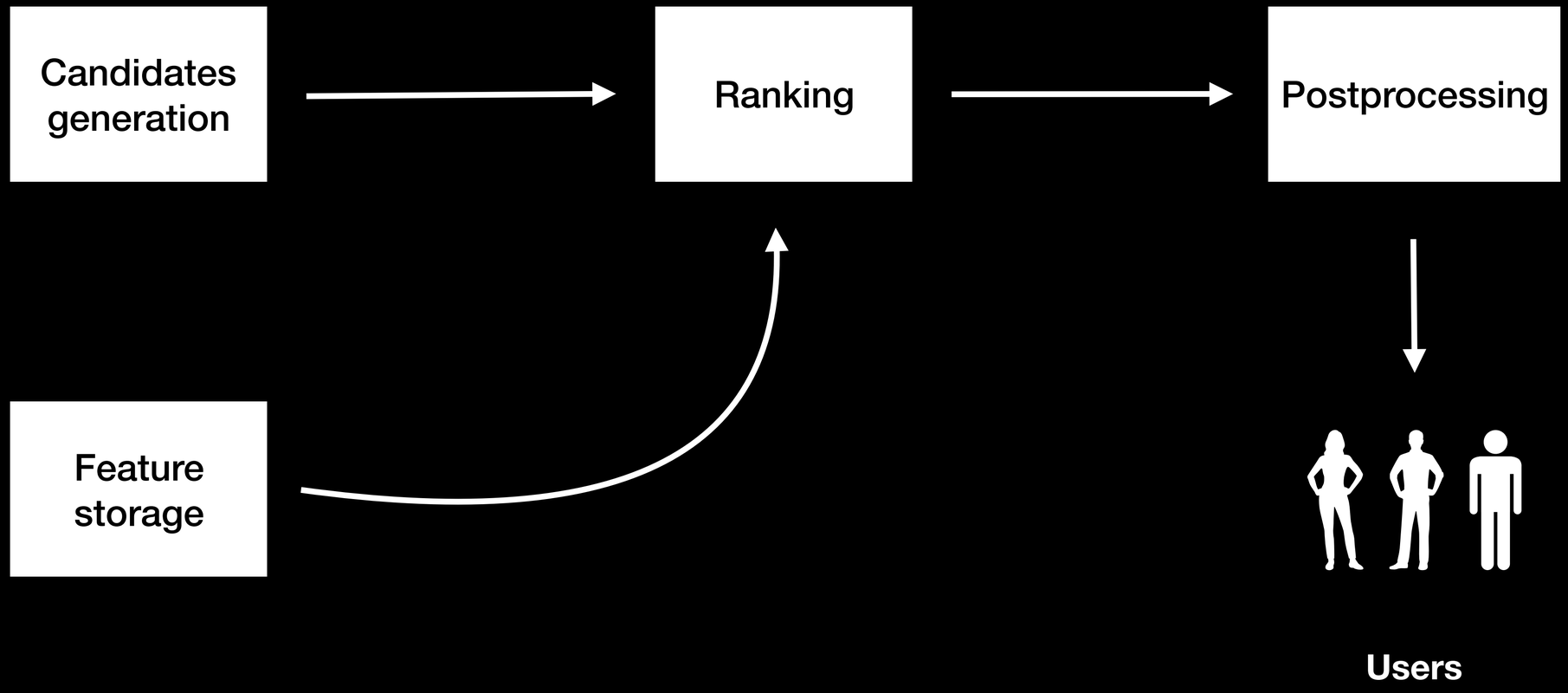
Recommendation Systems

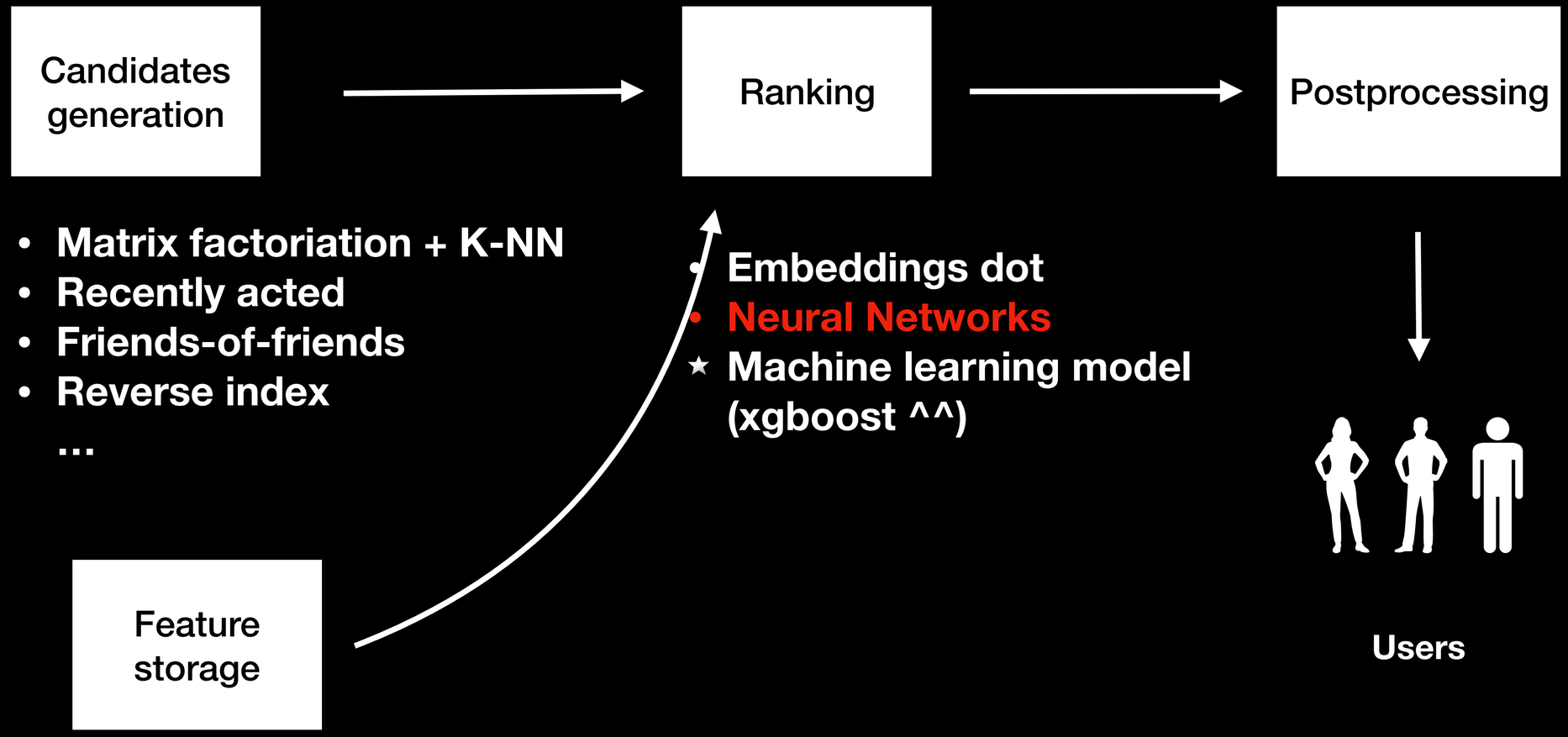
Deep Learning, part 2

Eugeny Malyutin / Sergey Dudorov

Recommendations pipeline:



Recommendations pipeline:

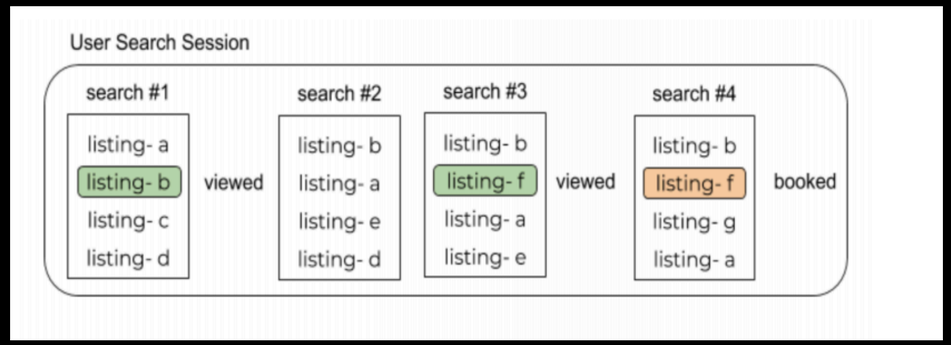


Neural networks as ranking:

- Task: airbnb search.

  Stop, what? We are about recsys. Are you **mad?** Nope. Have you seen airbnb search? It's totally about ranking and search is only «name» , no normal textual questions, all query is built from simple geo and price ranges so yes, it's **more recommender system then search.** They use ranking objectives, measures NDCG and got strong correlations between NDCG- and bookings- gains (offline-online correlation) that's why it's more recommendation then search

- Dataset: booked vs viewed.
  Offline quality: NDCG.
  Baseline: GBDT.
  Online metric: bookings.



Source: Applying Deep Learning To Airbnb Search

Ch1. Do not be a heroes:

Custom super-designed architecture.

The same GBDT features.

L2-loss with 1 as positives, 0 as negatives.

Awful results.

WHY?

Source: Applying Deep Learning To Airbnb Search

Ch1. Do not be a heroes:

Simple NN with 32-units hidden layer.

The same GBDT features.

L2-loss with 1 as positives, 0 as negatives.

Awful results.

WHY?

Source: Applying Deep Learning To Airbnb Search

Ch1. Do not be a heroes:

Simple NN with 32-units hidden layer. The same GBDT features. L2-loss with 1 as positives, 0 as negatives. Awful results.

- Feature normalisation:
  - Normal distribution: $(f\_val - \mu)/\sigma$
  - Power law distr: $log(\frac{1 + f\_val}{1 + median})$

Source: Applying Deep Learning To Airbnb Search

Ch1. Do not be a heroes:

Simple NN with 32-units hidden layer. The same GBDT features. L2-loss with 1 as positives, 0 as negatives. Awful results.

- Smooth distributions:
  - Spotting bugs
  - Facilitating generalization

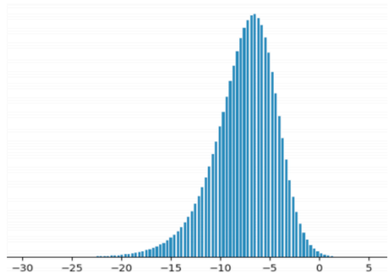


Figure 8: Distribution of output layer.

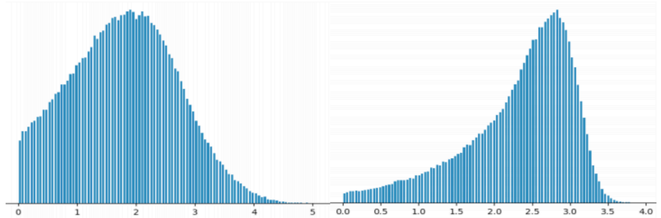


Figure 9: Example distributions from second hidden layer.

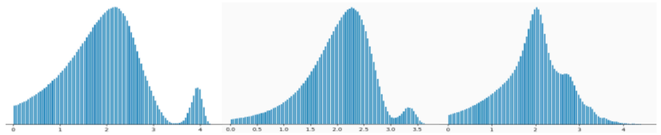


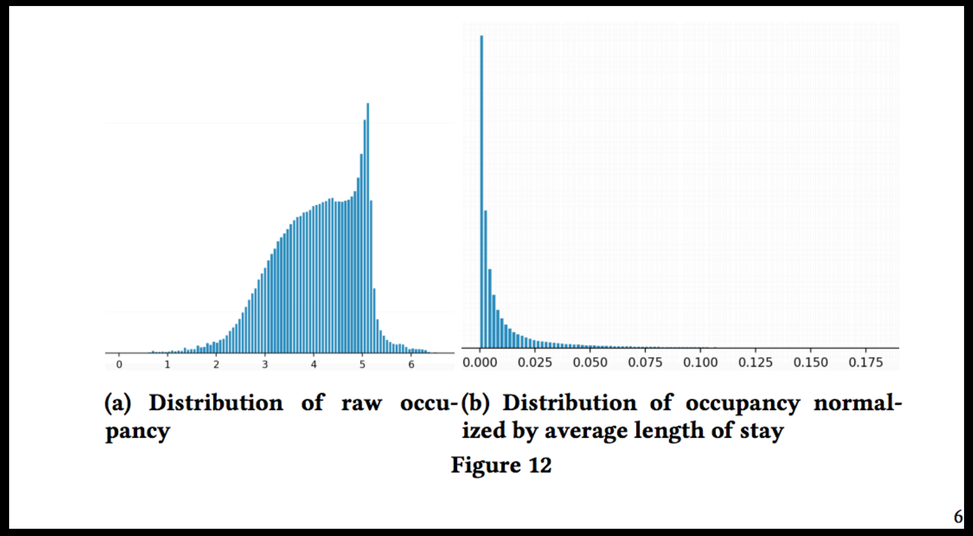
Figure 10: Example distributions from first hidden layer.

