ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

Семинар: нейросетевое ранжирование

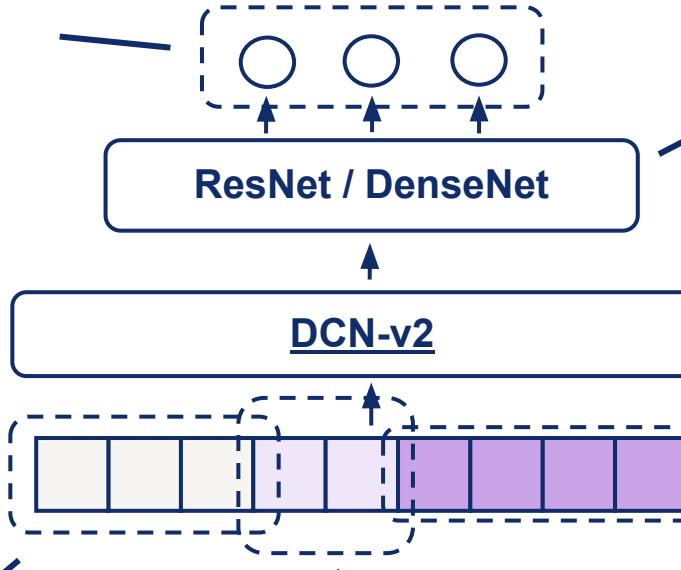
Артем Матвеев

Yandex

Яндекс

Эволюция нейросетевого ранжирования - далее ranker

Учим несколько **голов модели** на разные сигналы (просмотры, лайки, etc) с помощью **multi-task learning**



Моделируем **неявное взаимодействие признаков**

Моделируем **явное взаимодействие признаков** с помощью специальных **Deep Cross Network** слоёв

Это не всё!

- Mixture of Experts
- Knowledge Distillation
- HSTU

Кодируем **категориальные** признаки с помощью **Unified Embeddings**

Кодируем **вещественные** признаки с помощью **Piecewise Linear Encoding**

Кодируем **историю пользователя** и кандидата с помощью трансформера (**DIN**, **BST**, **TransAct**, **TransAct-V2**), а также эмбеды из любых других нейросетей

Структура семинара

1. **Кодирование категориальных признаков**
   - Unified Embeddings
2. **Кодирование вещественных признаков**
   - Piecewise Linear Encoding
3. **Нейросетевые слои**
   - DCN-V2
   - ResNet и DenseNet
4. **Кодирование истории**
   - TransAct
5. **Текущие тренды в нейросетевом ранжировании**
   - HSTU

Почему нейросеть, а не бустинг?

- **Feature engineering**:
  - Можно кодировать признаки высокой кардинальности (e.g., item id)
  - Можно подавать на вход тексты, картинки, последовательности, граф
  - Нейросети легко комбинировать (учить end-to-end, использовать эмбеддинги)
  - Специальные слои для моделирования взаимодействия признаков
  - Можно факторизовать вычисления (по пользователю, по айтему)

- **Многозадачность:**
  - **Multi-task learning:** можно учиться на много задач / сигналов сразу
  - **Knowledge transfer:** предобучение, дистилляция, обогащение редких сигналов за счет частых

- **Scaling hypothesis:** масштабирование по размеру датасета и по емкости модели

- Можно **дообучать** на новых данных

1. Кодирование категориальных признаков

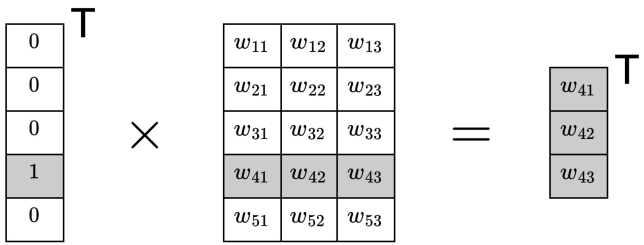
1. Кодирование категориальных признаков

- Нейросети не умеют напрямую работать с категориальными/множественными категориальными признаками. Самый простой способ закодировать категориальный признак - **one hot encoding**: e = (0, …, 1, …, 0, 1).

Можно попробовать подать такой вектор в нейросеть, но получаем огромную размерность на вход.

- Идея: сожмем one hot вектор в фиксированную размерность линейным слоем W.

Матрица эмбеддингов W



То есть:

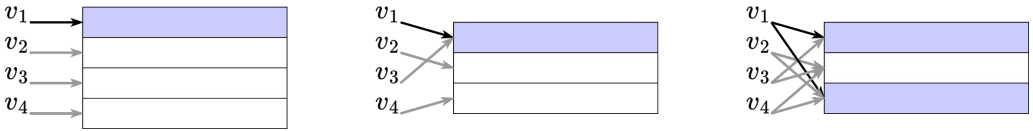
- Каждому значение признака сопоставили свой эмбеддинг.
- eW - операция embedding lookup, достающая нужный вектор из матрицы

1. Гигантские матрицы эмбеддингов

Признаки высокой кардинальности (e.g., item id) требуют много памяти. Суммарные объемы эмбеддинг-таблиц могут достигать терабайтов: e.g., 10 млн векторов размерности 256 (float32) – 76.3GB.

