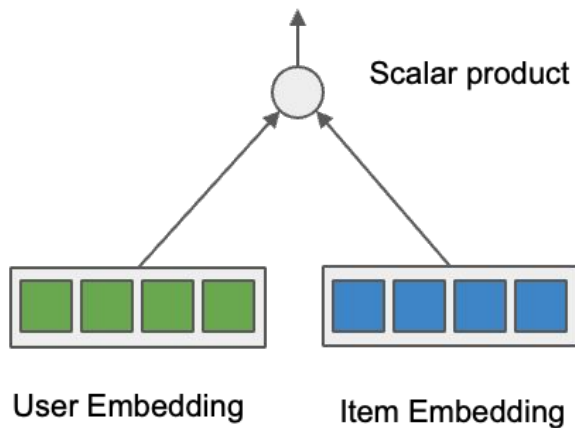
**04**



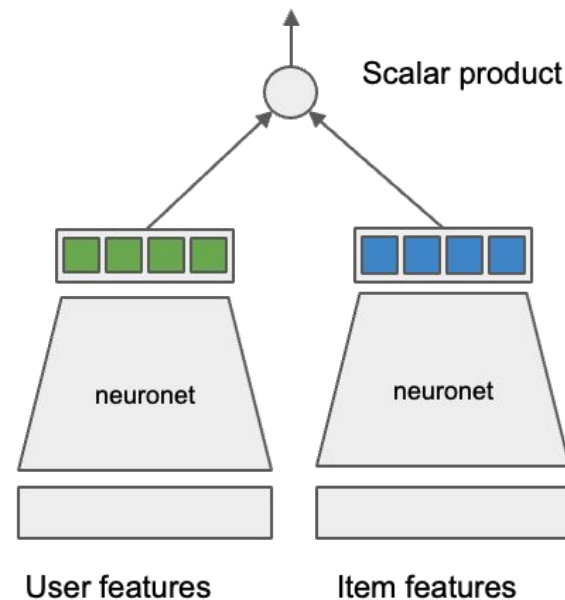
# SVD is a kind of neural network



Let's go back to the matrix decomposition and think about how we can take into account additional features in it.



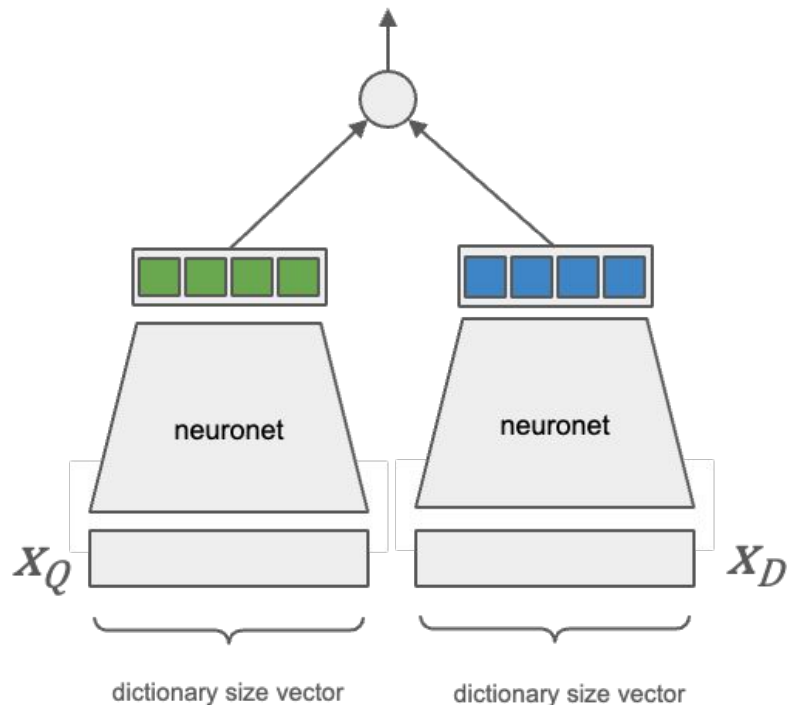
We teach using SGD





# Deep Structured Semantic Models

$$R(Q, D) = \text{cosine}(y_Q, y_D) = \frac{y_Q^T y_D}{\|y_Q\| \|y_D\|}$$



DSSM is a classic search and ranking model

- **Q** – text query, **D** – document
- $x_Q$  and  $x_D$  are their representations, for example, in the form of bag of words (~100k)
- The request and the document are translated by neural networks into embeddings of a small size (~300)
- Between them we consider the proximity function, cosine, scalar product, etc
- We rank documents by the proximity value



# Deep Structured Semantic Models: how to train?

We will consider the conditional probability of a click on the document  $\mathbf{D}$  under the condition of the query  $\mathbf{Q}$ .

$$P(D|Q) = \frac{\exp(\gamma R(Q, D))}{\sum_{\mathbf{D}} \exp(\gamma R(Q, D))}$$

$$R(Q, D) = \text{cosine}(y_Q, y_D) = \frac{y_Q^T y_D}{\|y_Q\| \|y_D\|}$$

Here

$\gamma$  – k-t smoothing, established empirically

$\mathbf{D}$  – the set of all documents

**Calculating the gradient of such a functional for each example is expensive. What can be done?**

# Deep Structured Semantic Models: how to train?



We will consider the conditional probability of a click on the document  $\mathbf{D}$  under the condition of the query  $\mathbf{Q}$ .

$$P(D|Q) = \frac{\exp(\gamma R(Q, D))}{\sum_{\mathbf{D}} \exp(\gamma R(Q, D))}$$

$$R(Q, D) = \text{cosine}(y_Q, y_D) = \frac{y_Q^T y_D}{\|y_Q\| \|y_D\|}$$

Here

$\mathbf{y}$  – k-t smoothing, established empirically

$\mathbf{D}$  – the set of all documents

Negative sampling options:

1. It is equally likely to select a subset of documents from non-selected ones
2. It is more likely to choose those unlisted documents whose popularity is higher
3. At each epoch of training, choose non-called documents with the maximum score (the score is taken from the previous epoch)



# Deep Structured Semantic Models: how to train?

Taking into account negative sampling, the probability of a click in the document is described by the formula

$$P(D|Q) = \frac{\exp(\gamma R(Q,D))}{\exp(\gamma R(Q,D)) + \sum_{d \in \mathbf{D}^-} \exp(\gamma R(Q,d))}$$

During the training process, we will maximize the likelihood of the sample, or, what is the same thing, minimize the loss:

$$L(\Lambda) = -\log \prod_{(Q, d \in \mathbf{D}^+)} P(d|Q)$$

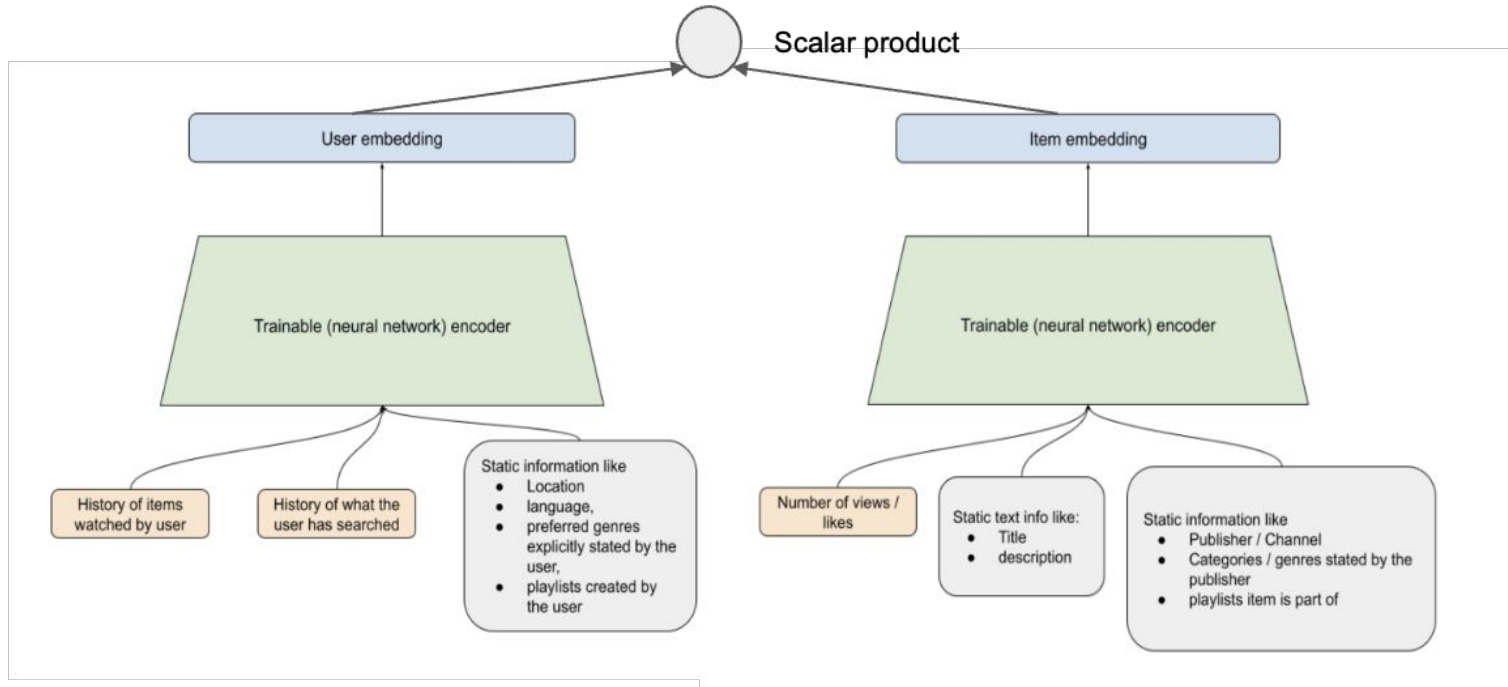
where  $\Lambda$  are the parameters of the layers