Source: Applying Deep Learning To Airbnb Search

Ch1. Do not be a heroes:

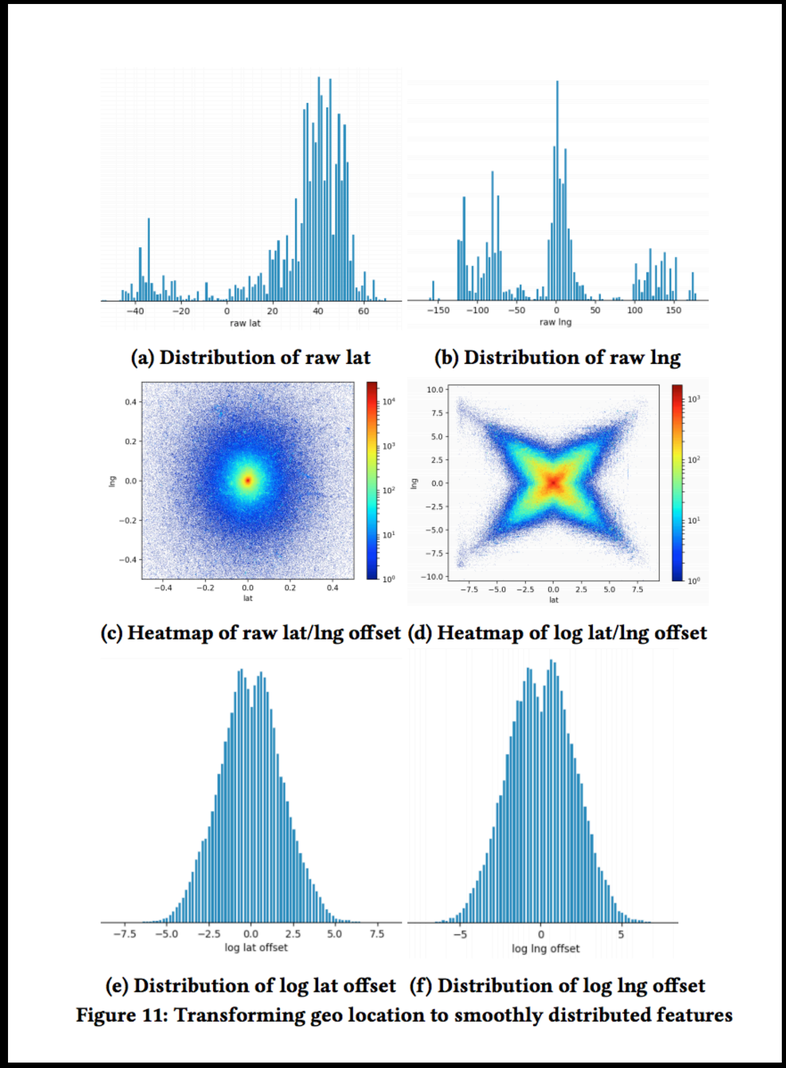
Simple NN with 32-units hidden layer. The same GBDT features. L2-loss with 1 as positives, 0 as negatives. Awful results.

- Special features mapping. Lat/lng to offset wrt users screen centered.
- Completness of features: occupancy



(a) Distribution of raw occupancy (b) Distribution of occupancy normalized by average length of stay

Figure 12



(a) Distribution of raw lat (b) Distribution of raw lng

(c) Heatmap of raw lat/lng offset (d) Heatmap of log lat/lng offset

(e) Distribution of log lat offset (f) Distribution of log lng offset

Figure 11: Transforming geo location to smoothly distributed features

Ch1. Do not be a heroes:

Simple NN with 32-units hidden layer.

Features are normalized and validated.

L2-loss with 1 as positives, 0 as negatives.

Normal results.

Source: Applying Deep Learning To Airbnb Search

Ch2. Main character runs up a stairway:

LambdaRank in NN:

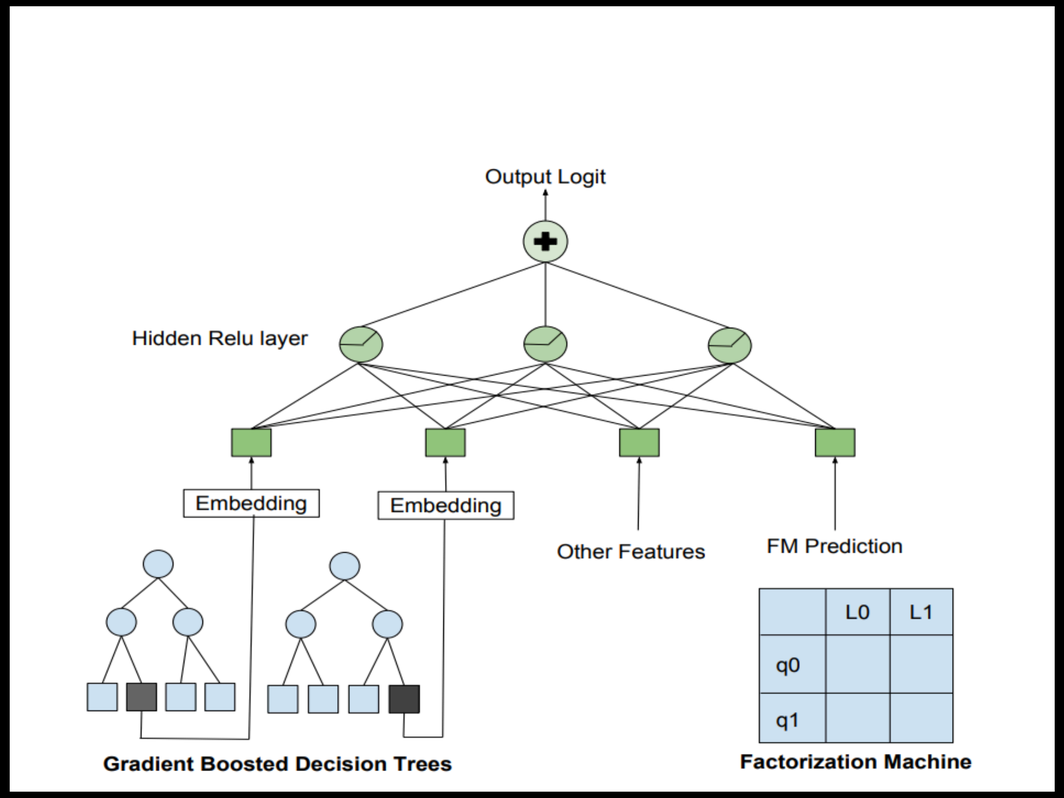
- Pairwise dataset. {positive_list, negative_list} sampled from the same session logs. CE loss between listed and not-listed
- Scaled at NDCG change of swapping pairs.
- ^^ Prioritizes 1 to 0 ranking over 125 to 124.

Ch2. Main character runs up a stairway:

GBDT-NN-shtein:

- GBDT up-ranks different samples.
- FM-machines with their embeddings.
- NN with pretty smooth generalisation and/or interactions

Frankenshtein to combine them all ->



Ch2. Main character runs up a stairway:

Frankenshtein to combine them all -> was overperformed by this network and 10x data.

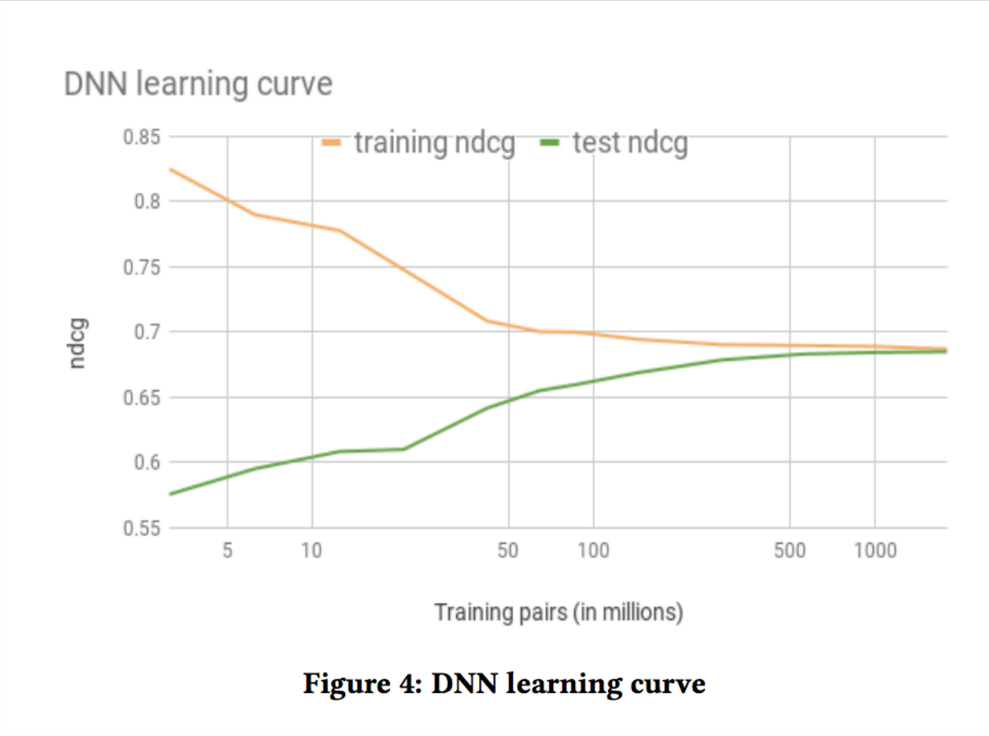
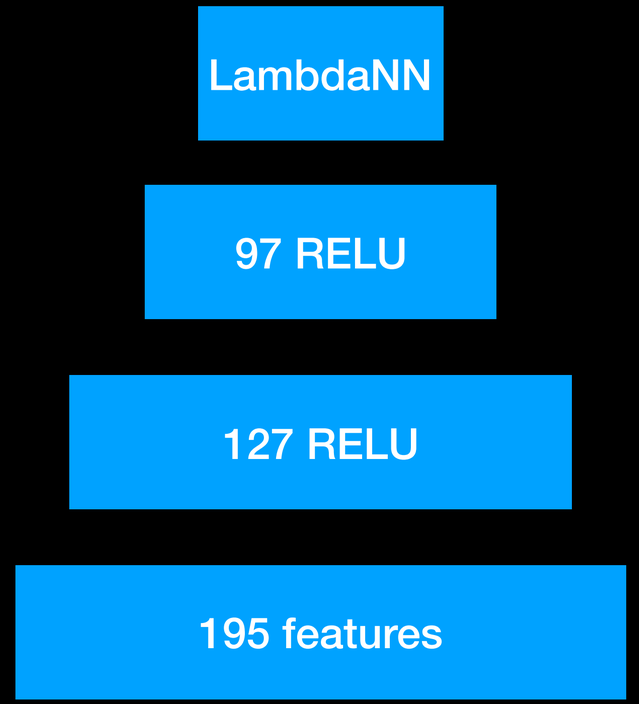


Figure 4: DNN learning curve



Ch2. Main character runs up a stairway:

- «General» embeddings (but location preferences)

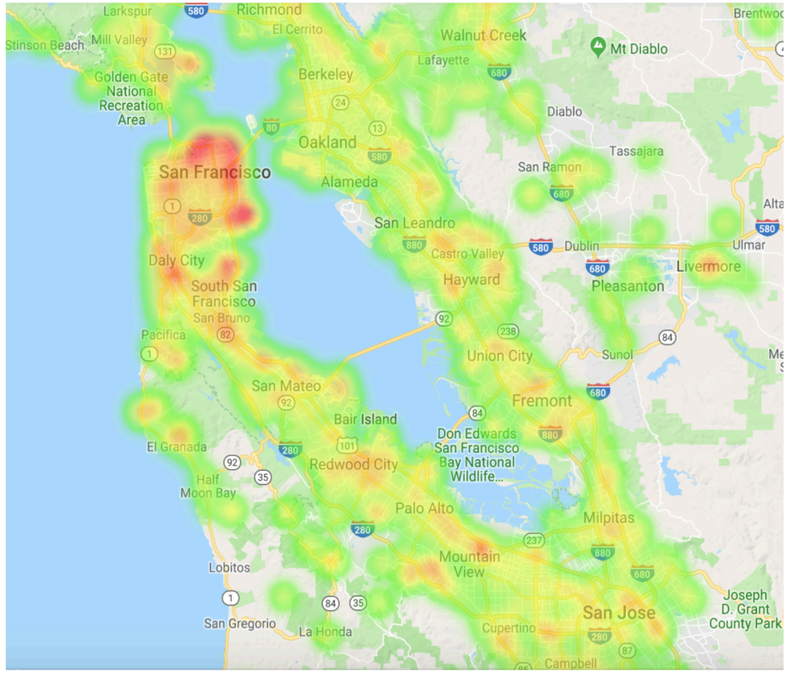


Figure 13: Location preference learnt for the query ”San Francisco”

- {”San Francisco”, 539058204} → 71829521

Ch3. Hero recall failed brothers:

- MOAR layers
- «Modern attention architectures»
- Direct entities embeddings
- …

Vol.2 Hero strikes back

Fig. 1. X-axis shows how the price of the booked listing offsets from the median price of search results for a guest. Y-axis is the number of users corresponding to a price offset.