Потенциальные решения:

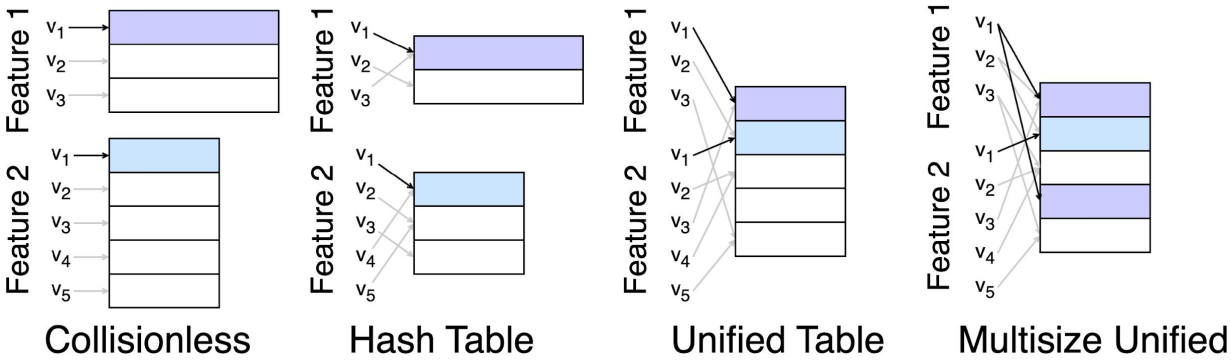
- продвинутая инфраструктура: CPU offloading, шардирование (**torchrec** [1])
- hashing trick



- semantic ids
- индуктивные представление (на основе контента)

1 https://github.com/pytorch/torchrec

1. Unified embeddings



Из статьи Unified Embedding: Battle-Tested Feature Representations for Web-Scale ML Systems

- **Единая** матрица эмбеддингов для всех признаков:
  - Для борьбы с коллизиями используем мультихэши
  - Для разных признаков используем разные сиды хэширования
  - Item id: 31415926535 → hashes: [51234, 1839, 22] → embedding: concat[emb1, emb2, emb3]

1. Почему это работает

Общая задача

Дан $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{|D|}, y_{|D|})\}$ с примерами из T категориальных признаков с словарями $\{V_1, V_2, \ldots, V_T\}$. Каждый пример $x = [v_1, v_2, \ldots, v_T]$, где $v_i \in V_i$.

- Матрица эмбеддингов $\mathbf{E} \in \mathbb{R}^{M \times d}$, отображения примера в эмбеддинг $g(\mathbf{x}; \mathbf{E})$.
- Хеш-функция $h(v) : V \to [M]$ назначает значение признака индексу строки (используется в $g(\mathbf{x}; \mathbf{E})$).
- Функция модели $f(\mathbf{e}; \boldsymbol{\theta})$ преобразует вложения в предсказание.

Задача обучения:

$$\arg\min_{\mathbf{E}, \boldsymbol{\theta}} \mathcal{L}_D(\mathbf{E}, \boldsymbol{\theta}), \quad \text{где} \quad \mathcal{L}_D(\mathbf{E}, \boldsymbol{\theta}) = \sum_{(\mathbf{x}, y) \in D} \ell(f(g(\mathbf{x}; \mathbf{E}); \boldsymbol{\theta}), y).$$

Используем $h_t(v)$ для каждого признака $t \in [T]$. Обозначаем $\mathbf{e}_m$ для m-й строки $\mathbf{E}$, и $1_{u,v}$ как индикатор коллизии хешей между u и v.

Далее предполагаем, что |T| = 2

1. Почему это работает

Рассмотрим частный случай (решаем бинарную классфикацию с помощью логистической регрессии):

$$y_i \in \{0, 1\}$$

$$D_0 = \{(x_i, y_i) \in D : y_i = 0\}$$

$$D_1 = \{(x_i, y_i) \in D : y_i = 1\}$$

$$C_{u,v,0} = |\{([u, v], y) \in D : y = 0\}|$$

$$\sigma_\theta(z) = \frac{1}{1 + \exp(-\langle z, \theta \rangle)}$$

$$z = g(x; \mathbf{E}) = [e_{h_1(x_1)}, e_{h_2(x_2)}]$$

$$\theta = [\theta_1, \theta_2],\ \theta_t \in \mathbb{R}^M$$

Функция потерь бинарной кросс-энтропии:

$$\mathcal{L}_D(\mathbf{E}, \theta) = -\sum_{(x,y) \in D_0} \log\left(\frac{1}{1 + \exp(-\langle \theta, g(x; \mathbf{E}) \rangle)}\right) - \sum_{(x,y) \in D_1} \log\left(\frac{1}{1 + \exp(\langle \theta, g(x; \mathbf{E}) \rangle)}\right)$$

1. Почему это работает

Перепишем функция потерь (через частоты совместного появления):

$$e_{u,v} = [e_{h_1(u)}, e_{h_2(v)}]$$

$$\mathcal{L}_D(\mathbf{E}, \theta) = -\sum_{u \in V_1} \sum_{v \in V_2} C_{u,v,0} \log \left(\frac{1}{1 + \exp(-\theta^\top e_{u,v})} \right) + C_{u,v,1} \log \left(\frac{1}{1 + \exp(\theta^\top e_{u,v})} \right)$$

После объединения сигмоидных функций:

$$\mathcal{L}_D(\mathbf{E}, \theta) = -\sum_{u \in V_1} \sum_{v \in V_2} C_{u,v,0} \log \exp(\theta^\top e_{u,v}) - (C_{u,v,0} + C_{u,v,1}) \log(1 + \exp(\theta^\top e_{u,v}))$$

Далее будем предполагать, что обучаем наш алгоритм с SGD. Посчитаем градиенты по эмбеддингам. Полный градиент для эмбеддинга с учетом внутри- и межпризнаковых взаимодействий:

$$\nabla_{E_{h(u)}} \mathcal{L}_D(\mathbf{E}, \theta) =$$

$$\theta_1 \sum_{v \in V_2} C_{u,v,0} - (C_{u,v,0} + C_{u,v,1}) \sigma_\theta(e_{u,v})$$