$\mathbf{D}^+$  – a lot of clicked documents

$\mathbf{D}^-$  – lots of negative sampling documents



# A quantum leap from search to recommendations



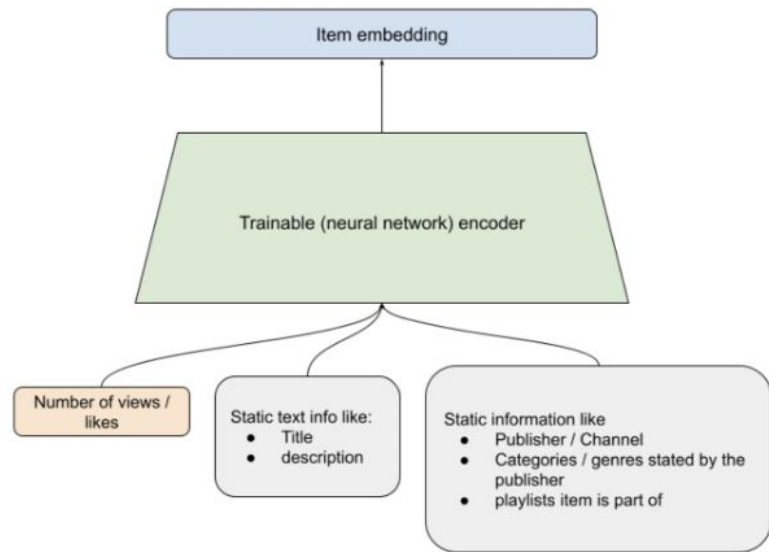
The search for relevant items can be presented as a ranking task, where the user, his history and features are used as a query.



# Document features

As signs, you can use:

- standard document statistics: number of likes, clicks, subscriptions
- author's signs: number of subscribers, genre
- unstructured data: document text (you can use BOW format, or you can use it before trained embeddings), videos and pictures (also pre-submit them as embedding)



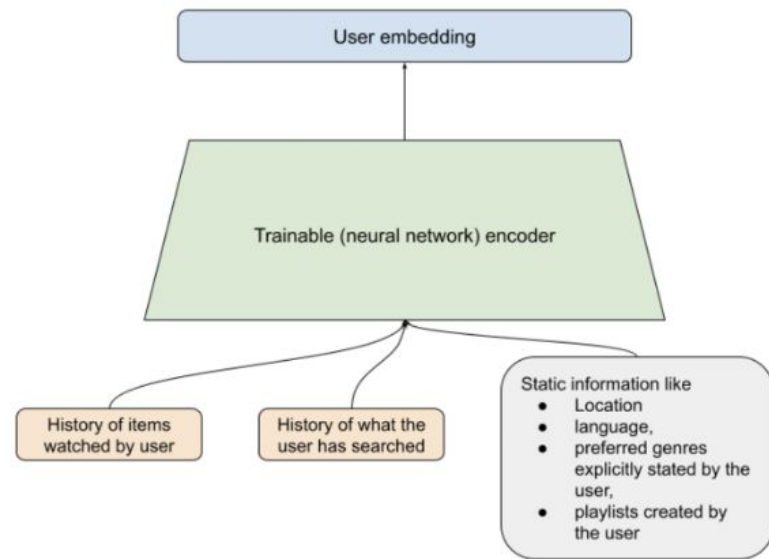


# User features

As features, you can use:

- information about the user: age, gender, language, how long the service
- has been using information about the context of the request: from which device was made, at what time
- information about the user's friends/subscribers and their interactions

It makes sense to use the user's history as an average embedding of those articles that he read. Or train RNN or Transformer on history and concatenate the result to the rest of the features.





# Learning strategies

---

**girafe**  
**ai**

**05**

# Two-tower neural network: types of losses



- MSE
- Cross Entropy Loss (CE)
- Pairwise loss
- Full Product Softmax loss (aka: Infancy, InfoMAX, SINCLAR)



# Cross entropy loss

The probability that user  $u$  will click on item  $i$  can be represented as:

$$\hat{p}_{ui} = \sigma(R(u, i)) = \sigma(\text{dot}(y_u, y_i))$$

Where:

$y_i$  – embedding of aitema

$y_u$  – embedding the user

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Then the loss has the form (  $r_{u,i} \in \{0, 1\}$  – rating):

$$L = - \sum_{u,i} (r_{ui} \log \hat{p}_{u,i} + (1 - r_{u,i}) \log(1 - \hat{p}_{u,i}))$$



# Pairwise losses

Consider a pair of items in which  $i_1$  – positive,  $i_2$  – negative, there are several options for pairwise loss:

$$L(R(u, i_1), R(u, i_2)) = \text{CrossEntropy}(1.0, \sigma(R(u, i_1) - R(u, i_2)))$$

---

the network learns to rank positive examples above negative ones

$$L(R(u, i_1), R(u, i_2)) = \max(0, \alpha - R(u, i_1) + R(u, i_2))$$

---

the network makes sure that the positive and negative examples differ as much as possible (known as triplet loss, which is used to train Siamese networks)

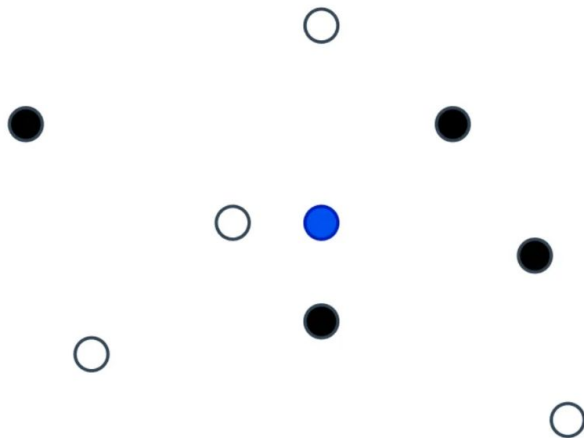
<https://www.v7labs.com/blog/contrastive-learning-guide>

<https://medium.com/@maksym.bekuzarov/losses-explained-contrastive-loss-f8f57fe32246>