Source: Improving Deep Learning For Airbnb Search

Ch1: Naive

$$DNN_\theta(u, q, l_{noPrice}) + (-tanh(w * P + b)$$

$$P = log(\frac{1 + price}{1 + price_{median}})$$

- -5.7% avg price in booking, -1.5% of bookings

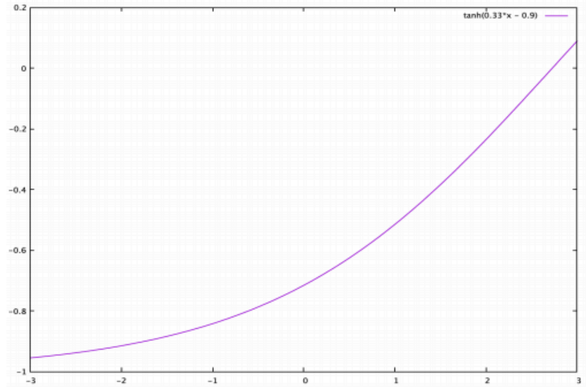


Fig. 2. X-axis is normalized price feature. Y-axis is the value of the $tanh$ term in Equation 1.

Ch1: Less naive

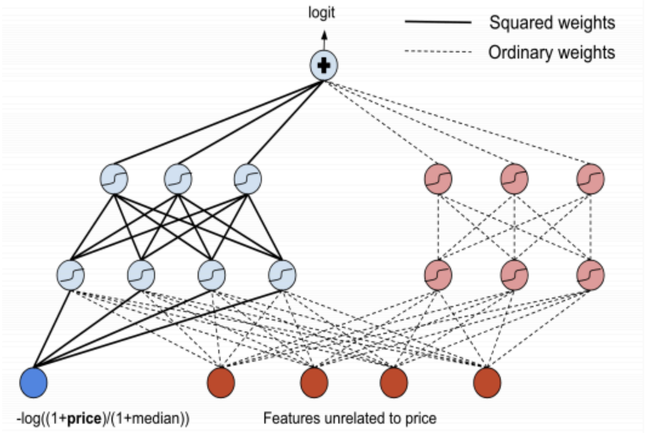


Fig. 3. DNN architecture partially monotonic w.r.t price. Bold solid lines indicate weights that are squared, dashed lines indicate ordinary weights.

Ch1: Less naive

- **-1.4% of bookings**
- Tried even additional loss component - nothing.

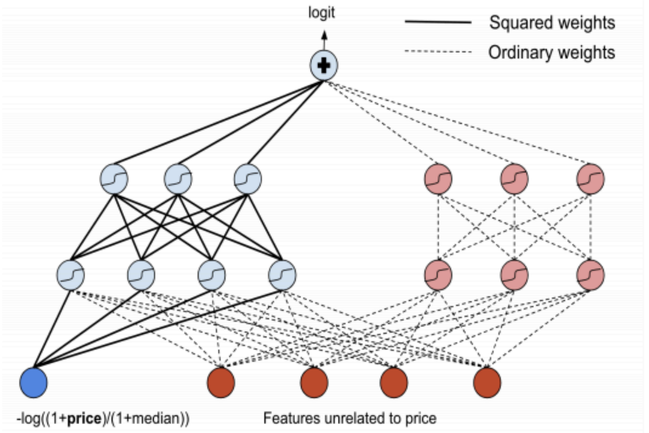


Fig. 3. DNN architecture partially monotonic w.r.t price. Bold solid lines indicate weights that are squared, dashed lines indicate ordinary weights.

Ch2: I - interpretability

- Individual conditional expectation (ICE) plots

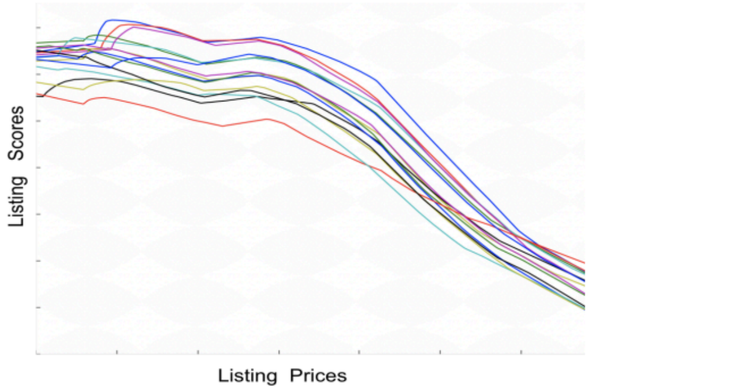


Fig. 4. ICE plot of listing scores (y-axis) vs listing prices (x-axis). Each curve represents a listing in the search result.

Ch2: I - interpretability

- Individual conditional expectation (ICE) plots

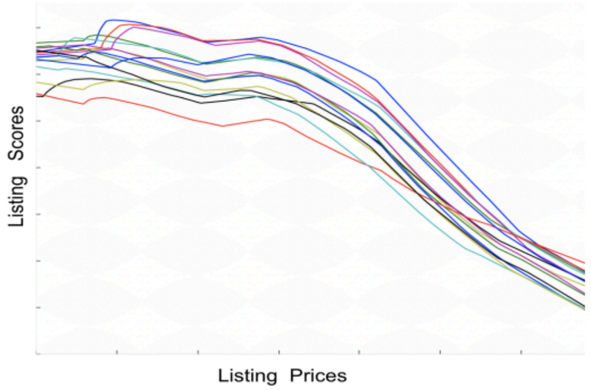


Fig. 4. ICE plot of listing scores (y-axis) vs listing prices (x-axis). Each curve represents a listing in the search result.

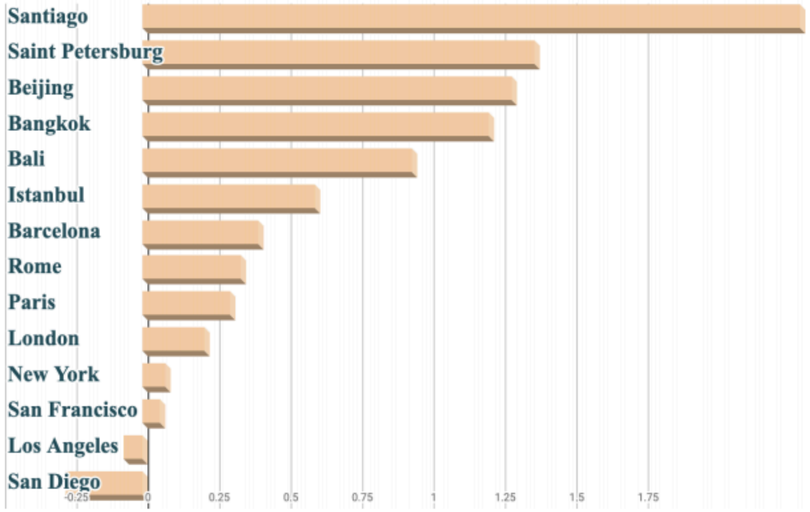


Fig. 5. Average difference between the median price of search results for a guest and the price of the booked listing, split by cities

Ch2: I - interpretability

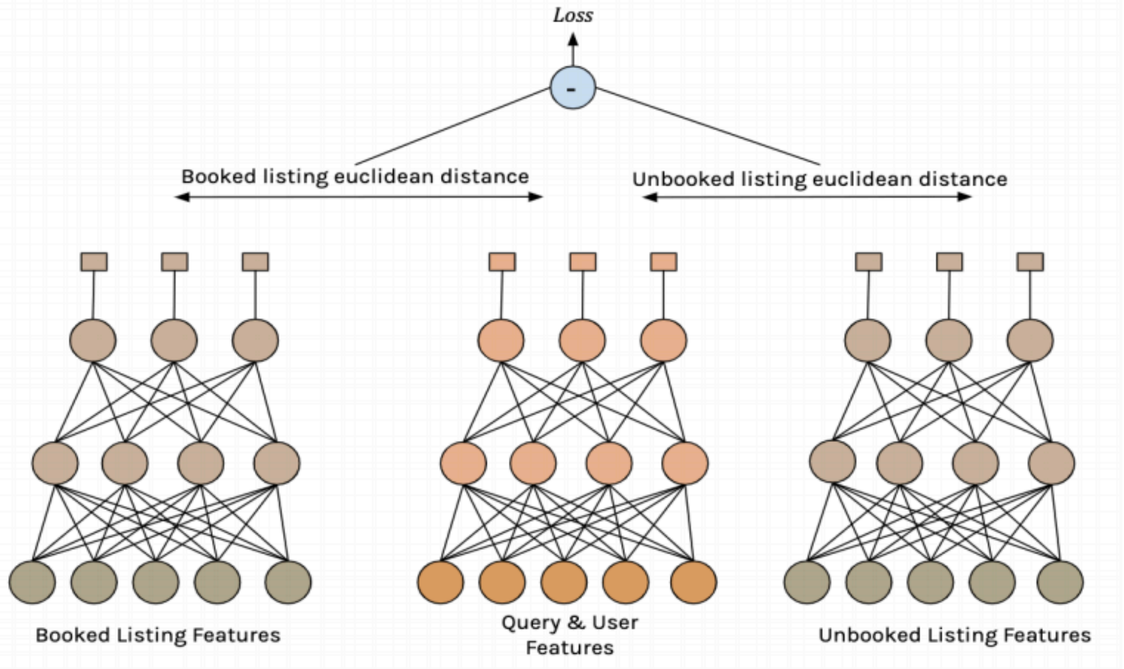


Fig. 6. Pairwise loss computation for training the two tower architecture.

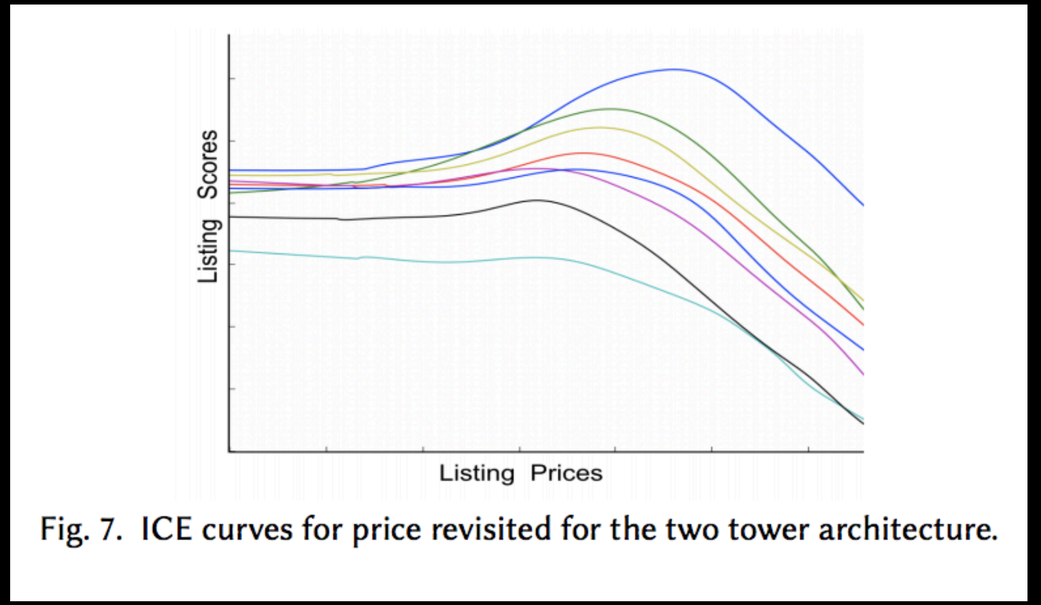


Fig. 7. ICE curves for price revisited for the two tower architecture.

- **-2.3% of bookings, +0.75% revenue**
- -33% peak latency(!)

Ch2: I - interpretability

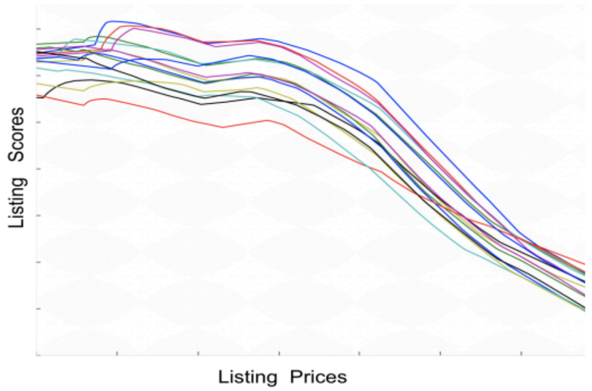


Fig. 4. ICE plot of listing scores (y-axis) vs listing prices (x-axis). Each curve represents a listing in the search result.