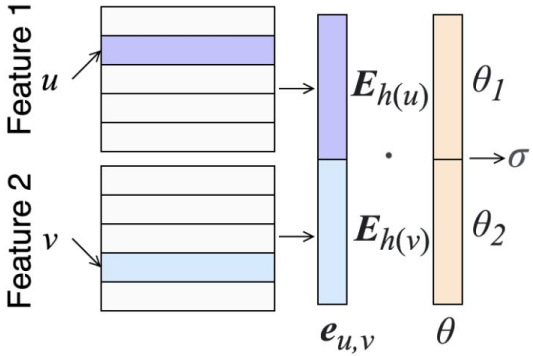
$$+\theta_1 \sum_{w \in V_1, w \neq u} 1_{u,w} \sum_{v \in V_2} C_{w,v,0} - (C_{w,v,0} + C_{w,v,1}) \sigma_\theta(e_{u,v})$$

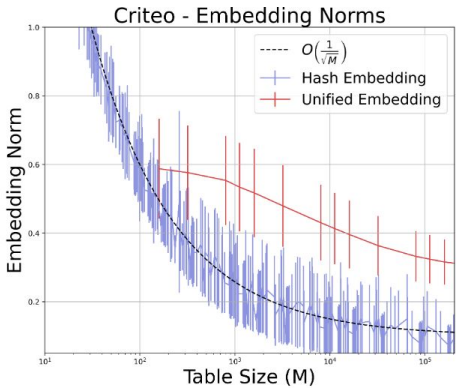
$$+\theta_2 \sum_{v \in V_2} 1_{u,v} \sum_{w \in V_1} C_{w,v,0} - (C_{w,v,0} + C_{w,v,1}) \sigma_\theta(e_{w,u})$$

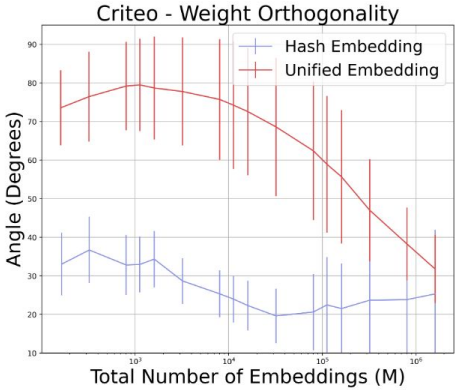
Анализируем:

- (1) collisionless компонента.
- (2) intra-feature компонента.
- (3) inter-feature компонента.
- Компоненты (2) и (3) смещают реальный градиент.
- Intra-feature bias сонаправлен с collisionless компонентой, поэтому модель не может убрать это смещение.
- В случае SGD inter-feature bias может невелирован за счет θ_1 ортогонально θ_2, т.к. во время SGD, $e_{h(u)}$ представляет собой линейную комбинацию градиентов по шагам обучения, что означает, что $e_{h(u)}$ может быть разложено на компоненты в направлении θ_1 и inter-feature компоненты в направлении θ_2. Поскольку $\langle \theta_1, \theta_2 \rangle = 0$, проекция $\theta_1^\top e_{h(u)}$ эффективно устраняет inter-feature компонент.

1. Почему это работает







Из статьи Unified Embedding: Battle-Tested Feature Representations for Web-Scale ML Systems

Вывод:

Не все коллизии одинаково проблематичны. Межпризнаковые коллизии могут быть смягчены однослойной нейронной сетью, поскольку разные признаки обрабатываются разными параметрами модели. Коллизии в рамках одного признака убираем за счет нескольких взятий хешей.

2. Кодирование вещественных признаков

2. Вещественные признаки

Вещественный признак as is использовать не получится, так как нейросети чувствительны к выбросам, масштабам, пропускам в признаке.

Популярные трансформации:

- **log1p**: x -> log(x + 1), уменьшает скошенность распределения, логнормальное приводит к нормальному (хорошо для нейросетей)
- **sigmoid squashing**: x -> σ(ax + b), сглаживает выбросы, не требует клиппинга
- **функция распределения**: x -> P(X <= x), приводит признаки в равномерное распределение на [0, 1]
- **периодические функции**: x -> [sin(2 * pi * c1 * x), cos(2 * pi * c1 * x), …], для обработки признаков, связанных с временем, например x - это часа, c1 = 1 / 24

2. Квантование (бинаризация)

Можно превратить число в категорию:

- Квантование: разбиваем значения на **бины** по квантилям
- Каждому бину соответствует категория (номер бина)
- Превратили вещественный признак в категориальный

2. Кусочно-линейное кодирование (PLE)

$$\texttt{PLE}(x) = [e_1, \ldots, e_T] \in \mathbb{R}^T$$

$$e_t = \begin{cases} 0, & x < b_{t-1} \texttt{ AND } t > 1 \\ 1, & x \geq b_t \texttt{ AND } t < T \\ \frac{x - b_{t-1}}{b_t - b_{t-1}}, & \text{otherwise} \end{cases} \quad (1)$$

where PLE stands for "**p**eicewise **l**inear **e**ncoding". We provide the visualization in Figure 1.

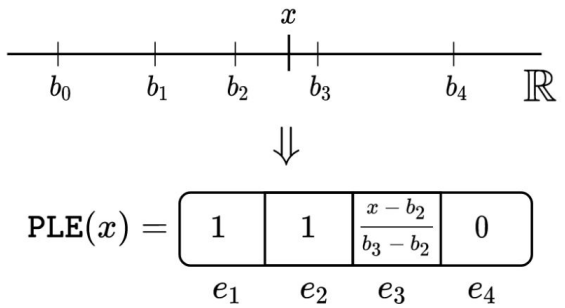


Figure 1: The piecewise linear encoding (PLE) in action for $T = 4$ (see Equation 1).