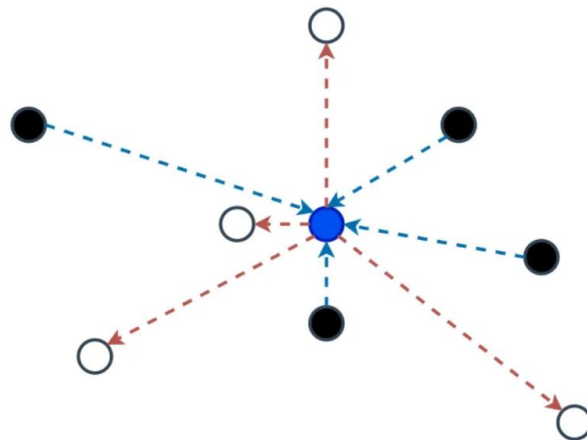


# Pairwise losses

● - points, similar to ●  
○ - points, dissimilar to ●



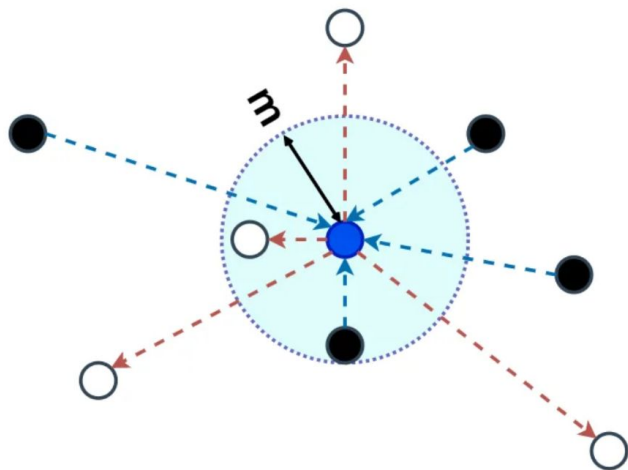
● - points, similar to ●  
○ - points, dissimilar to ●



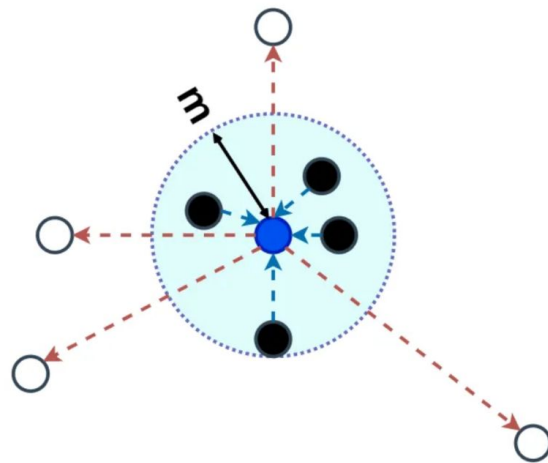


# Pairwise losses

● - points, similar to ●  
○ - points, dissimilar to ●

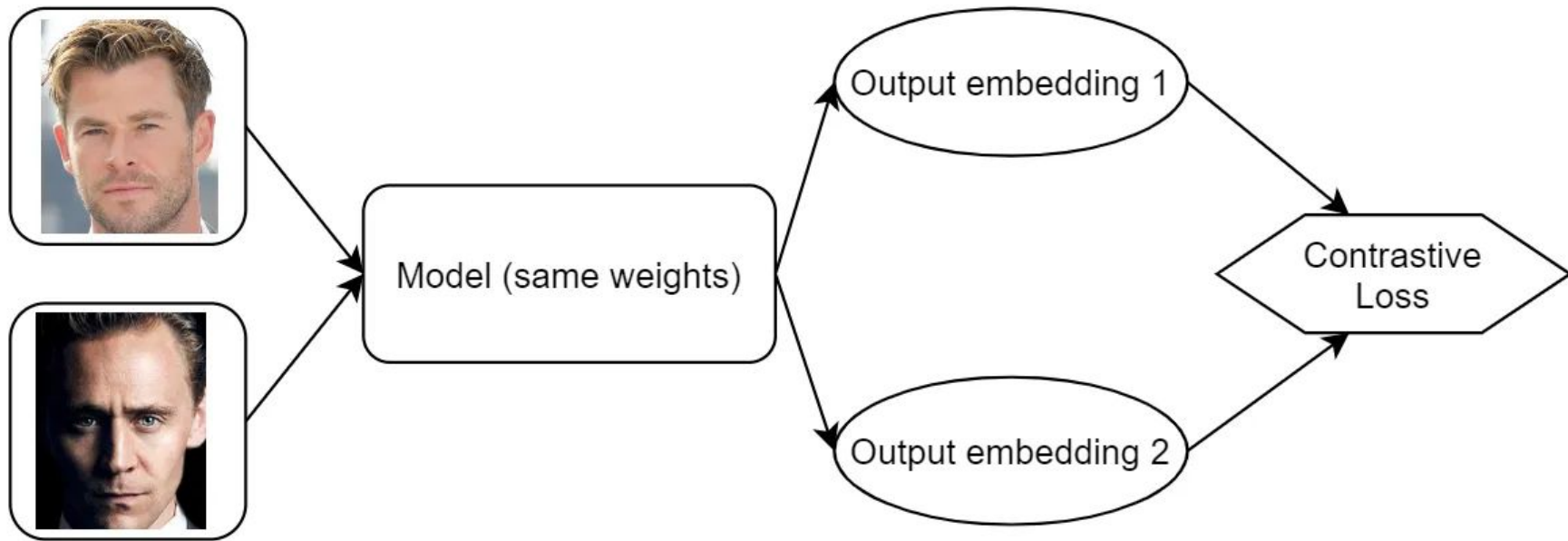


● - points, similar to ●  
○ - points, dissimilar to ●



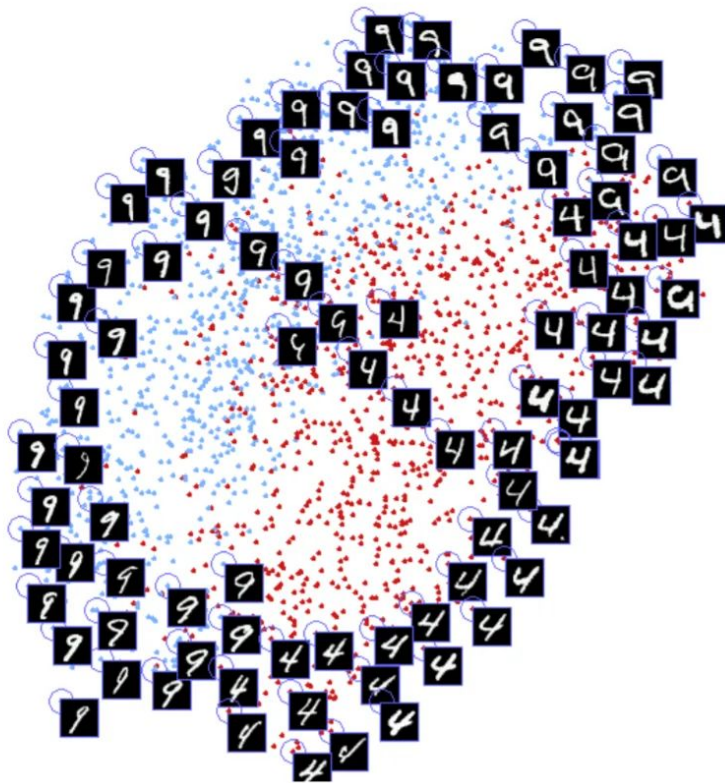


# Pairwise losses





# Pairwise losses







# Full Product Softmax loss

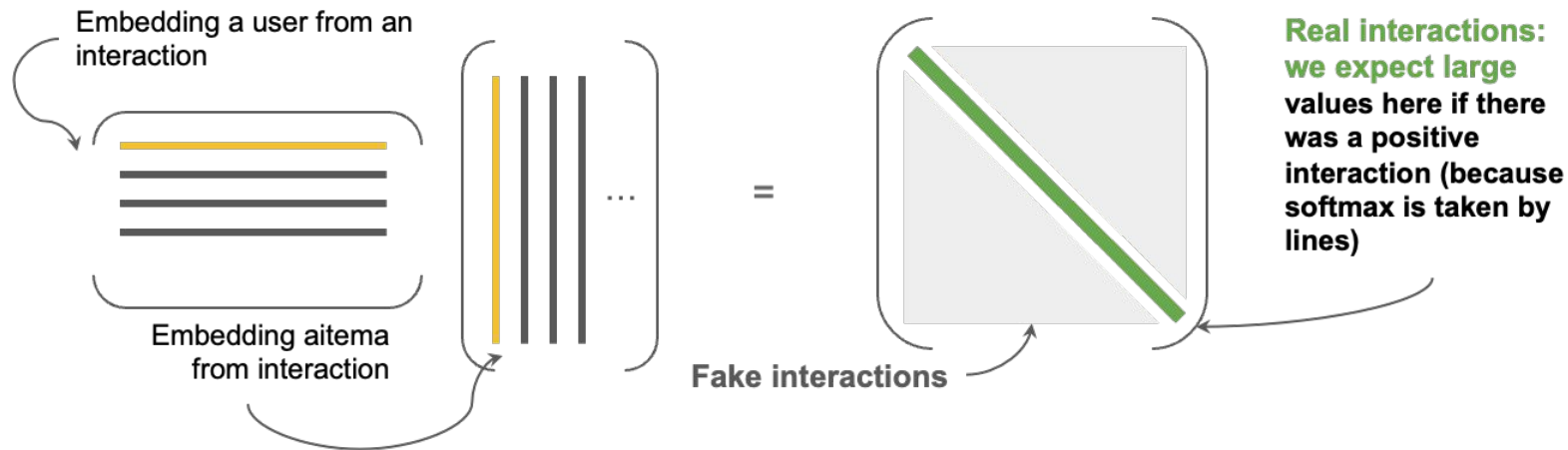
Consider a batch of interactions of size  $m$ , consisting of