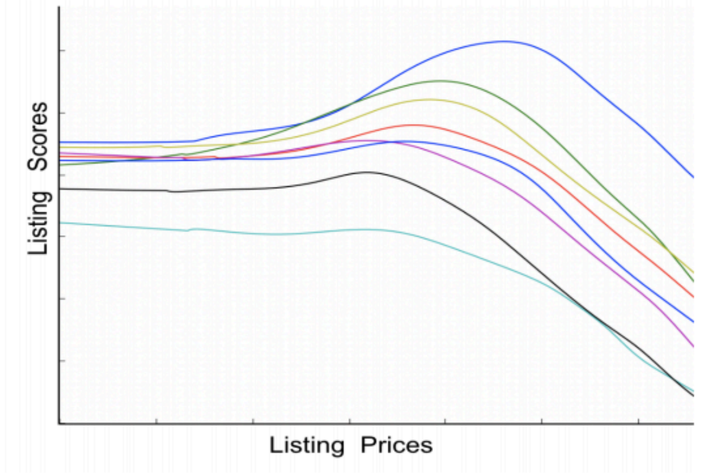


Fig. 7. ICE curves for price revisited for the two tower architecture.

Ch3: Lessons learned

- Users lead - model follows.
- Modern architectures highly tied with applications
- Architecture engineering, not complifications.
- Read moar:
  - Cold start
  - Positional bias
  - Improving Deep Learning For Airbnb Search (c)

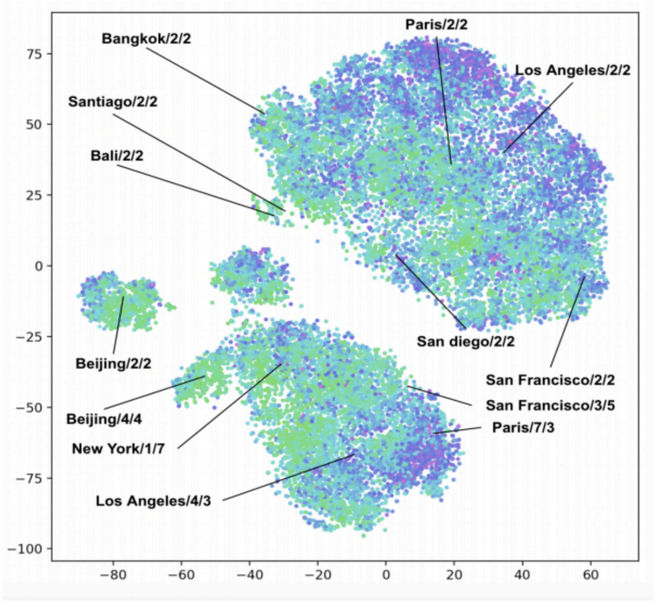


Fig. 8. t-SNE plot for output of query tower mapped to 2-d. Each dot represents a query. Some of the queries are labeled in the *city/guest-count/trip-length* format. Color of the dots indicate price of the listing booked for the query, cheaper towards green, expensive towards blue.

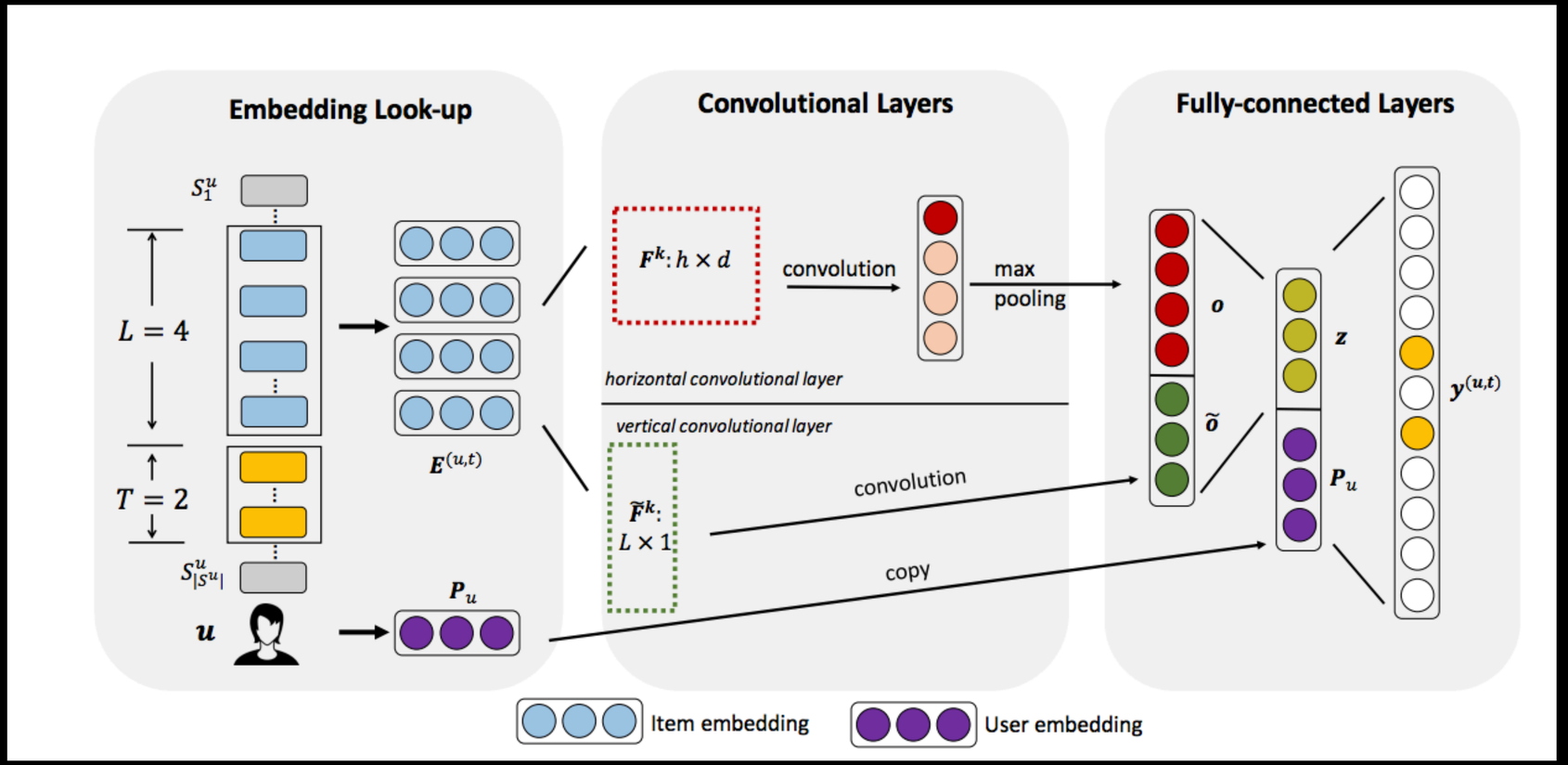
Modern architectures are like:

RecSys

The rest of DL

Caser:

- L — history size
- T — num of steps to predict
- S_t^u — user interacted item at time
- P_u — user embedding



Source:Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding

Caser:

- Horizontal convolution - to capture «semantic" pattern. Capture union-level patterns with multiple union sizes (c)
- These vertical Iters are capturing point-level sequential patterns through weighted sums over previous items’ latent representations. (c)
- Binary cross-entropy loss for next (T(!)) items to allow «skip-behaviour».

What’s went wrong?

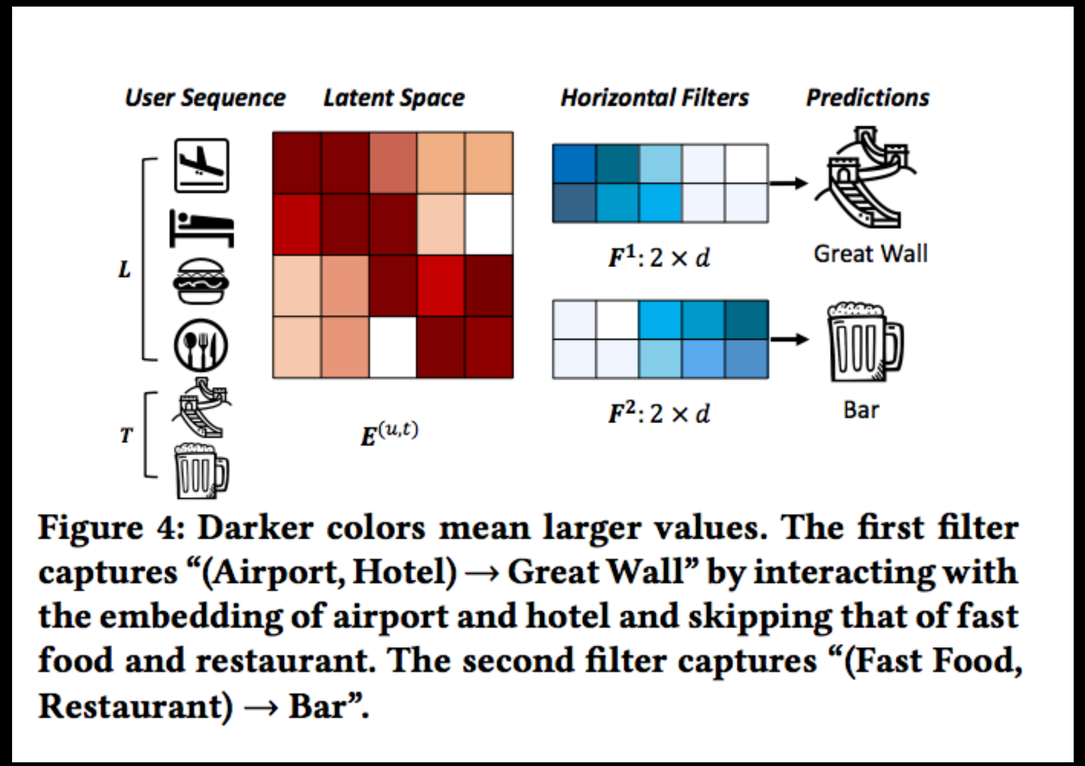


Figure 4: Darker colors mean larger values. The first filter captures “(Airport, Hotel) → Great Wall” by interacting with the embedding of airport and hotel and skipping that of fast food and restaurant. The second filter captures “(Fast Food, Restaurant) → Bar”.

Caser:

- Horizontal convolution - to capture «semantic" pattern. Capture union-level patterns with multiple union sizes (c)
- These vertical lters are capturing point-level sequential patterns through weighted sums over previous items’ latent representations. (c)
- Binary cross-entropy loss for next (T(!)) items to allow «skip-behaviour».

  What’s went wrong?

- Lack of capacity limited to size of a filters.
- Factually E(u,t) matrix has no geo-spatial local feature interactions. Learning such embeddings factually trickier then classic one.
- Binary CE…:
  - How to add new items?
  - How to add new users?
  - How can we re-evaluate network with new user activity?

YT(youtube) recommender:

- Loss: $P(w_t = i \mid U, C) = \frac{e^{v_i u}}{\sum_{j \in J} e^{v_j u}}$

  Problems?