Из статьи On Embeddings for Numerical Features in Tabular Deep Learning

- Кодируем не только номер бина, но и где внутри него находимся
- SOTA-подход

2. Кусочно-линейное кодирование (PLE)

Свойства:

- Эмбеддинги вычисляются независимо для каждого признака
- В MLP-архитектурах эмбеддинги конкатенируются в один вектор
- В Transformer-архитектурах эмбеддинги используются без дополнительных преобразований
- При T = 1 PLE эквивалентно скалярному представлению
- В отличие от категориальных признаков, PLE учитывает упорядоченность числовых данных
- Наиболее распространенный подход к построению бинов — разбиение по квантилям эмпирического распределения числового признака.

3. Нейросетевые слои

3. Нейросетевые слои

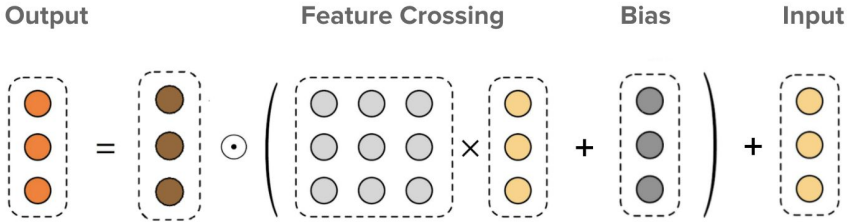
- Знаем, что кросс-признаки очень полезны в feature engineering.
- Пусть есть набор признаков на вход модели [x1, x2].
- Явным взаимодействием признаков будем называть признак, например, вида: x1 * x2
- Неявным взаимодействием будем называть, например, DNN([x1, x2]).
- MLP над конкатенацией векторов не может выучить скалярное произведение[1]
- => Нужно явно перемножить признаки!

1 Neural Collaborative Filtering vs. Matrix Factorization Revisited

3. Нейросетевые слои: DCN-v2

Cross-слой

- L слоев моделируют всевозможные комбинации признаков степени L + 1
- Diminishing returns: максимум профита при 2-3 слоях



$$x_{i+1} = x_0 \odot (W \times x_i + b) + x_i$$

THEOREM 4.1 (BITWISE). *Assume the input to an l-layer cross network be* $\mathrm{x} \in \mathbb{R}^d$, *the output be* $f_l(\mathrm{x}) = 1^\top \mathrm{x}^l$, *and the* i^{th} *layer is defined as* $\mathrm{x}^i = \mathrm{x} \odot W^{(i-1)}\mathrm{x}^{i-1} + \mathrm{x}^{i-1}$. *Then, the multivariate polynomial* $f_l(\mathrm{x})$ *reproduces polynomials in the following class:*

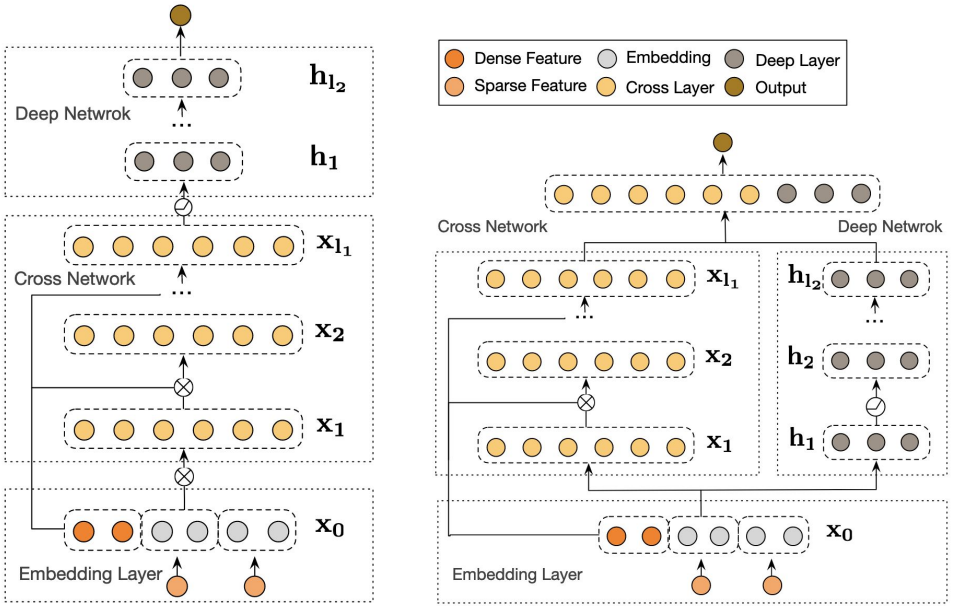
$$\left\{ \sum_{\boldsymbol{\alpha}} c_{\boldsymbol{\alpha}} \left(W^{(1)}, \ldots, W^{(l)} \right) x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d} \;\middle|\; 0 \le |\boldsymbol{\alpha}| \le l+1, \boldsymbol{\alpha} \in \mathbb{N}^d \right\},$$

where $c_{\boldsymbol{\alpha}} = \sum_{\mathrm{j} \in C_l^{|\boldsymbol{\alpha}|-1}} \sum_{\mathrm{i} \in P_{\boldsymbol{\alpha}}} \prod_{k=1}^{|\boldsymbol{\alpha}|-1} w_{i_k i_{k+1}}^{(j_k)}$, $w_{ij}^{(k)}$ *is the* $(i,j)^{th}$ *element of matrix* $W^{(k)}$, *and* $P_{\boldsymbol{\alpha}} = \text{Permutations}\left(\cup_i \{\underbrace{i, \ldots, i}_{\alpha_i \text{ times}} \mid \alpha_i \neq 0\}\right)$.