User embedding matrices  $U \in \mathbb{R}^{m \times d}$

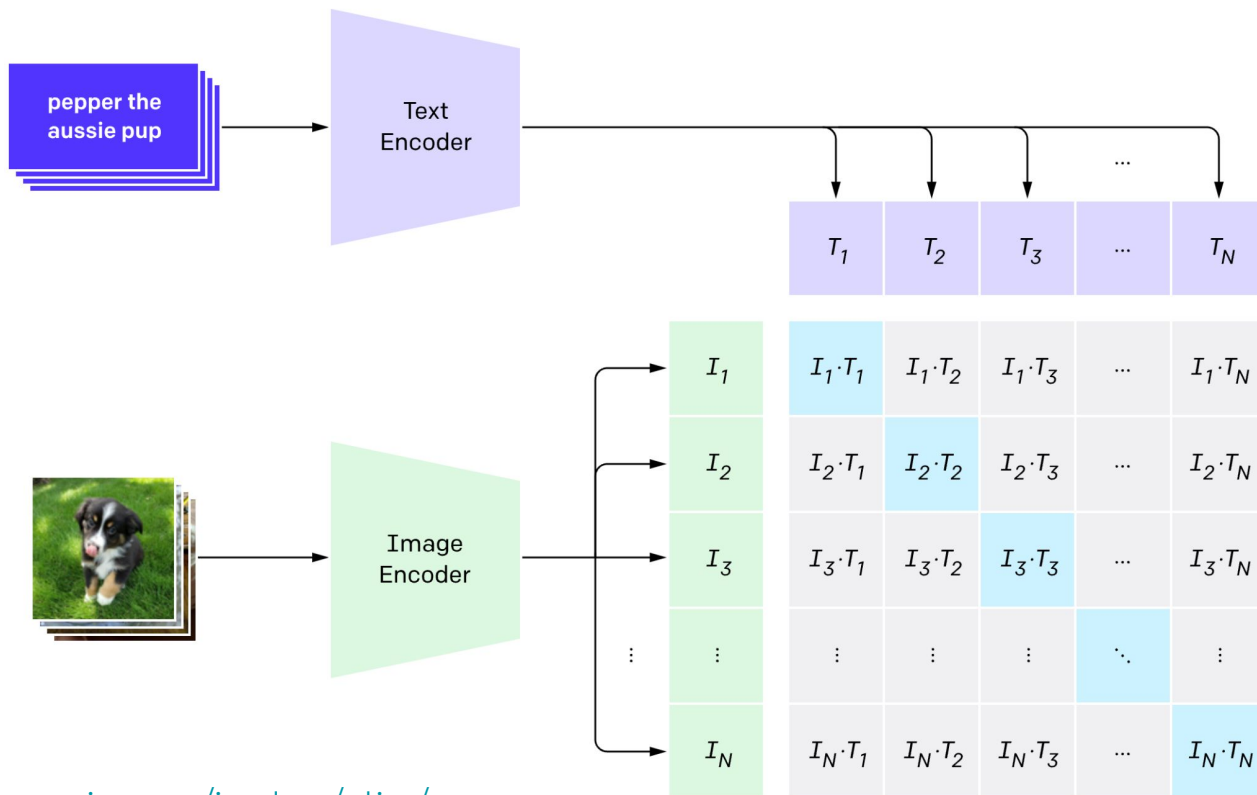
Embedding matrices of aitems  $I \in \mathbb{R}^{m \times d}$

Target vectors  $r \in \mathbb{R}^m$

Consider a matrix  $\text{Softmax}(\alpha \cdot UI^T + \beta)$ ,  $UI^T \in \mathbb{R}^{m \times m}$  where softmax is taken by rows.