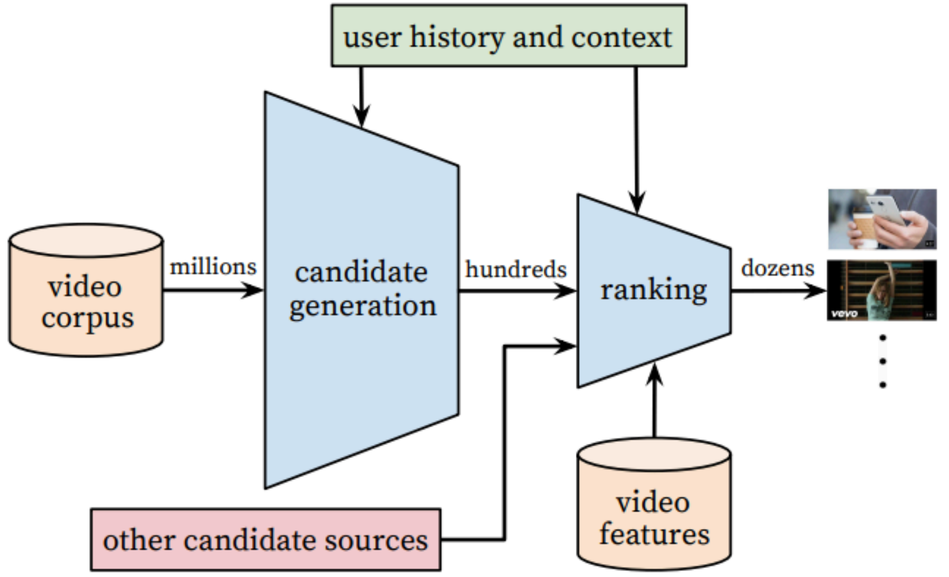
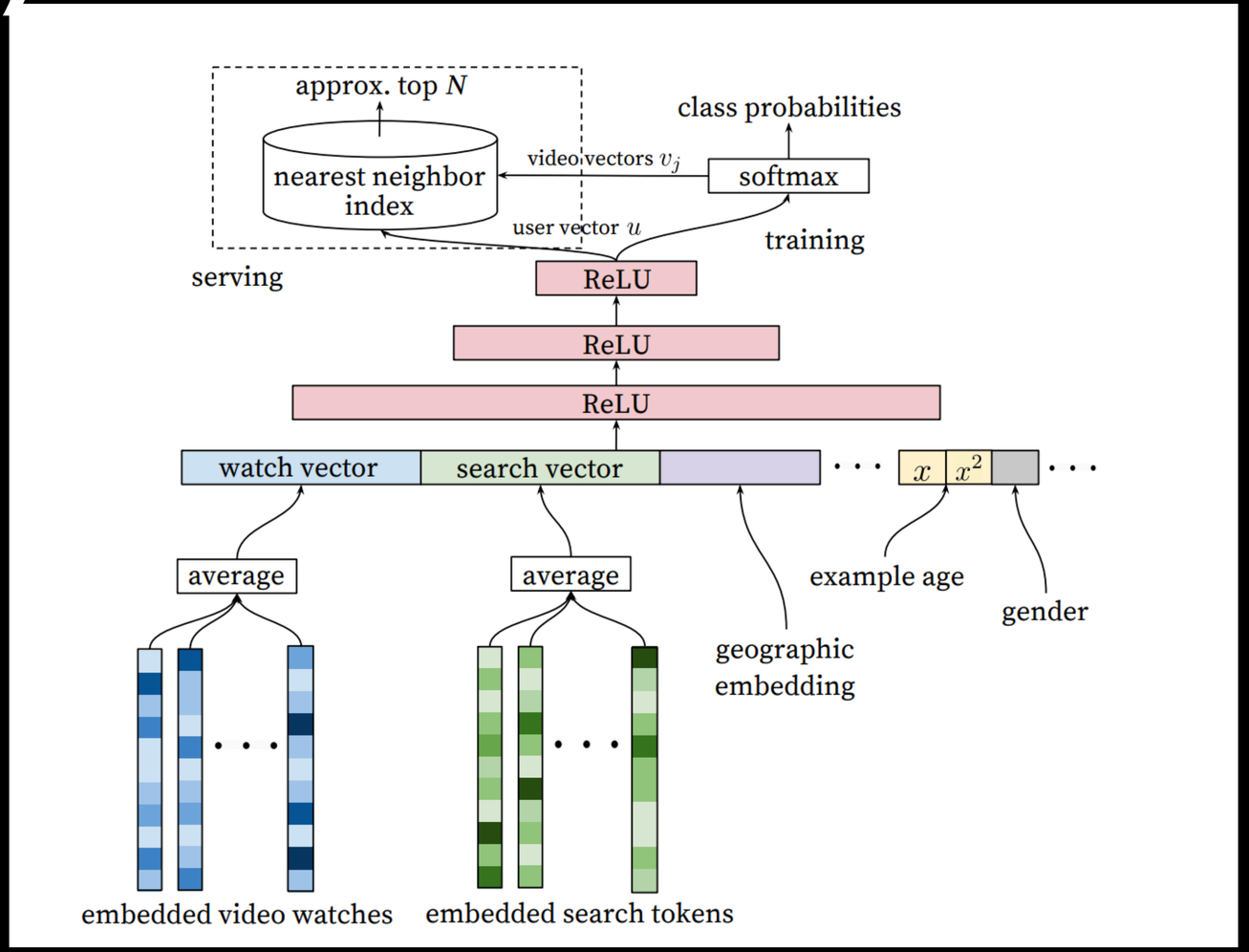


Figure 2: Recommendation system architecture demonstrating the "funnel" where candidate videos are retrieved and ranked before presenting only a few to the user.

Source:

YT(youtube) recommender:



- Loss:

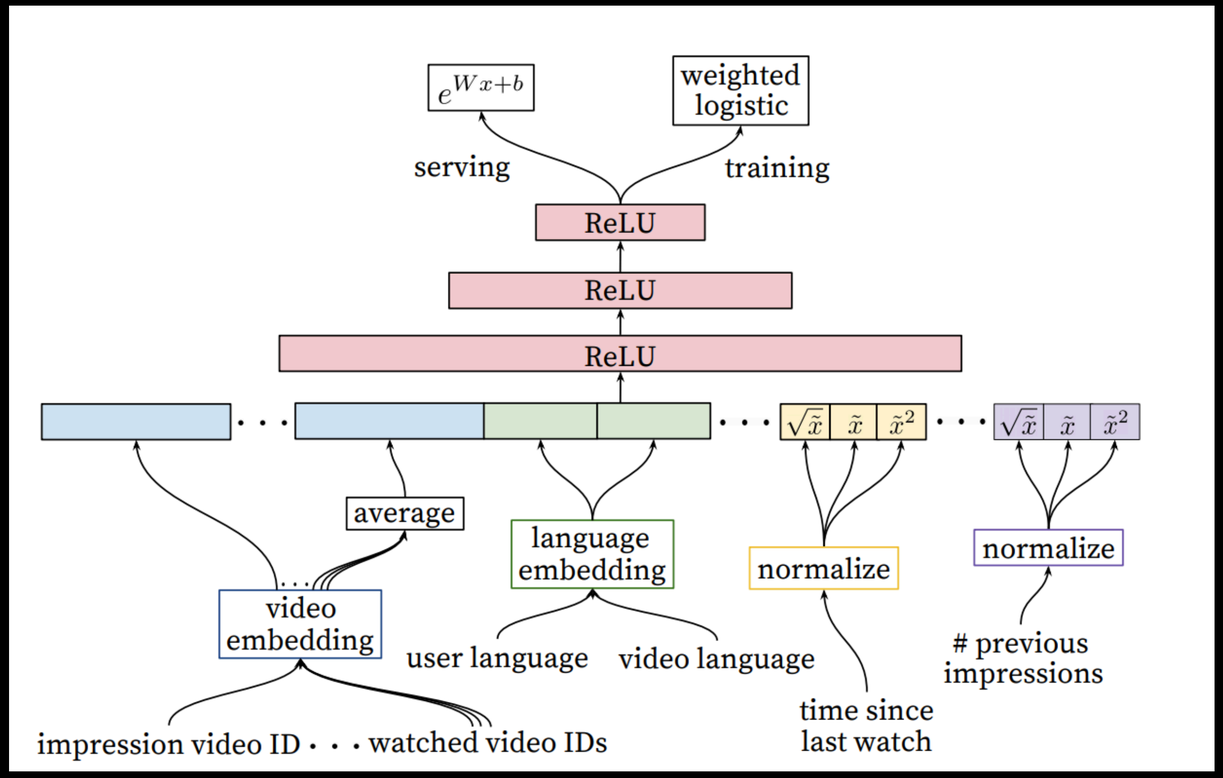
$$P(w_t = i \mid U, C) = \frac{e^{v_i u}}{\sum_{j \in J} e^{v_j u}}$$

Yep, neg sampling

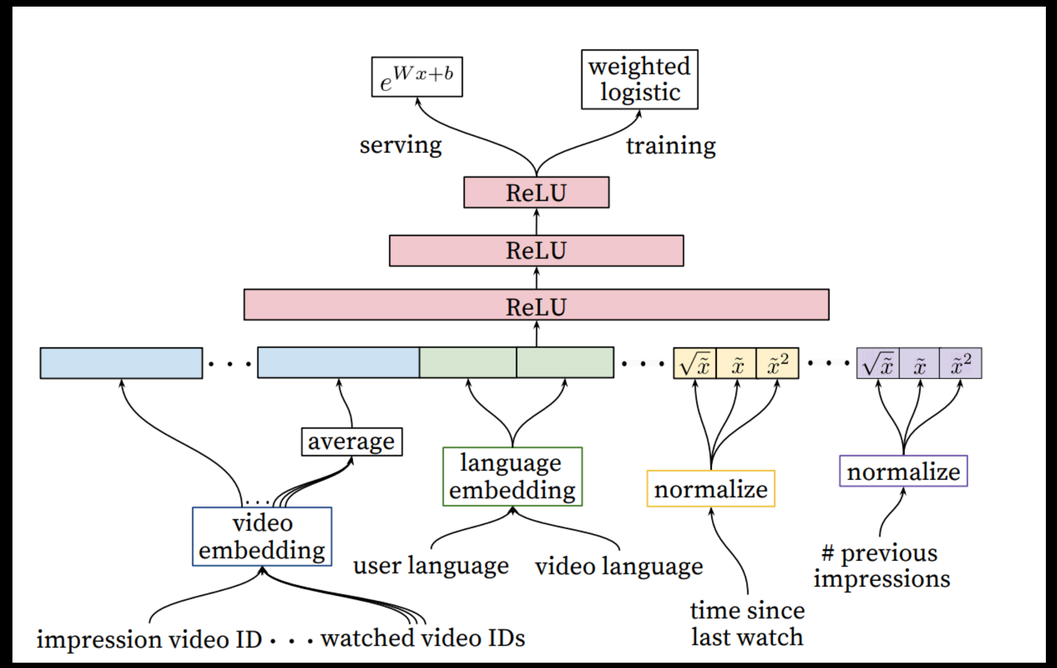
- Better with additional sources then recommended
- ANN in serve time

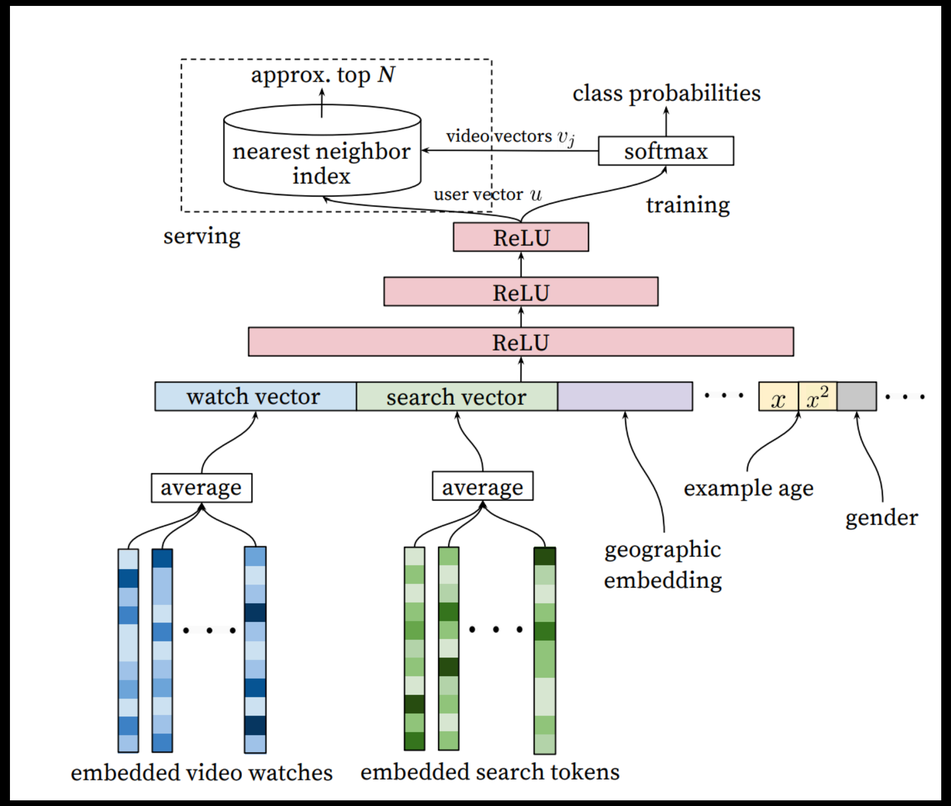
YT(youtube) recommender:

- Tips and tricks:
  - Continuous normalisation
  - Shared embedding (last video ID, current video ID, searched video ID)
  - Weighted log regr. For watch time:
    Positive weighted to T_i watch time
    Negatives unit
    odds ~ $\frac{\sum T_i}{N-k}$,
    learned odds ~ $E[T](1+P)$
    (P is small)



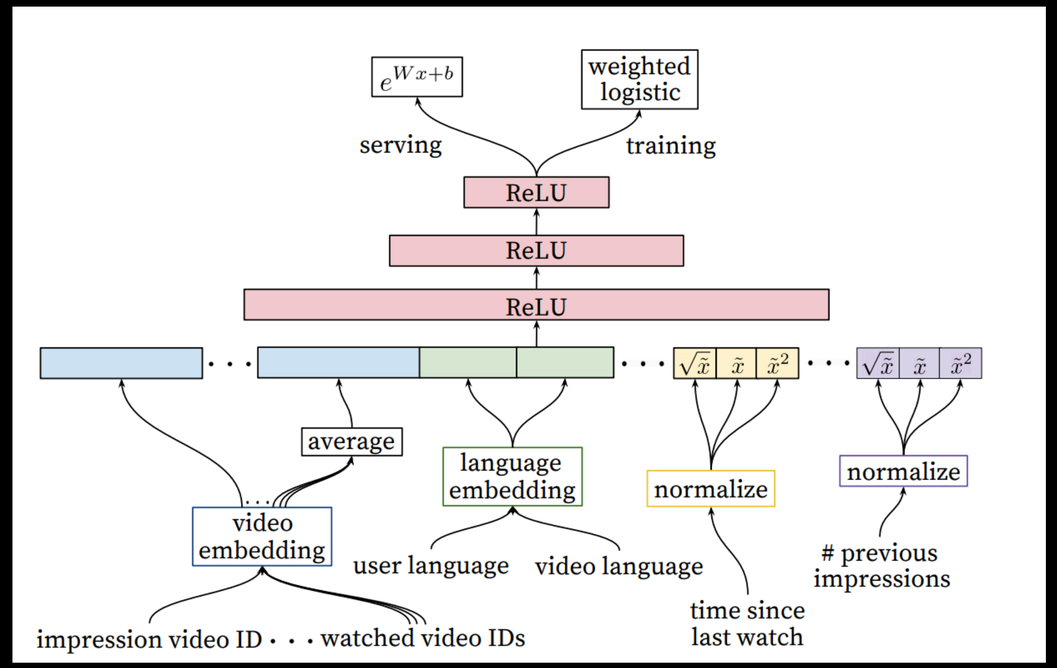
YT(youtube) recommender:

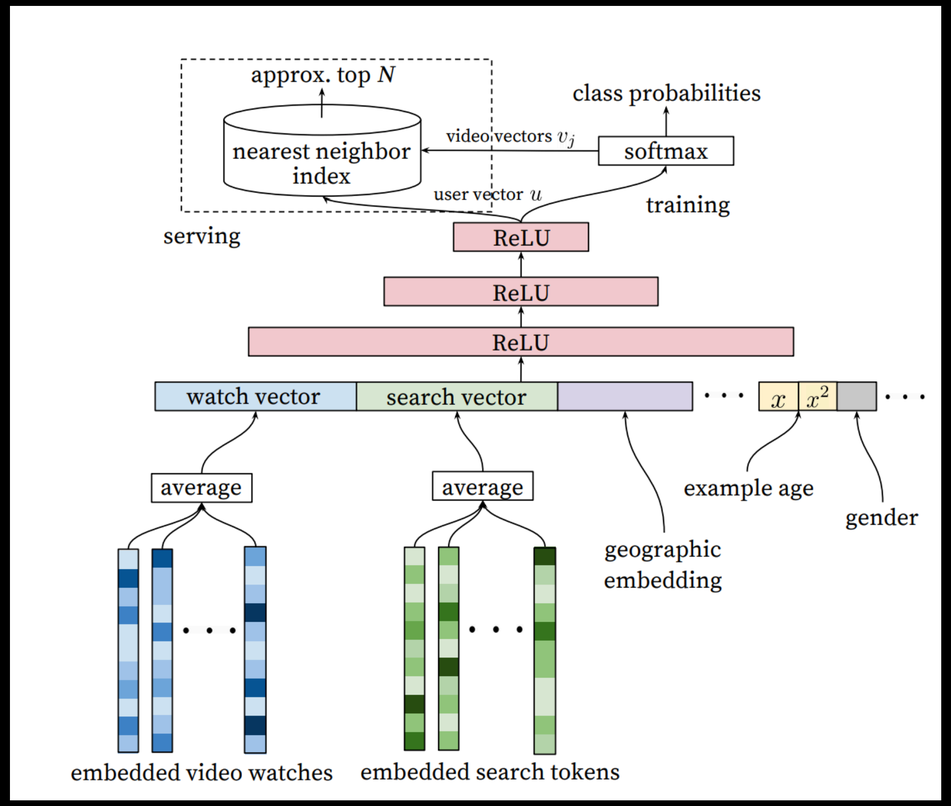




Problems?

YT(youtube) recommender:





Problems?

- New items (not users)
- New feedback

- New features
- Time-dependent features

Recurrent neural networks:

Ну не заводятся и не заводятся

Controllable Multi-Interest Framework for Recommendation:

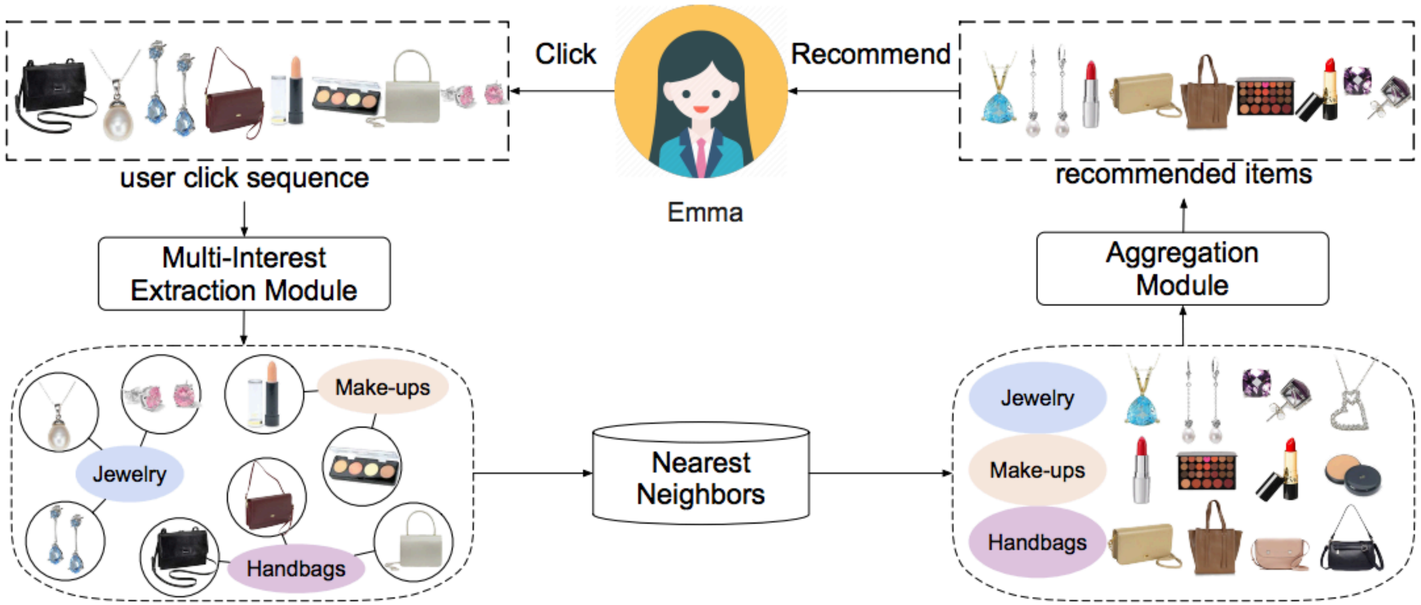
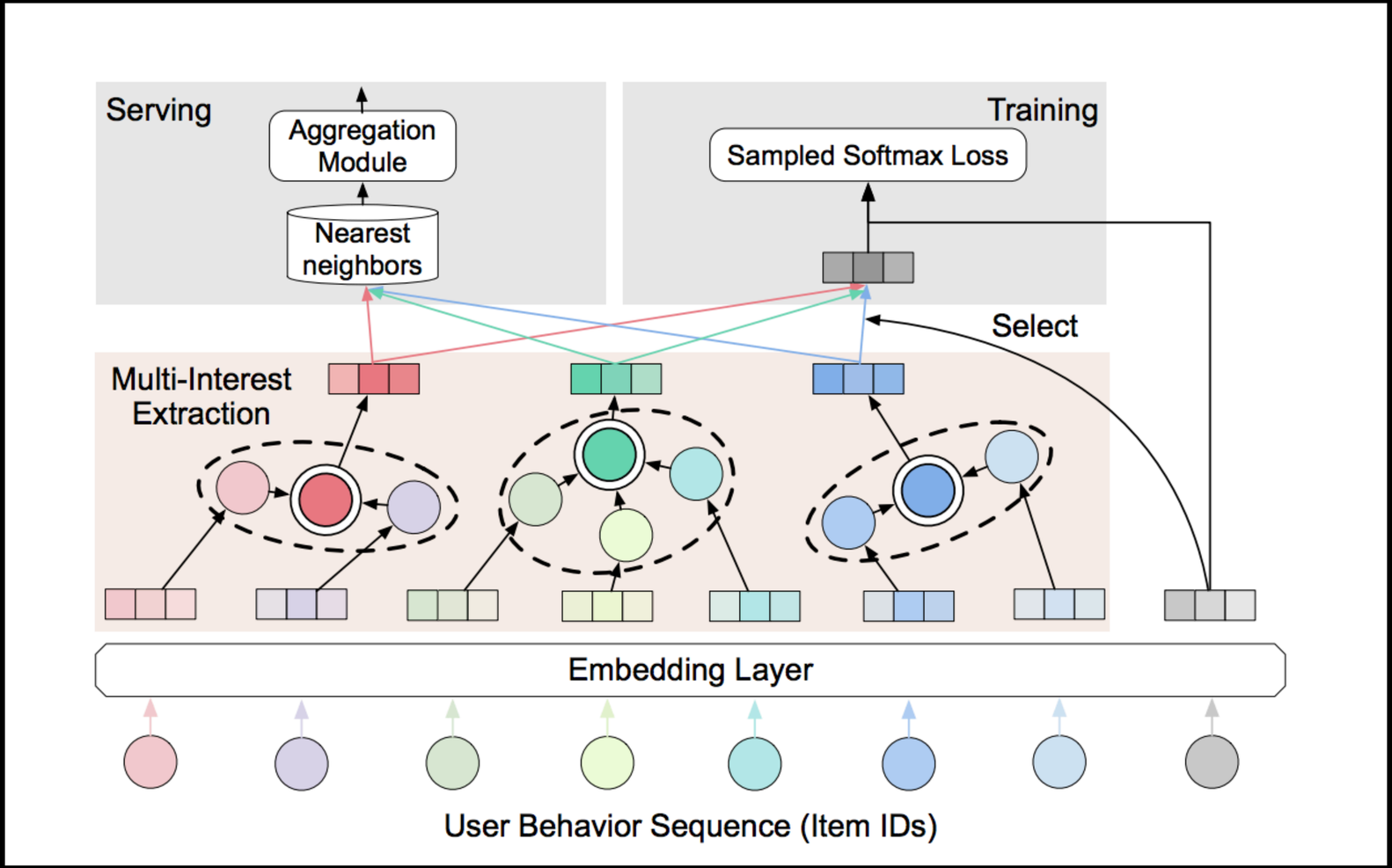


Figure 1: A motivating example of our proposed framework. An e-commerce platform user, Emma, has multiple interests including jewelry, handbags, and make-ups. Our multi-interest extraction module can capture these three interests from her click sequence. Each interest retrieves items from the large-scale item pool based on the interest embedding independently. An aggregation module combines items from different interests and outputs the overall top-N recommended items for Emma.

Source: Controllable Multi-Interest Framework for Recommendation

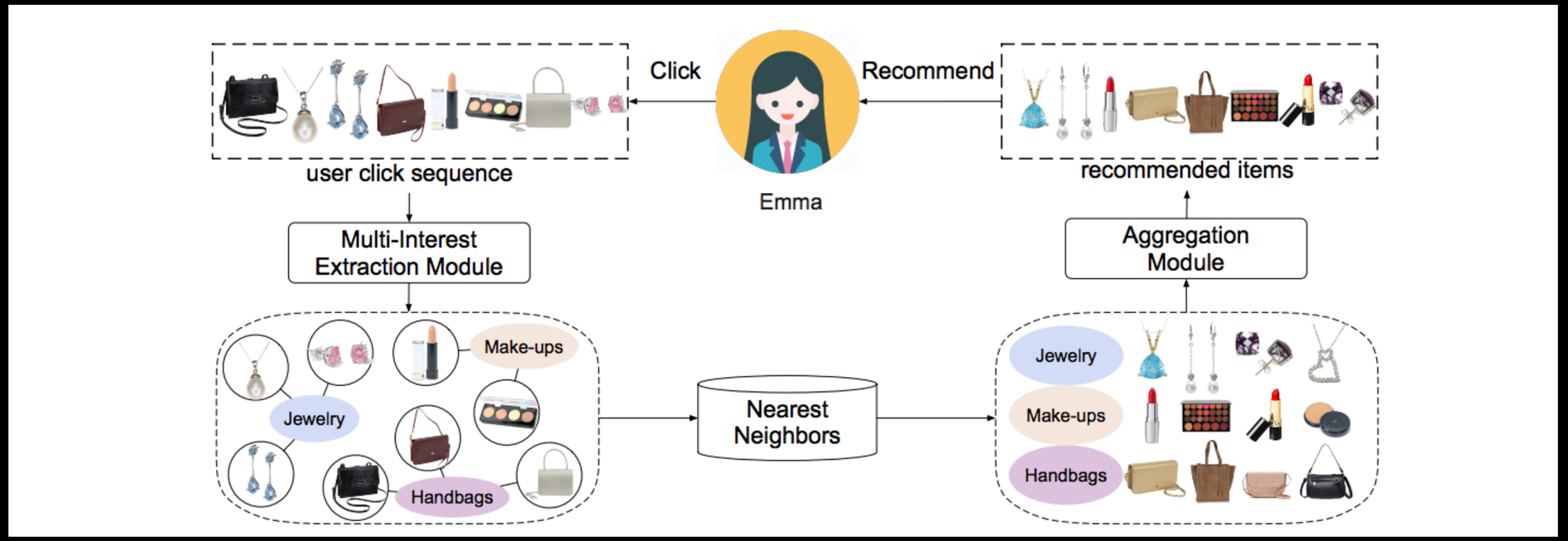
Controllable Multi-Interest Framework for Recommendation:



Controllable Multi-Interest Framework for Recommendation:

- Attention:
  - $A = softmax(W_2^T tanh(W_1 H^T))$, where H^T - prev embeddings matrix, $H = [d\ x\ n], A = n\ x\ att\_heads$
    d — ebb size
    n — history size
    att_heads — num heads

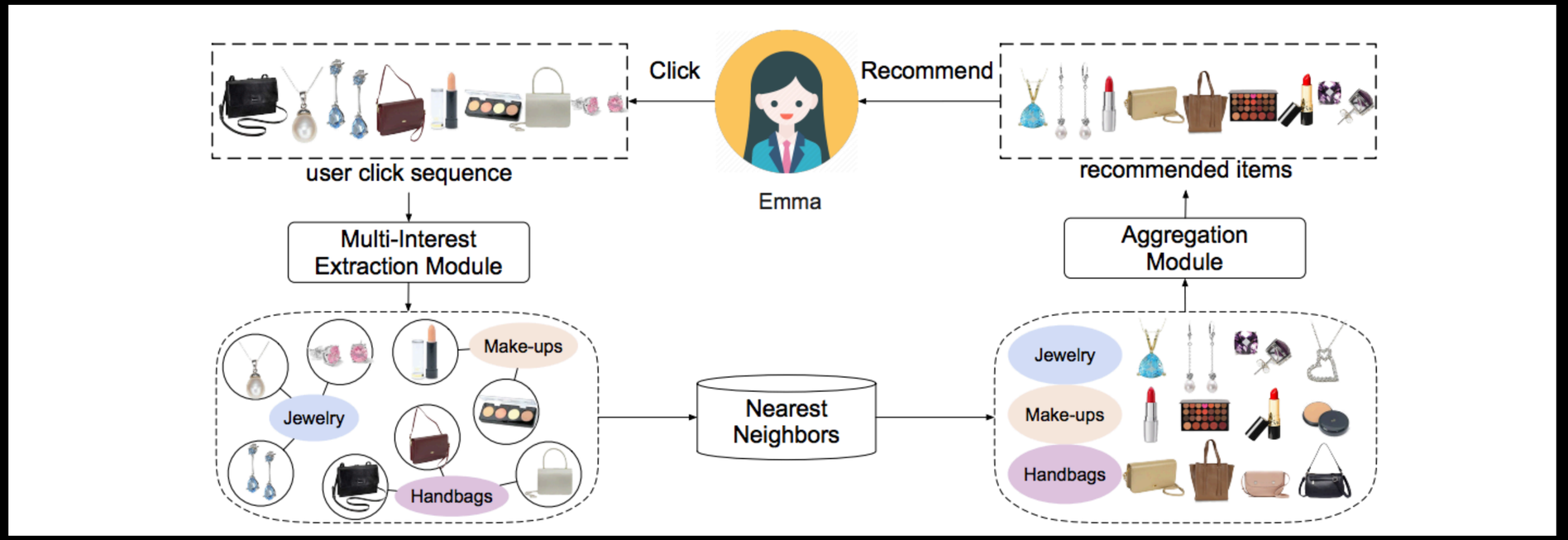
- Attention:
  - $V_U = HA;$
    V_u — user interests matrix, d x att_heads



Controllable Multi-Interest Framework for Recommendation:

- Attention:
  - $A = softmax(W_2^T tanh(W_1 H^T))$, where H^T - prev embeddings matrix, $H = [d\ x\ n], A = n\ x\ att\_heads$
    d — ebb size
    n — history size
    att_heads — num heads

- Model training:
  $v_u = V_u[:, argmax(V_u^T e_i)]$
  Sampled softmax



Controllable Multi-Interest Framework for Recommendation:

- Attention:
  - $A = softmax(W_2^T tanh(W_1 H^T))$, where H^T - prev embeddings matrix, $H = [d\ x\ n], A = n\ x\ att\_heads$
    d — ebb size
    n — history size
    att_heads — num heads

- Aggregation model:

$$f(u,i) = max_k(e_i^T, v_u^k)$$

$$Q(u,S) = \sum_{i \in S} f(u,i) + \lambda \sum_{i \in S} \sum_{j \in S} g(i,j)$$

$$g(i,j) = 1\ if\ CATE(i) \neq CATE(j)$$

- Model training:
  $v_u = V_u[:, argmax(V_u^T e_i)]$
  Sampled softmax

Controllable Multi-Interest Framework for Recommendation:

- Attention:
  - $A = softmax(W_2^T tanh(W_1 H^T))$, where H^T - prev embeddings matrix, $H = [d\ x\ n], A = n\ x\ att\_heads$
    d — ebb size
    n — history size
    att_heads — num heads

- Aggregation model:

$$f(u,i) = max_k(e_i^T, v_u^k)$$

$$Q(u,S) = \sum_{i \in S} f(u,i) + \lambda \sum_{i \in S} \sum_{j \in S} g(i,j)$$

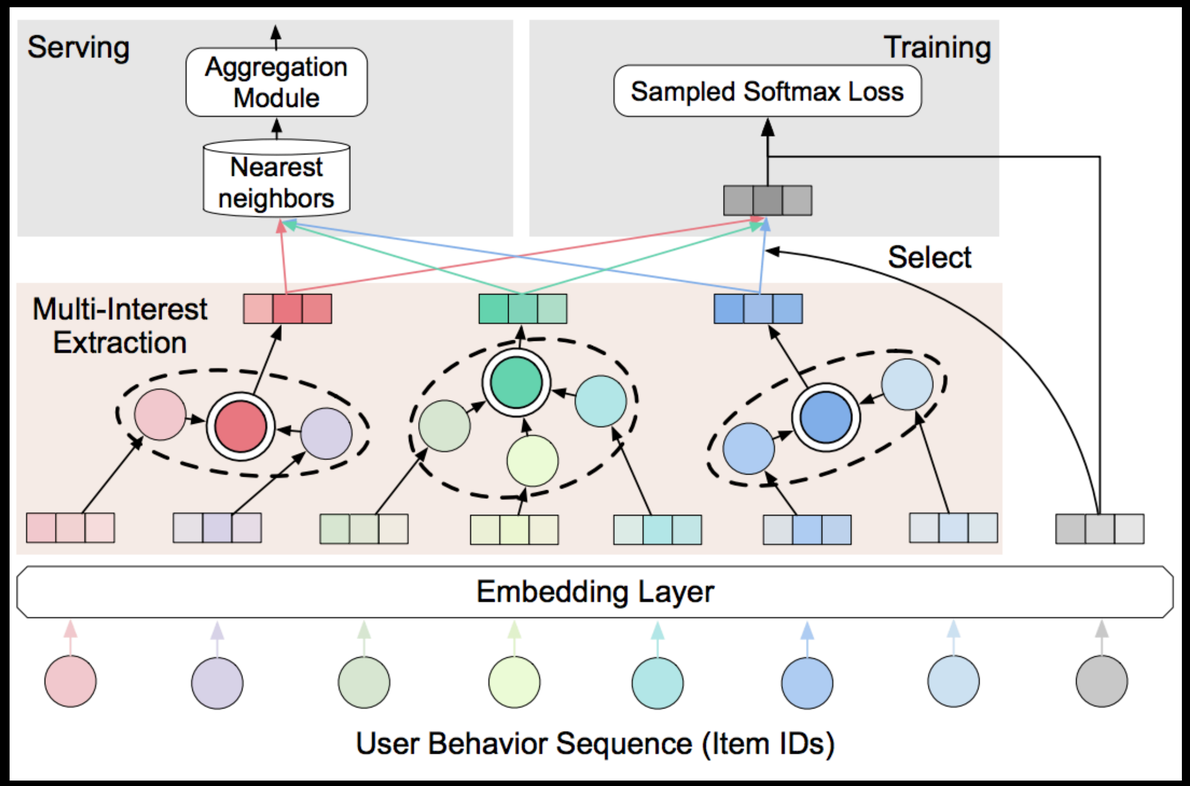
$$g(i,j) = 1\ if\ CATE(i) \neq CATE(j)$$

PRBLMS?

- Model training:

$$v_u = V_u[:, argmax(V_u^T e_i)]$$

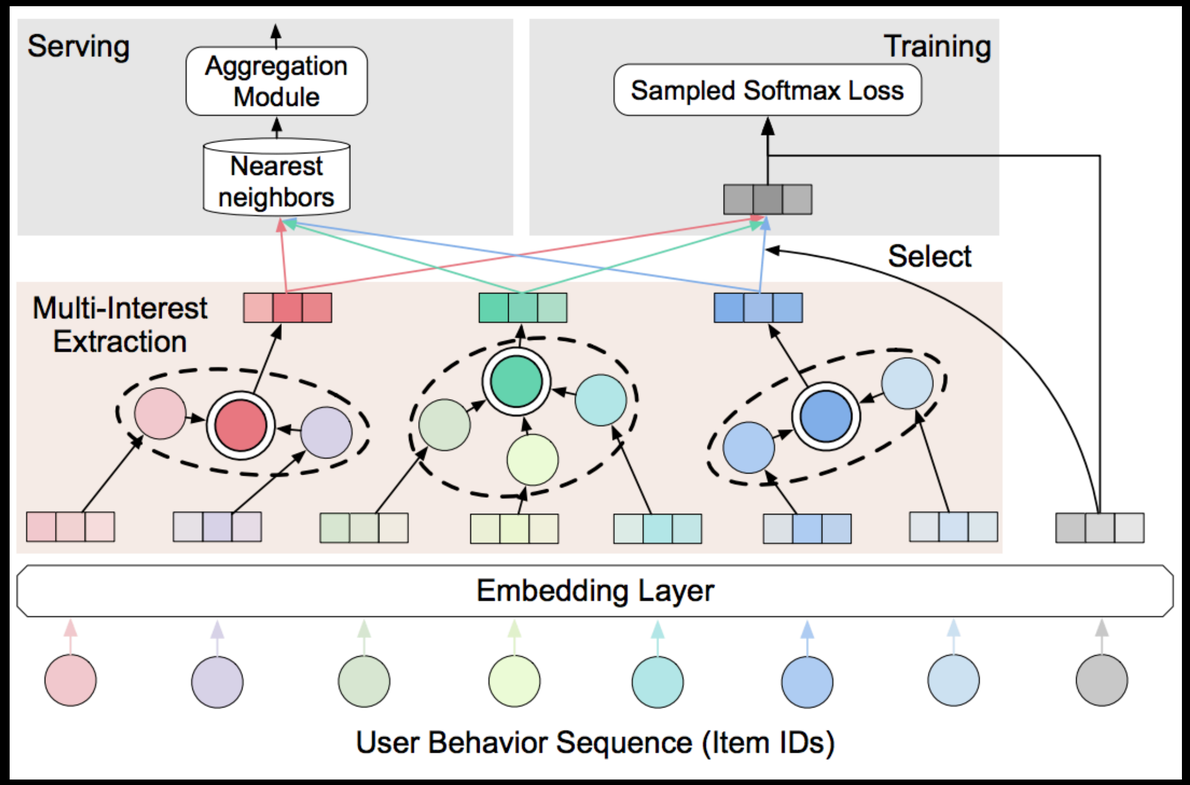
Sampled softmax



Controllable Multi-Interest Framework for Recommendation:

PRBLMS?

- Model training:
  $v_u = V_u[:, argmax(V_u^T e_i)]$ — same heads
- Pre-setted num of interests
- Over-confident heads
- No feature used



Controllable Multi-Interest Framework for Recommendation:

PRBLMS?