Из статьи DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems

3. Нейросетевые слои: DCN-v2

- На практике лучше Stacked
- Проблема: очень тяжелый в применении слой, т.к. работает над конкатенацией векторов всех признаков.

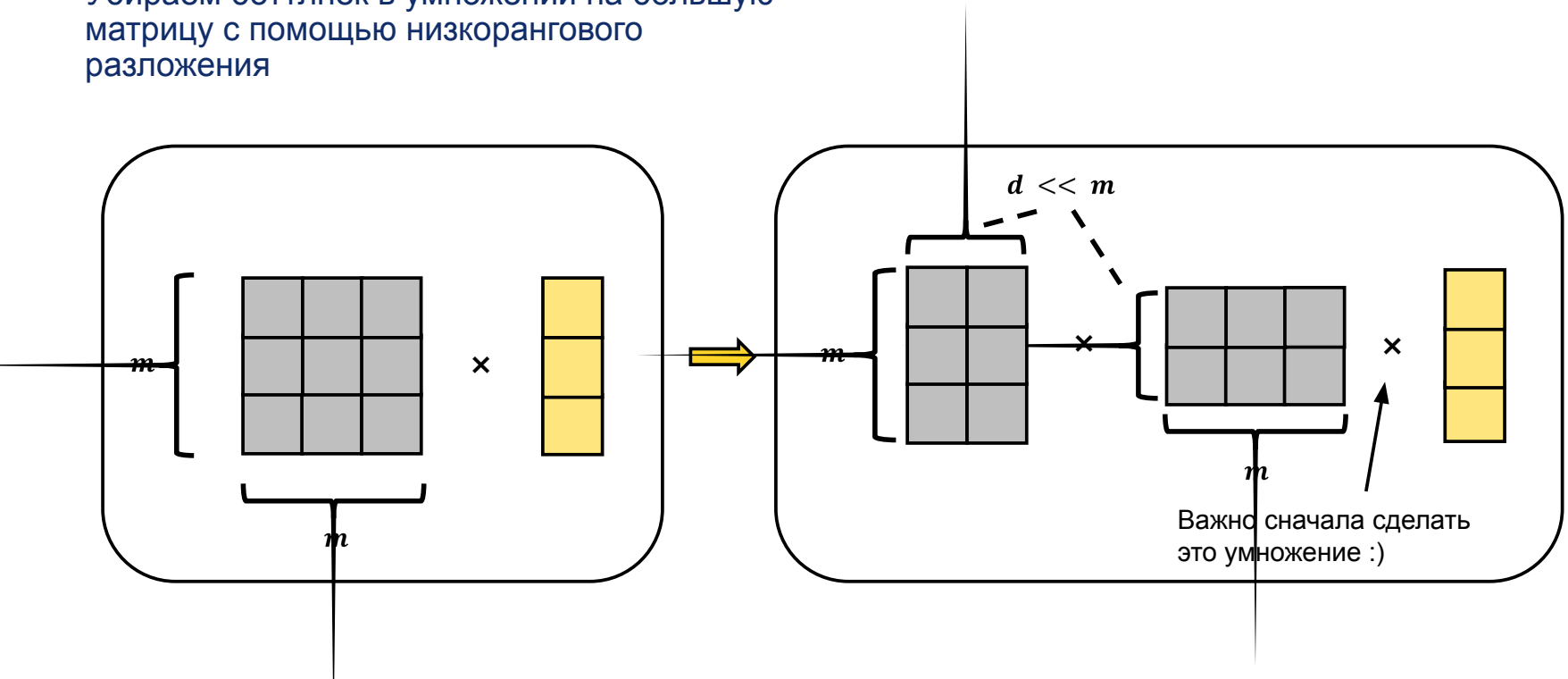


(a) Stacked (b) Parallel

Из статьи DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems

3. Низкоранговый DCN

Убираем боттлнек в умножении на большую матрицу с помощью низкорангового разложения



3. DCN-v2

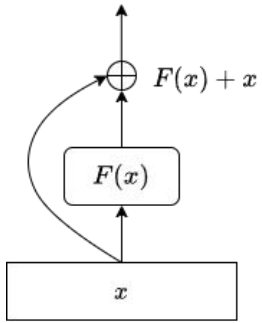
- Говорят, что DCN **явно моделирует взаимодействия низкого порядка** между признаками
- Эффективен только над сырыми эмбеддингами, а не абстрактными векторами (например, не на выходе трансформера)

… нужно комбинировать с MLP:

- MLP **неявно моделирует взаимодействия высокого порядка** между признаками
- На практике лучше сначала сделать DCN, потом MLP (параллельное применение работает хуже); перед MLP обычно сужаем вектор из DCN
- DNN выучивает только неявные взаимодействия, плохо аппроксимирует dot-product => нужные глубокие сети.
- CrossNet добавляет явные взаимодействия признаков => не нужны глубоки DNN => может применять в рантайме.

3. Нейросетевые слои: альтернативы MLP

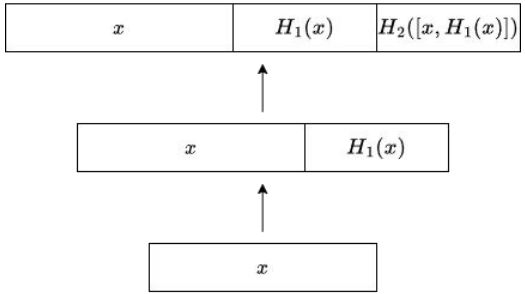
ResNet[1]



$$\text{Output} = x + F(x)$$

- Каждый блок доуточняет / корректирует вход
- Можно занулить лишние слои
- Градиентный шорткат для борьбы с затуханием

DenseNet[2]

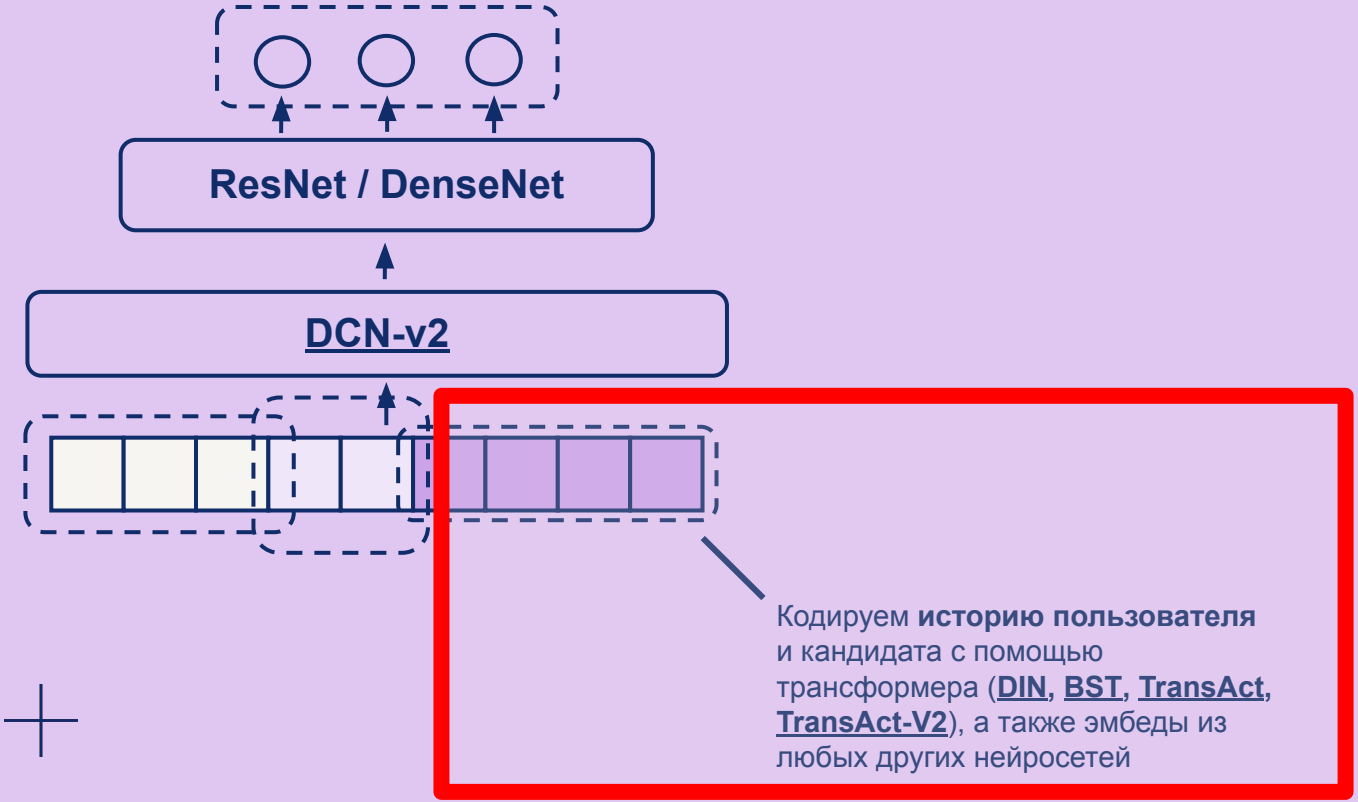


$$\text{Output} = \text{concat}\big(x, F(x)\big)$$

- Каждый новый слой расширяет представление
- Модель увеличивает выразительность и ничего не забывает

1 Deep Residual Learning for Image Recognition
2 Densely Connected Convolutional Networks

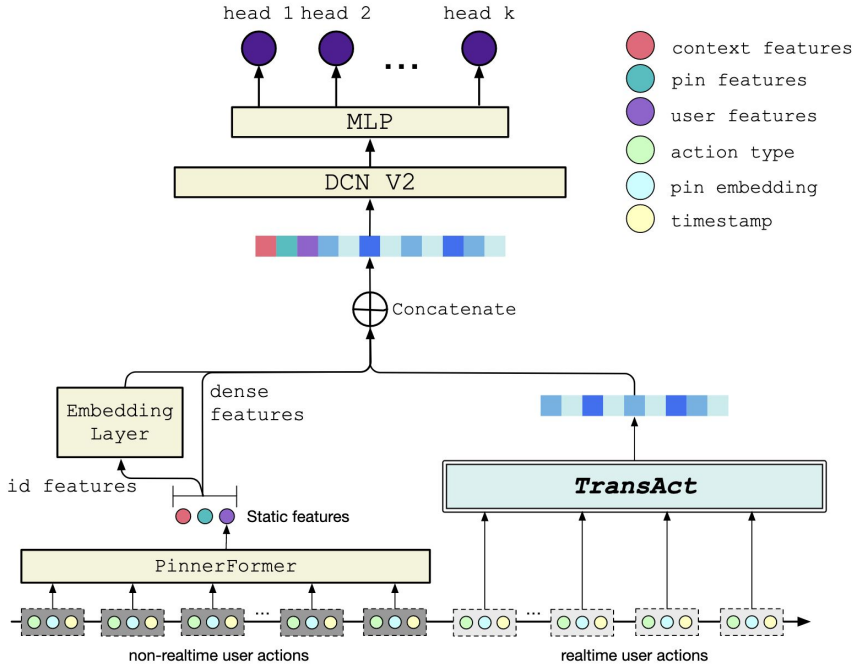
4. Кодирование истории



Кодируем **историю пользователя** и кандидата с помощью трансформера (**DIN**, **BST**, **TransAct**, **TransAct-V2**), а также эмбеды из любых других нейросетей

4. Кодируем историю: TransAct

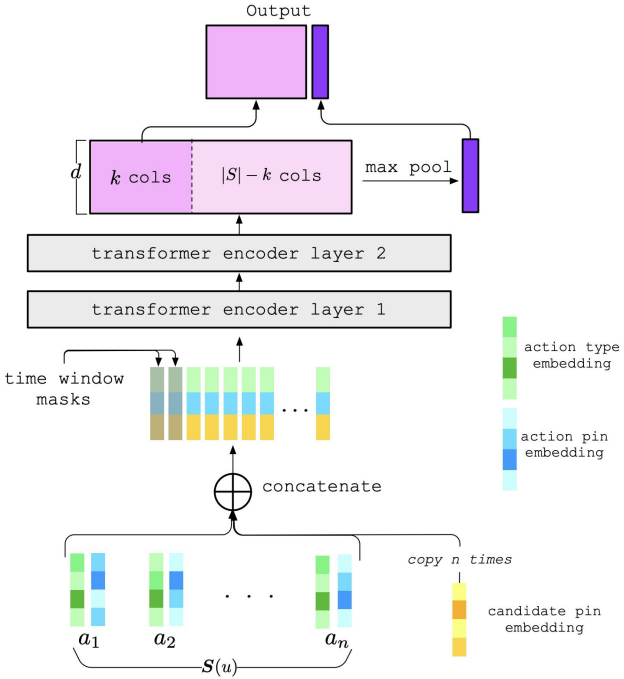
- Pinterest, основная единица контента = pin (картинка).
- Два трансформера: оффлайн и реалтайм.
- Событие на вход в трансформер кодируется через контент pin-а, timestamp, фидбек пользователя на этот pin (like, share, view, …).
- Пользовательские эмбеддинги из обоих трансформеров конкатенируется к полному набору фичей и подаются на вход в нейросетевой ранкер.
- Категориальные признаки низкой кардинальности кодируются через Embedding Layer.



Из статьи [TransAct: Transformer-based Realtime User Action Model for Recommendation at Pinterest]

4. TransAct

- Target-aware модель, раннее связывание кандидата с историей пользователя.
- Хотим, чтобы рекомендации не были похожи на самые последние взаимодействия пользователя. Применяется случайная маска к самым свежим событиям в истории при обучении, чтобы трансформер не смотрел на самые свежие события.
- Берется k самых свежих выходных состояний трансформера. MaxPooling всех выходов. Далее эти k состояний конкатенируются с MaxPooling вектор, распрямляются в один вектор и подаются на вход нейросетевому ранкеру.
- Всего 2 слоя трансформера, DeepSeekV3 = 61 слой.



Из статьи TransAct: Transformer-based Realtime User Action Model for Recommendation at Pinterest

5. Тренды в нейросетевом ранжировании

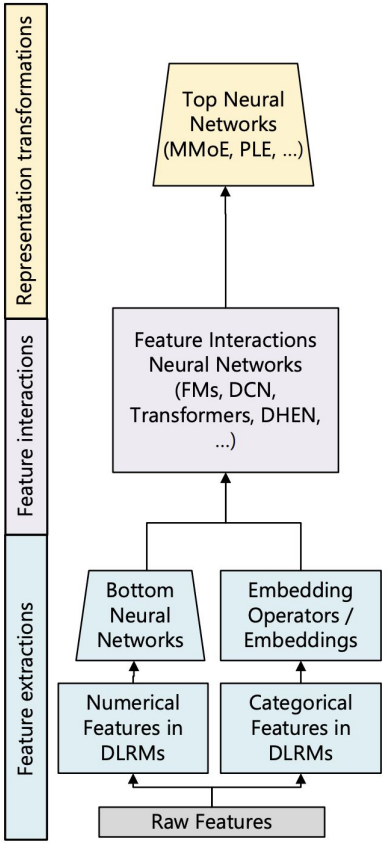
5. Текущие тренды в нейросетевом ранжировании

- В других доменах ручные признаки уже не используются. В RecSys над ними работали тысячи инженеров десятки лет, побить end-to-end нейросетью не так просто.
- Все признаки строятся на основе истории, таргета (айтем) и контекста.
- Если на вход в сильную нейросеть подать всю историю, таргет и контекст, то по Bitter Lesson она должна все выучить сама.

Область	Раньше	Сейчас побеждает
NLP	Лингвистические правила и фичи	Large Decoder-Only Transformers
CV	Ручная разработка признаков (HOG, SIFT)	CNN, ViT
Speech	MFCC + HMM	End-to-end модели (wav2vec, Whisper)
RecSys	Градиентный бустинг, ручные фичи	DLRM, DeepFM, DCN, Transformers

5. Actions Speak Louder Than Words

- Это DLRM
- Много feature engineering-а
- От этого отказываются: Meta, Alibaba, Meituan, ByteDance, … в пользу автоматического выучивания счетчиков.



Из статьи Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations

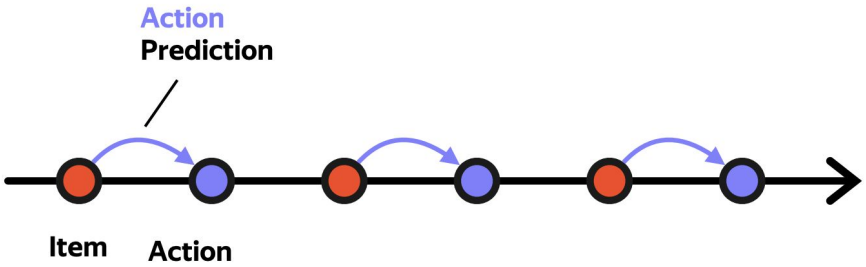
5. Генеративная постановка

- Представим пользователя в виде последовательности
- Добавим статические признаки (возраст, пол, страна, город, etc)
- Откажемся от вещественных признаков (избавляемся от feature engineering)
- Item-action interleaving (отдельный токен для айтема, отдельный для экшна)

$$x_j \quad i_0, a_0, i_1, a_1, \ldots, i_{n-1}, a_{n-1}$$

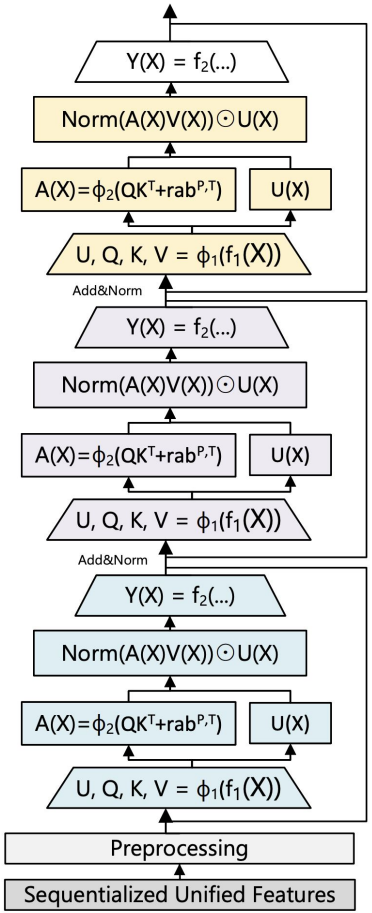
$$y_j \quad a_0, \emptyset, a_1, \emptyset, \ldots, a_{n-1}, \emptyset$$

- Получаем target-aware модель



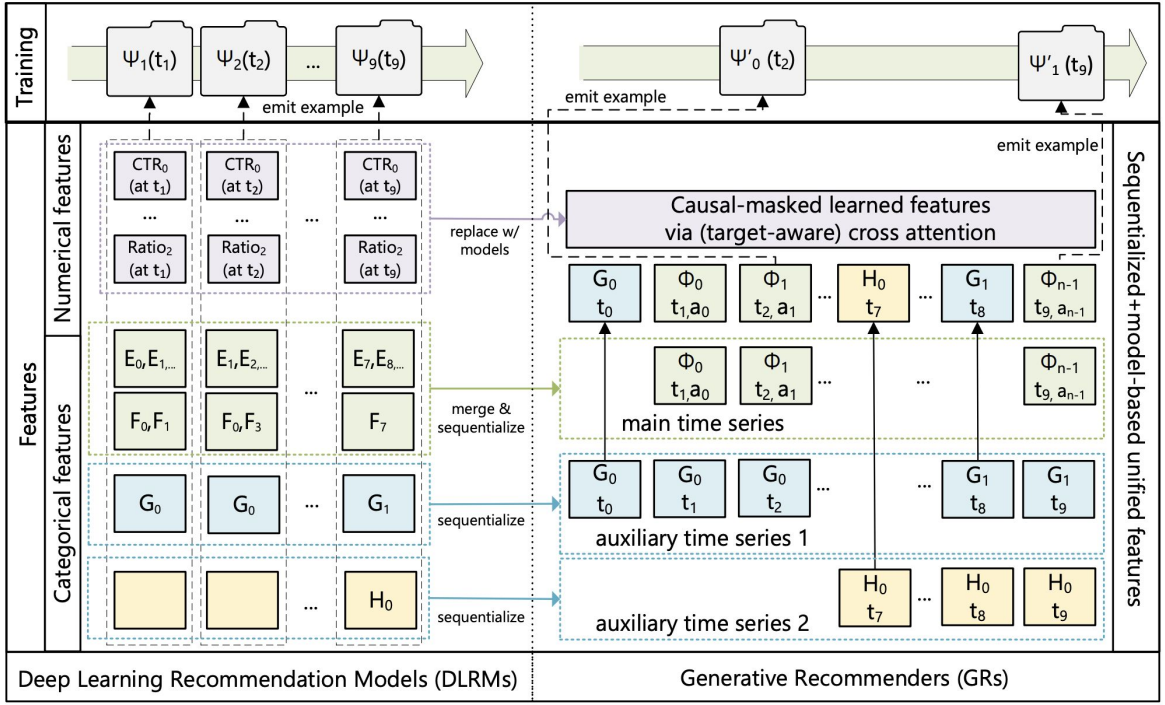
5. HSTU

- Самые полезные признаки в ранжировании — всевозможные счётчики, особенно user-item
- **Каузальный трансформер без позиционной информации не может даже выучить свою позицию! (посчитать количество токенов до себя)**
- Софтмакс плохо улавливает интенсивность элементов в последовательности — избавимся!
- Уберём FFN (сильно уменьшим память на активации)
- Добавить в Attention матрицу U для поэлементных умножений, похожих на всякие feature interaction слои



Из статьи Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations

5. Actions Speak Louder Than Words



Из статьи Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations

6. Домашняя работа: трансформер над историей пользователя для кандидатогенерации