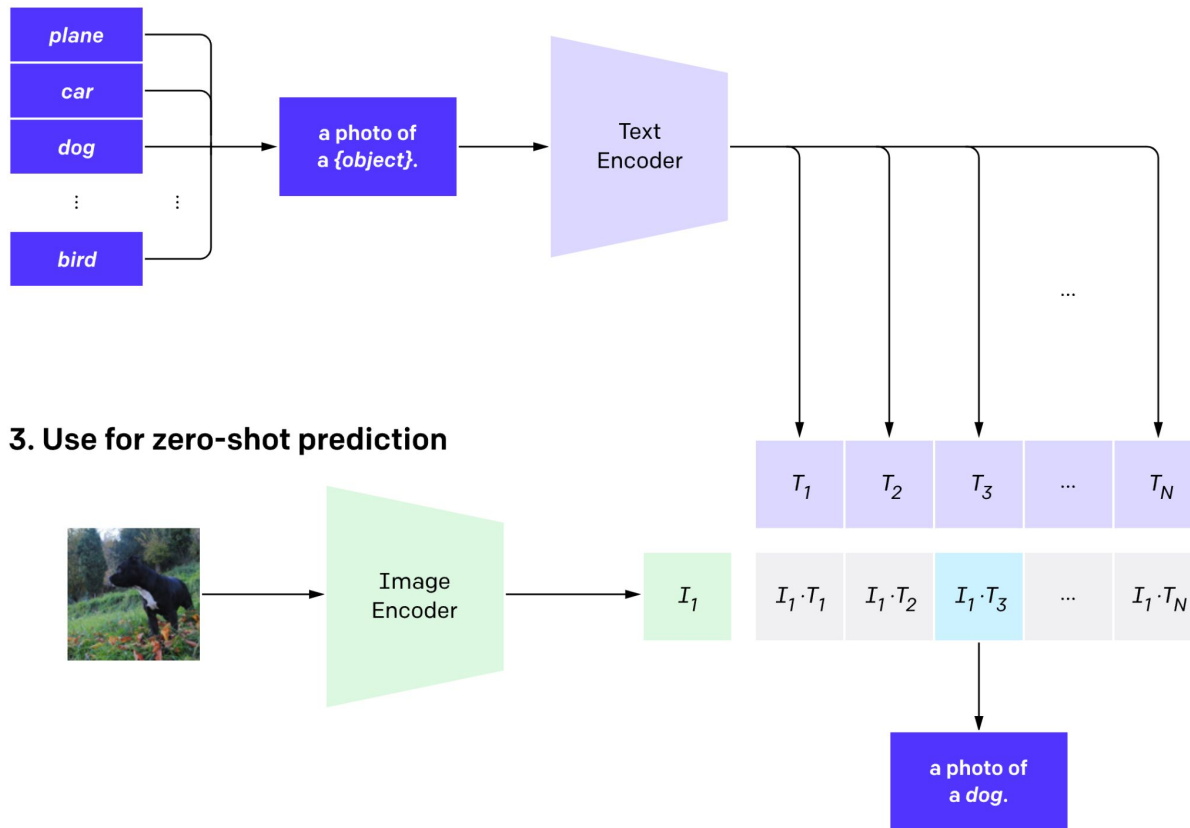


# CLIP pre-training



<https://openai.com/index/clip/>

# CLIP usage





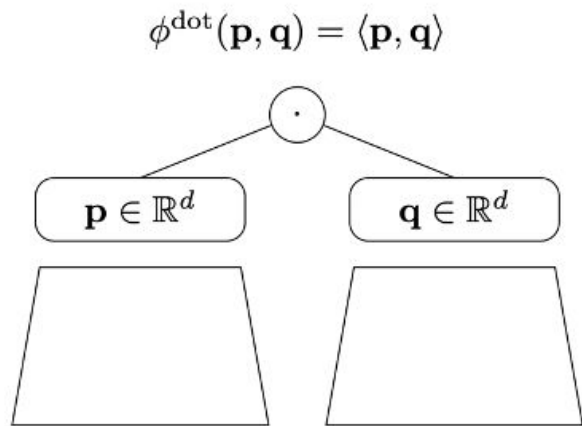
# Full Product Softmax loss

Consider a loss of the form

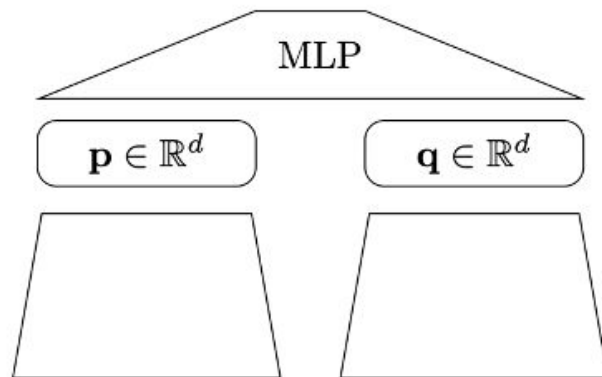
$$L = -(r > 0)^T \cdot \log(\text{diag}(\text{softmax}(\alpha \cdot UI^T + \beta)))$$

The loss makes the diagonal elements of the matrix larger than the rest of the elements: so in a dataset with unique users and documents on the diagonal of the optimal matrix will be  $r > 0$

# What else have we tried?



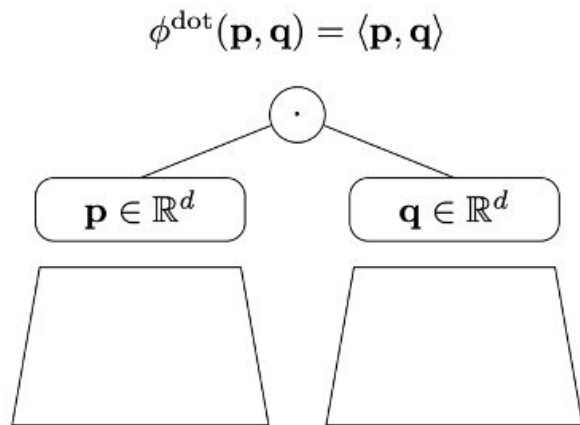
$$\phi^{\text{MLP}}(\mathbf{p}, \mathbf{q}) = \mathbf{f}_{W_l, \mathbf{b}_l}(\dots \mathbf{f}_{W_1, \mathbf{b}_1}([\mathbf{p}, \mathbf{q}]) \dots)$$



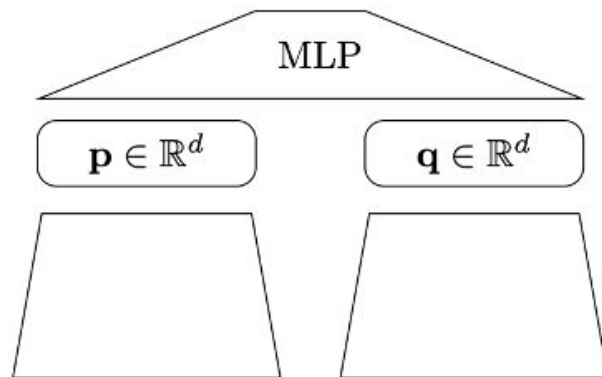
1. Users and items have learnable embedding
2. Embedding is concatenated and passed through MLP
3. The neural network provides a prediction of the rating



# What else have we tried?



$$\phi^{\text{MLP}}(\mathbf{p}, \mathbf{q}) = \mathbf{f}_{W_l, b_l}(\dots \mathbf{f}_{W_1, b_1}([\mathbf{p}, \mathbf{q}]) \dots)$$



Method	Movielens		Pinterest		Result from
	HR@10	NDCG@10	HR@10	NDCG@10	
Popularity	0.4535	0.2543	0.2740	0.1409	[8]
SLIM [25, 30]	<u>0.7162</u>	<u>0.4468</u>	0.8679	<u>0.5601</u>	[8]
iALS [20]	0.7111	0.4383	0.8762	0.5590	[8]
NeuMF (MLP+GMF) [17]	0.7093	0.4349	<u>0.8777</u>	0.5576	[8]
Matrix Factorization	<b>0.7294</b>	<b>0.4523</b>	<b>0.8895</b>	<b>0.5794</b>	Fig. 2

# Conclusions



1. The fundamental difference between neural network models and factorization machines is that we do not limit ourselves only to linear transformations and add non-linearity
2. Two-tower architecture (in particular DSSM) has a number of advantages:
  - A. A large space for creativity in the design of features 🙌
  - B. Fast inference, since embedding and items can be pre-calculated offline
  - C. The ability to build an offline (and even online) selection of candidates for trained embedding
3. Teaching an alternative dot-y measure of proximity using MLP is a flexible idea

# Self-attention

---

**girafe**  
**ai**

**06**



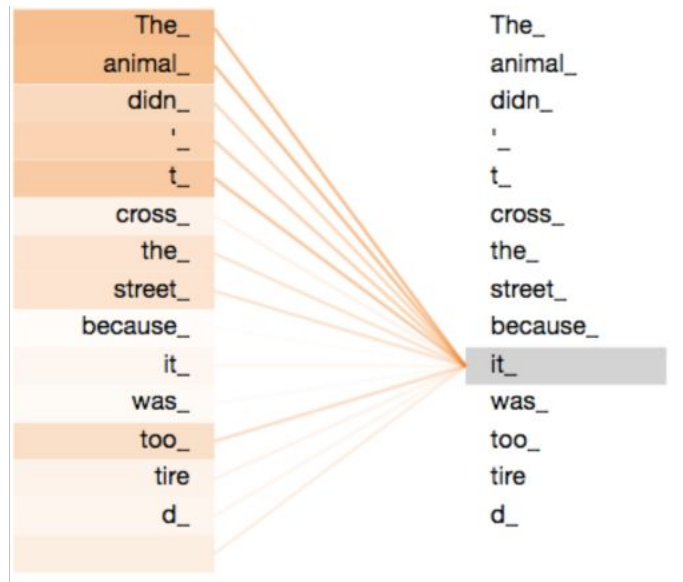
# Self-attention



The layer receives the embedding  $x_1, \dots, x_t$  of sequence elements as input and converts them into embeddings  $z_1, \dots, z_t$  that take into account information about the elements in all other positions.

Let's look at the illustration for the texts:

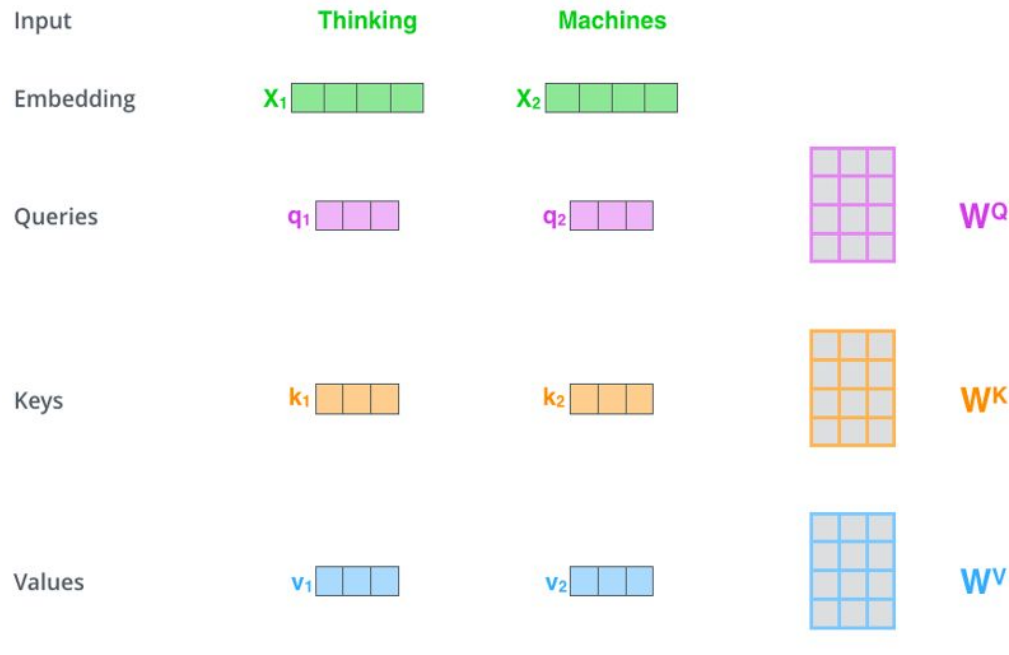
- In each position we look at all the words in the text
- We determine which words need to be looked at more strongly, and which ones are weaker
- Based on this information, we are building a new embedding position



# Self-attention



The first step is to build three vectors  
Key, Query and Value for each word  
in the sentence

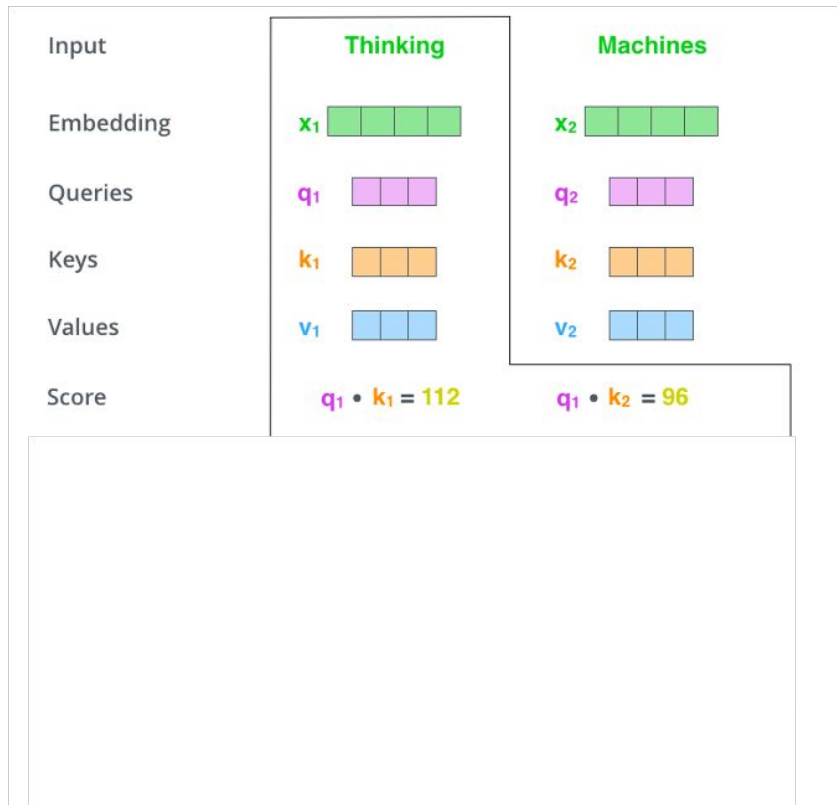


# Self-attention



The first step is to build three vectors  
Key, Query and Value for each word in  
the sentence

Further, the scalar product of the  
vectors  $q_i$  and  $k_i$  shows how much  
attention should be paid to the word  $j$  in  
position  $i$ .



# Self-attention



The first step is to build three vectors Key, Query and Value for each word in the sentence

Further, the scalar product of the vectors  $q_i$  and  $k_j$  shows how much attention should be paid to the word  $j$  in position  $i$ .

Then we normalize the scores by the root of the dimensions of the key and query vectors and calculate the softmax of the scores for each position.

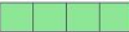
We get the weights with which we look at each word in the sentence.

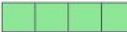
Input

Thinking

Machines

Embedding

$x_1$  

$x_2$  

Queries

$q_1$  

$q_2$  

Keys

$k_1$  

$k_2$  

Values

$v_1$  

$v_2$  

Score

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

Divide by 8 (  $\sqrt{d_k}$  )

14

12

Softmax

0.88

0.12

# Self-attention



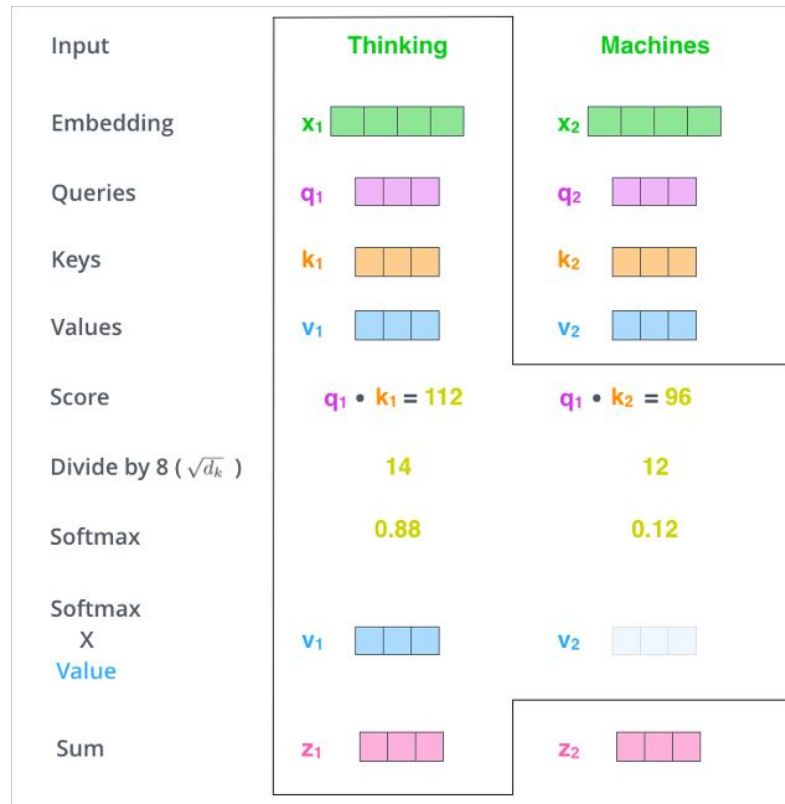
The first step is to build three vectors Key, Query and Value for each word in the sentence

Further, the scalar product of the vectors  $q_i$  and  $k_j$  shows how much attention should be paid to the word  $j$  in position  $i$ .

Then we normalize the scores by the root of the dimensions of the key and query vectors and calculate the softmax of the scores for each position.

We get the weights with which we look at each word in the sentence.

With these weights, we average the value vectors and get a new embedding of a word in a sentence that takes into account all other words





# Self-attention in matrix form

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$

$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

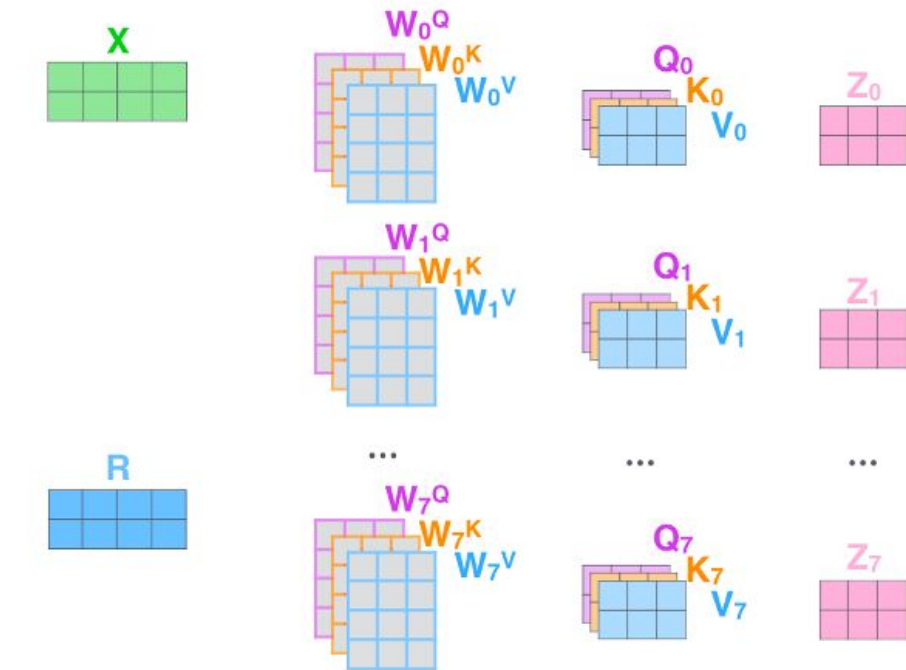
$$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V}$$
$$= \mathbf{Z}$$



# Multi-head Self-attention

We will use several parallel heads of self-attention.

For each head we have our own projection matrices



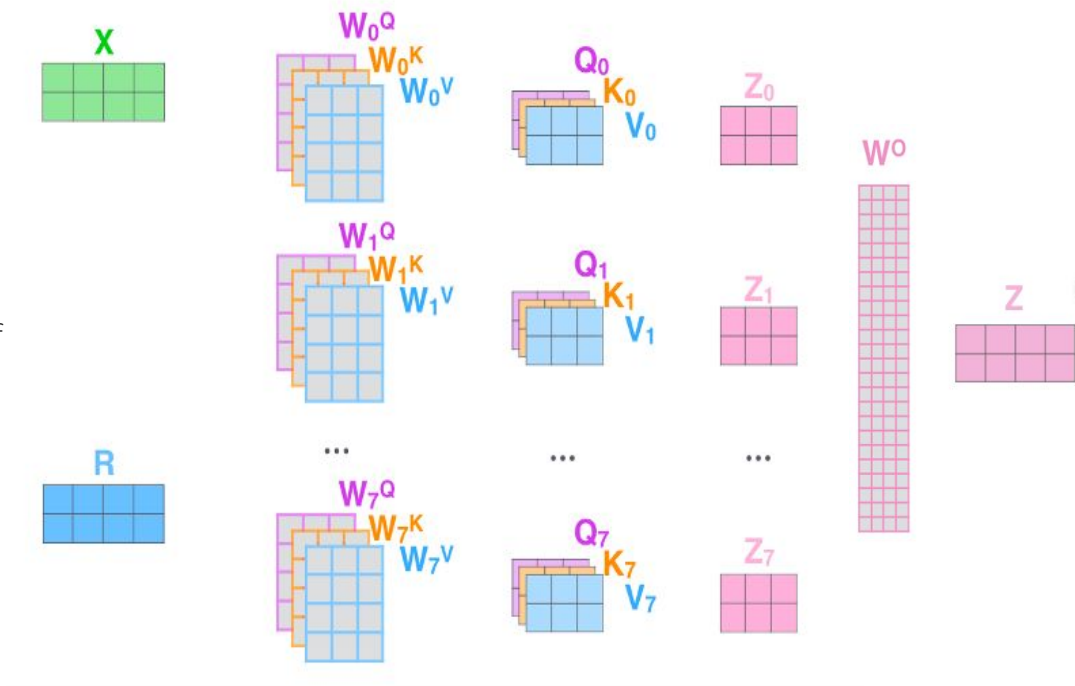


# Multi-head Self-attention

We will use several parallel heads of self-attention.

For each head we have our own projection matrices

At the end we concatenate the embedding and from all the heads and linearly project it into the embedding of the correct size

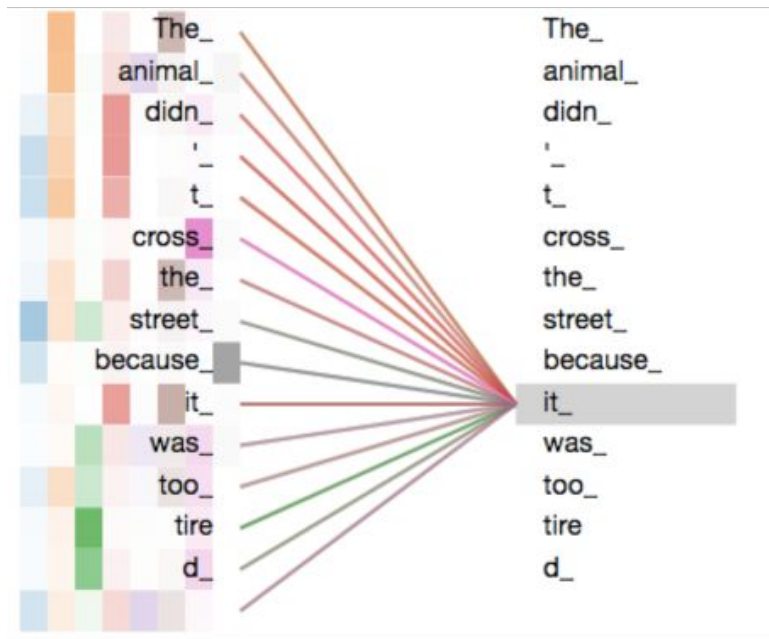






# Multi-head Self-attention

The main idea: different heads pay attention to the different dependencies between the elements of the sequence



# Positional Encoding

---

**girafe**  
**ai**

**07**

# Positional Encoding



Problem: in this formula there is no dependence on the position in the sentence, we work with its elements as with a set

# Positional Encoding



Problem: in this formula there is no dependence on the position in the sentence, we work with its elements as with a set

Solution: to embedding positional embedding and, instead of  $x_i$ , consider  $x_i + p_i$ , where positional embedding  $p_i$  depends only on position  $i$ .

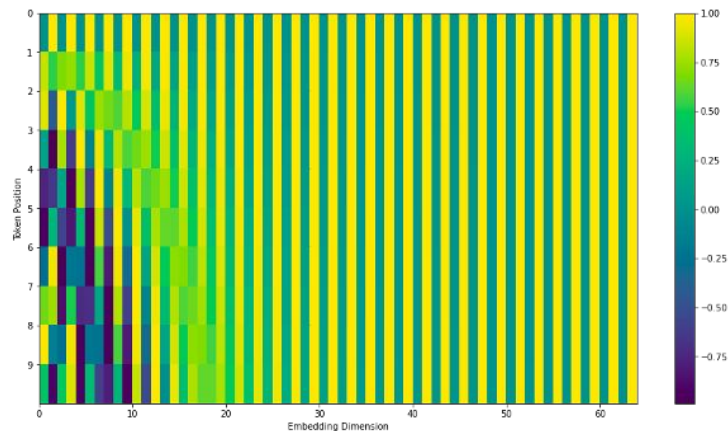


# Positional Encoding

Problem: in this formula there is no dependence on the position in the sentence, we work with its elements as with a set

Solution: to embedding positional embedding and, instead of  $\mathbf{x}_i$ , consider  $\mathbf{x}_i + \mathbf{p}_i$ , where positional embedding  $\mathbf{p}_i$  depends only on position  $i$ .

Option 1: positional embedding is deterministically calculated by cosine formulas





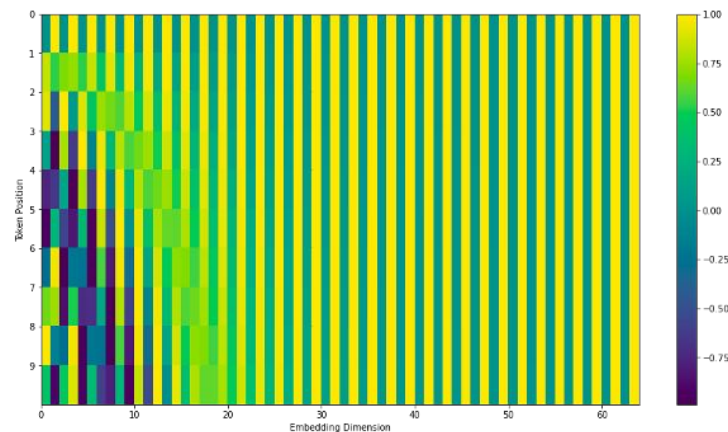
# Positional Encoding

Problem: in this formula there is no dependence on the position in the sentence, we work with its elements as with a set

Solution: to embedding positional embedding and, instead of  $\mathbf{x}_i$ , consider  $\mathbf{x}_i + \mathbf{p}_i$ , where positional embedding  $\mathbf{p}_i$  depends only on position  $i$ .

Option 1: positional embedding is deterministically calculated by cosine formulas

Option 2: Positional embedding is trained together with the entire model



# Bert4Rec

---

**girafe**  
**ai**

# 08

# Bert4Rec



- The self-attention approach shows itself well in tasks related to texts, audio and images
- The Bert4Rec model is an attempt to apply this approach to the task of recommendations
- The model is applied to a sequence of items from the user's history and predicts the next item





# Multi-head Self-attention

The layer receives the embedding  $h_1, \dots, h_t$  of sequence elements as input and converts them into embeddings  $h'_1, \dots, h'_t$  that take into account information about all other positions.

Let's define the basic Attention operation, which takes the input of the matrix  $Q, K, V$  and returns

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{m}} \right) V, \quad Q, K, V \in \mathbb{R}^{t \times m}$$

- $\text{Attention}(Q, K, V)_i = \text{softmax} \left( \frac{Q_i K^T}{\sqrt{m}} \right) V$

This is a linear combination of strings from  $V$  with weights  $\text{softmax} \left( \frac{Q_i K_1}{\sqrt{m}} \right), \dots, \text{softmax} \left( \frac{Q_i K_t}{\sqrt{m}} \right)$

- Accordingly,  $Q_i$  and  $K$  express how much contribution information about all other elements should have to the element of position  $i$ .
- The matrix  $V$  carries the information that each position should transmit.



# Multi-Head Self-Attention

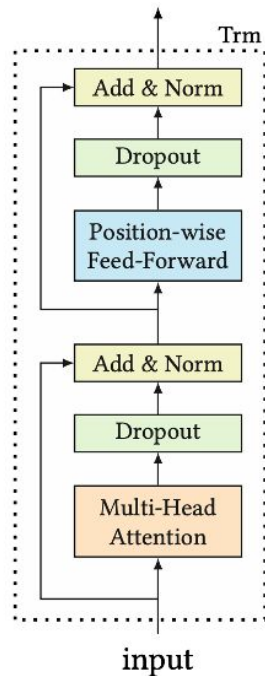
Let the input embedding form a  $H \in \mathbb{R}^{t \times d}$   
matrix  
Define

$$MH(H) = [\text{head}_1, \dots, \text{head}_h] W^O, \quad W^O \in \mathbb{R}^{d \times d}$$

$$\text{head}_i = \text{Attention}(HW_i^Q, HW_i^K, HW_i^V), \quad W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d/h}$$

$$W^O, W_i^Q, W_i^K, W_i^V \quad \text{-- trainable parameters}$$

# Transformers layer



(a) Transformer Layer.

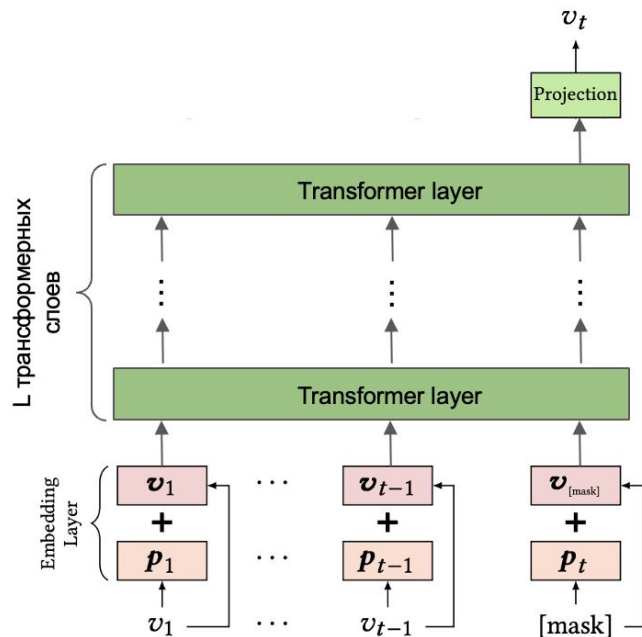
Consists of:

- Multi-Head Self-Attention Layer
- Position-wise Feed-Forward, applied to outputs piecemeal
- Dropout, skip connection and LayerNorm after both previous ones

$$FFN(x) = GELU(xW^{(1)} + b^{(1)})W^{(2)} + b^{(2)}$$



# Bert4Rec Architecture



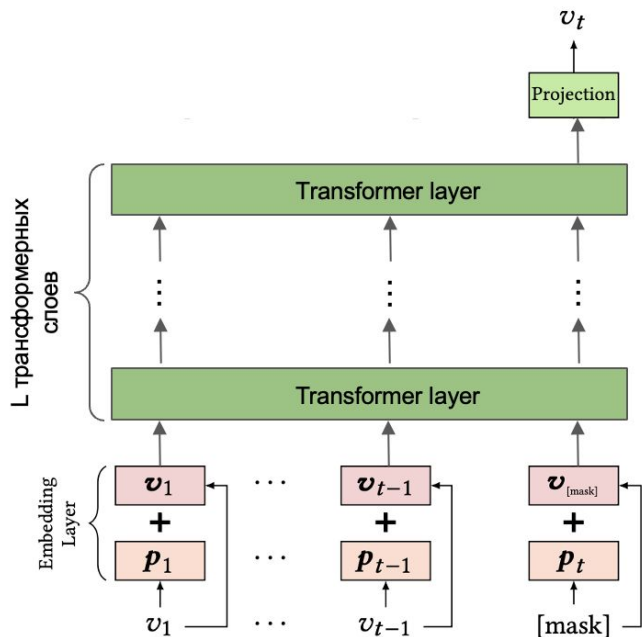
The model consists of

- Layers of embeddings of aitemes
- $L$  Transformer layers
- A projecting head that performs predictions

(b) BERT4Rec model architecture.



# Bert4Rec Architecture



(b) BERT4Rec model architecture.

The model consists of

- Layers of embeddings of aitems
- $L$  Transformer layers
- A projecting head that performs predictions

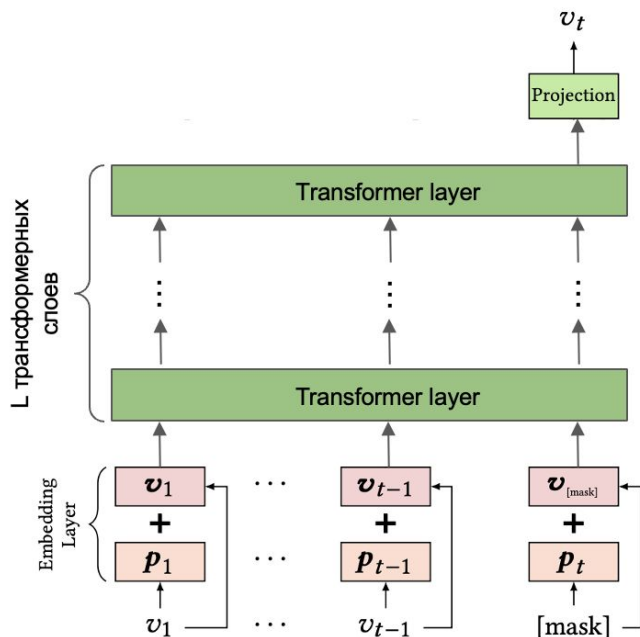
Items of positive user interactions are submitted to the input in order of time.

A special [mask] item is added to the end of the story.

To the output embedding corresponding to this special item, we apply a projection head that predicts the relevant next item.



# Bert4Rec Architecture



(b) BERT4Rec model architecture.

The model consists of

- Layers of embeddings of aitems
- $L$  Transformer layers
- A projecting head that performs predictions

Items of positive user interactions are submitted to the input in order of time.

A special [mask] item is added to the end of the story.

To the output embedding corresponding to this special item, we apply a projection head that predicts the relevant next item.

The projection layer is arranged as

$$P(v) = \text{softmax} \left( \text{GELU}(h_i^L W^P + b^P) E^T + b^O \right),$$

where  $E \in \mathbb{R}^{|V| \times d}$  the trainable matrix of aitems



# Bert4Rec Training

- We will randomly mask the share of the items  $\rho$  from the user's history, that is, we will replace it with a special item [mask]

**Input:**  $[v_1, v_2, v_3, v_4, v_5] \xrightarrow{\text{randomly mask}} [v_1, [\text{mask}]_1, v_3, [\text{mask}]_2, v_5]$

**Labels:**  $[\text{mask}]_1 = v_2, \quad [\text{mask}]_2 = v_4$

- Let  $S_u^m$  be the set of masked positions of the user  $u$ ,

$S'_u$  – a sequence of aitems with replaced by [mask] aitems,

$v_m^*$  – predictions of aitems in disguised positions

Then we optimize the likelihood, that is:

$$\mathcal{L} = -\frac{1}{|S_u^m|} \sum_{m \in S_u^m} \log P(v_m^* = v_m | S'_u)$$

- In other words, learning is similar to Masked Language Model learning in text tasks

# Thanks for attention!

Questions?



**girafe**  
**ai**