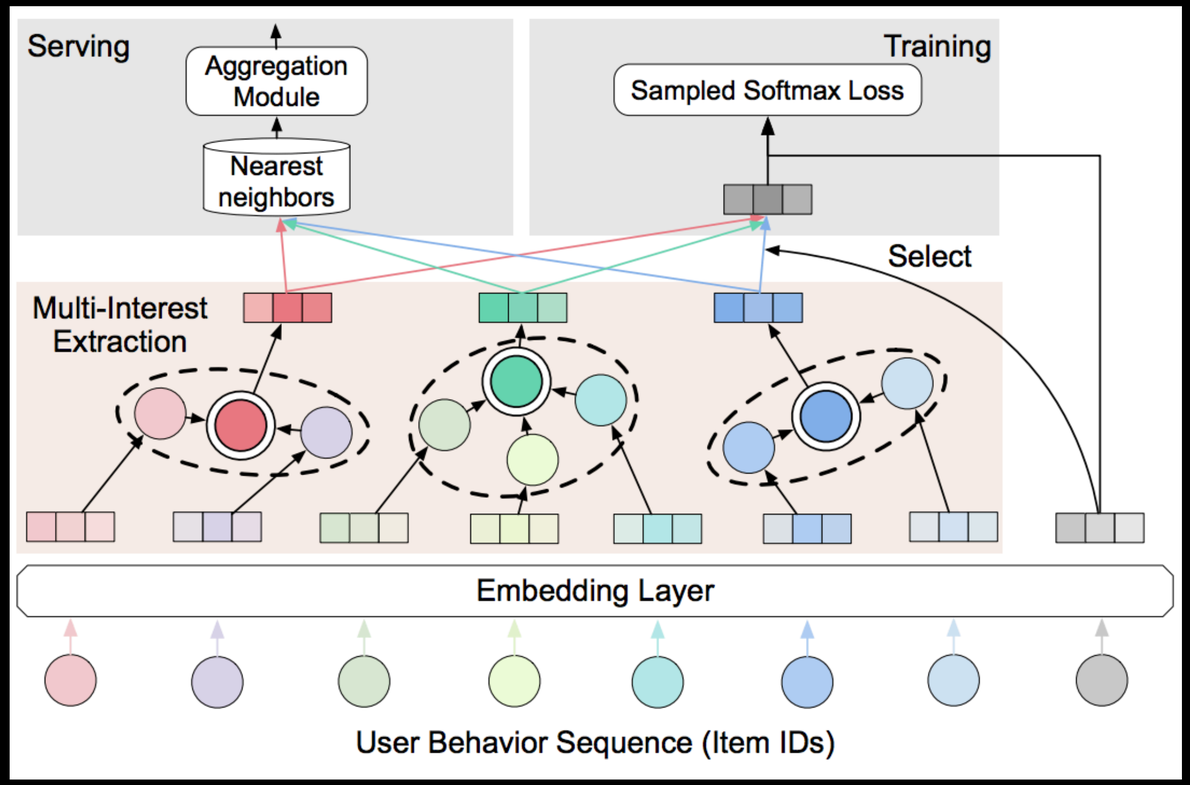
- Model training:
  $v_u = V_u[:, argmax(V_u^T e_i)]$ — same heads

  Additive component to loss:
  $loss = loss + \alpha * (1 - max\_pair\_dot)$

- Pre-setted num of interests
  …

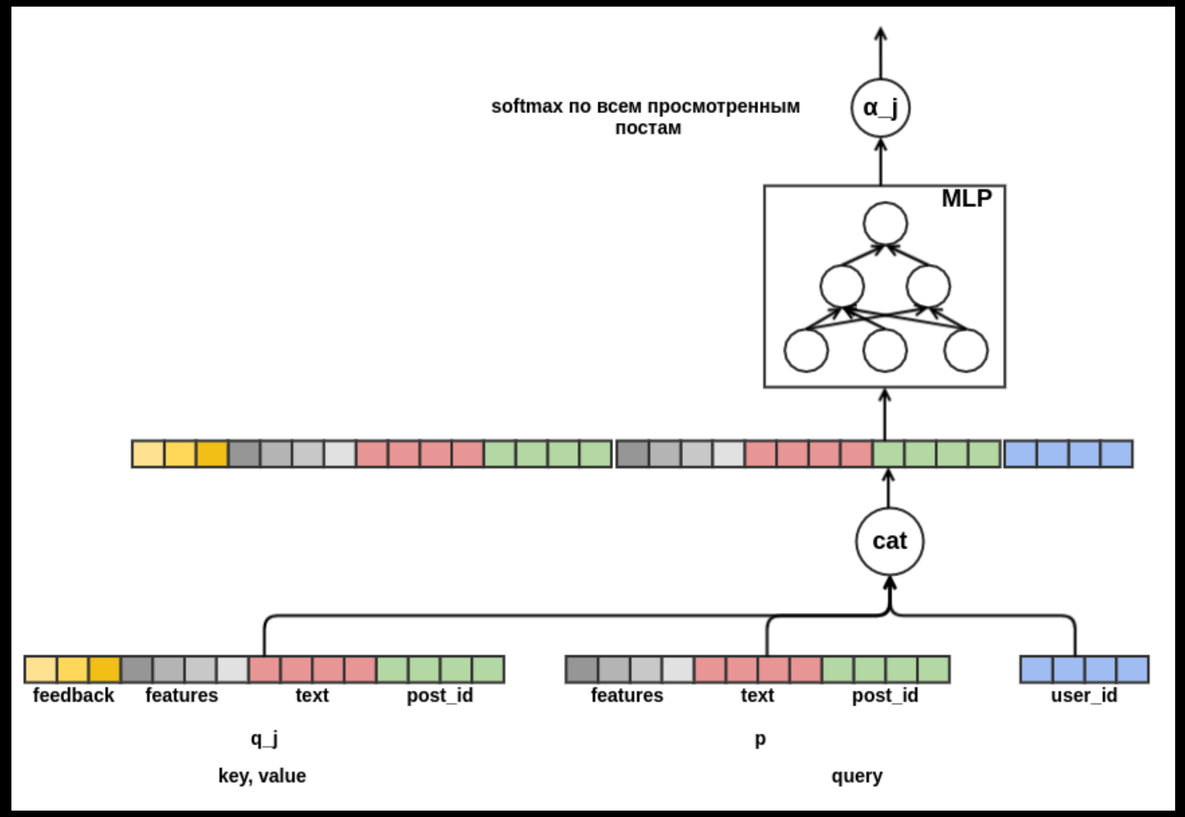
- Over-confident heads
  Temperature, label smooth, dropout

- No feature used
  Add some features, it's attention ;)



Controllable Multi-Interest Framework for Recommendation:

- **No feature used**
  Add some features, it's attention ;)



https://habr.com/ru/company/mailru/blog/445348/

MEANTIME: Mixture of Attention Mechanisms with Multi-temporal Embeddings for Sequential Recommendation:

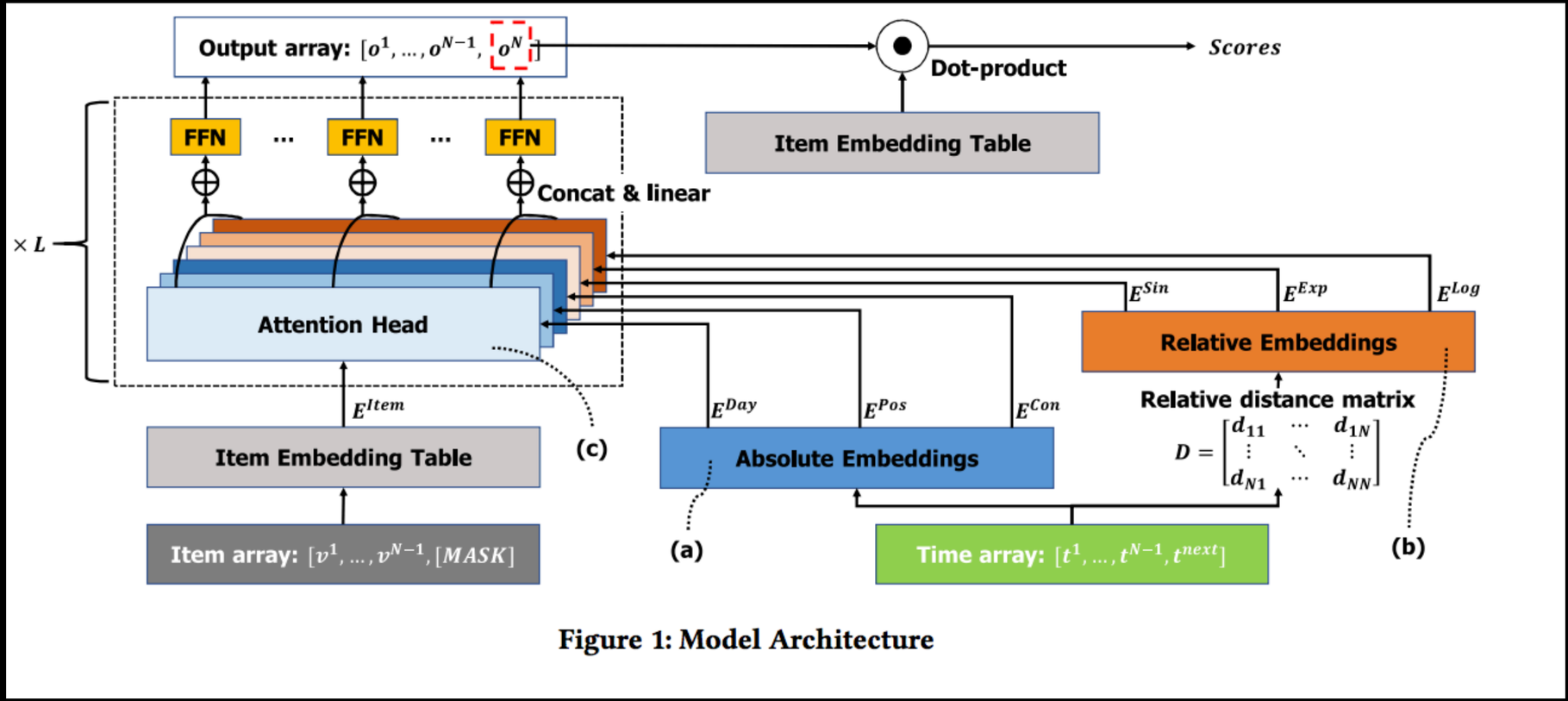


Figure 1: Model Architecture

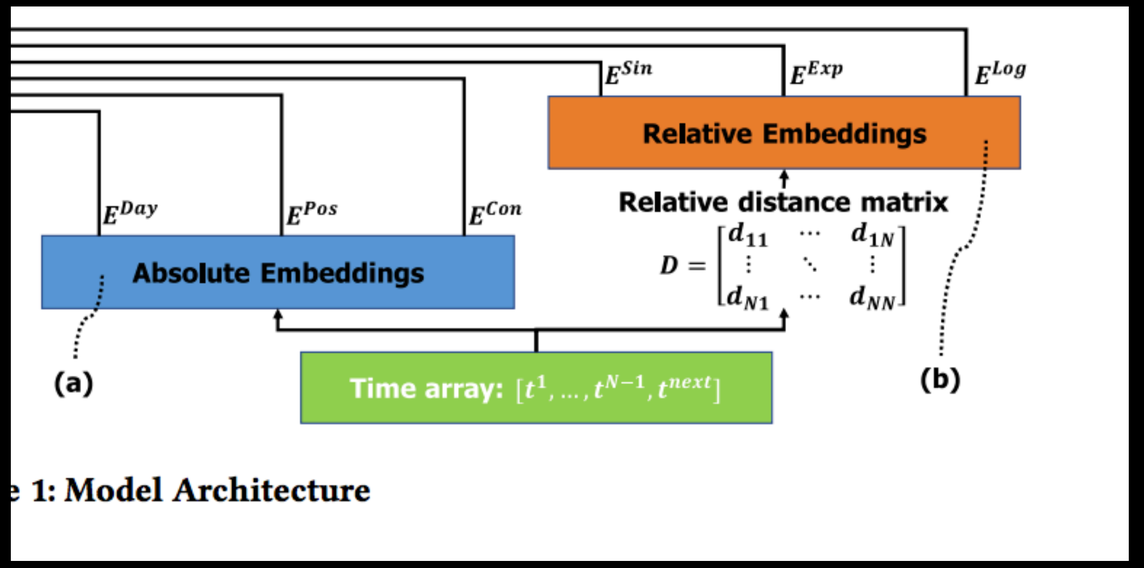
Source: MEANTIME: Mixture of Attention Mechanisms with Multi-temporal Embeddings for Sequential Recommendation:

MEANTIME: Mixture of Attention Mechanisms with Multi-temporal Embeddings for Sequential Recommendation:

- E^{pos} — relative positional embedding.
  To learn sequential patterns.

- E^{day} — embedding of a day (problems?)
  confine the attention scope within the same (or similar) day

- E^{con} — stack of repeated vectors (?) to eliminate positional bias



e 1: Model Architecture

following equation:

$$\vec{\theta}_{ab,2c} = sin(\frac{d_{ab}}{freq^{\frac{2c}{h}}}) \qquad \vec{\theta}_{ab,2c+1} = cos(\frac{d_{ab}}{freq^{\frac{2c}{h}}}) \qquad (1)$$

where $\vec{\theta}_{ab,c}$ is the c^{th} value of the vector $\vec{\theta}_{ab}$ and $freq$ is an adjustable parameter. Likewise, **Exp** encoder and **Log** encoder converts d_{ab} to $\vec{e}_{ab}$ and $\vec{l}_{ab}$ respectively by applying the following equations:

$$\vec{e}_{ab,c} = exp(\frac{-|d_{ab}|}{freq^{\frac{c}{h}}}) \qquad \vec{l}_{ab,c} = log(1 + \frac{|d_{ab}|}{freq^{\frac{c}{h}}}) \qquad (2)$$

- Sin captures periodic occurrences.
- Larger time gap is either quickly decayed to zero in **Exp**
- or manageably increased in **Log**.

Source: MEANTIME: Mixture of Attention Mechanisms with Multi-temporal Embeddings for Sequential Recommendation:

Conclusions:

- NN for rank:

  Accurate feature analysis.
  Good generalization but lack of interpretability.
  Not architecture but data analysis.

- It’s all about embeddings, but embeddings overfits easily

- Not user embedding, but user embedding as function of items.

- Bag of tricks to make things works: label smooth, time encoders, additives losses and etc.

Source: MEANTIME: Mixture of Attention Mechanisms with Multi-temporal Embeddings for Sequential Recommendation: