Яндекс

Нейросетевое ранжирование

ШАД, курс по рекомендательным системам, весна 2025

Кирилл Хрыльченко

Введение

Примеры сервисов с **нейросетевыми рекомендациями:**

- YouTube, Netflix, TikTok, Spotify
- Amazon, eBay, Etsy
- Instacart, Delivery Hero
- Twitter/X
- Facebook Ads, Google Ads
- Яндекс: Музыка, Директ, Поиск, Лавка, Маркет, Кинопоиск, Карты, etc

В поиске, рекомендациях и рекламе используются **похожие нейросетевые технологии**

Структура нейросетевых лекций

Сегодня: Нейросетевое ранжирование

- Как обрабатывать признаки
- Нейросетевые слои и архитектуры
- Многозадачность

Далее: Нейросетевой отбор кандидатов

- Двухбашенные модели, трансформеры и т.д.

…

Последняя лекция курса: Тренды в рекомендательных системах

- Генеративные модели, семантические айдишники и т.д.

Сегодняшняя лекция

Преимущества нейросетей

- Проблемы градиентного бустинга
- Bitter lesson

Обработка признаков

- Категориальные и вещественные признаки

Нейросетевые слои

- Кросс-признаки, факторизационные машины
- DeepFM, DLRM, DCN-v2

Многозадачность

- Mixture of experts
- Self-supervised pre-training
- Distillation

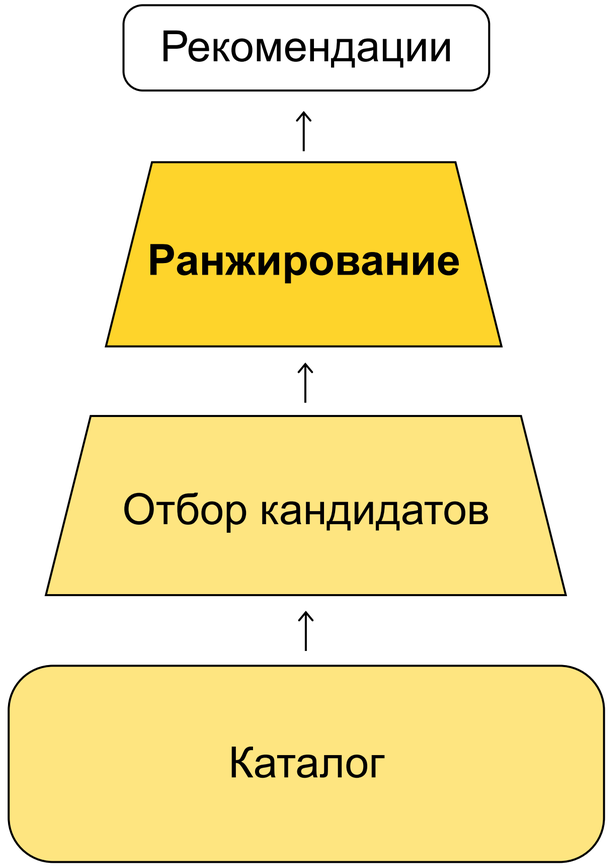
01

Зачем нужны нейросети

Или почему не обойтись градиентным бустингом?

Recap ранжирования

- Использует **всю доступную информацию** про пользователя (user) и айтемы (item)
- **Раннее связывание** (user-item features), e.g. счётчики
- Выдает для каждого айтема скаляр (скор, оценку), по которому их можно **отранжировать**
- **Impression-aware:** учится именно на показанных айтемах, $P(feedback \mid user, item)$



Проблемы градиентного бустинга

Feature engineering:

- Сложно кодировать признаки высокой кардинальности (e.g., item id)
- Сложно работать с текстами/картинками/графами
- Неудобно комбинировать с нейросетями (учить end-to-end, использовать эмбеддинги)

Одноголовая архитектура – предсказываем одну величину:

- Сложно учитывать несколько сигналов
- Сложно учиться на редкие сигналы

Плохо масштабируется по размеру датасета и по емкости модели

Для борьбы с **distribution drift** нужно переобучать с нуля (нельзя дообучать)

… линейные модели – baby нейросети :)

Преимущества нейросетей

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

Bitter lesson[1]

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

"The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin."

Richard Sutton, *The Bitter Lesson* (2019)

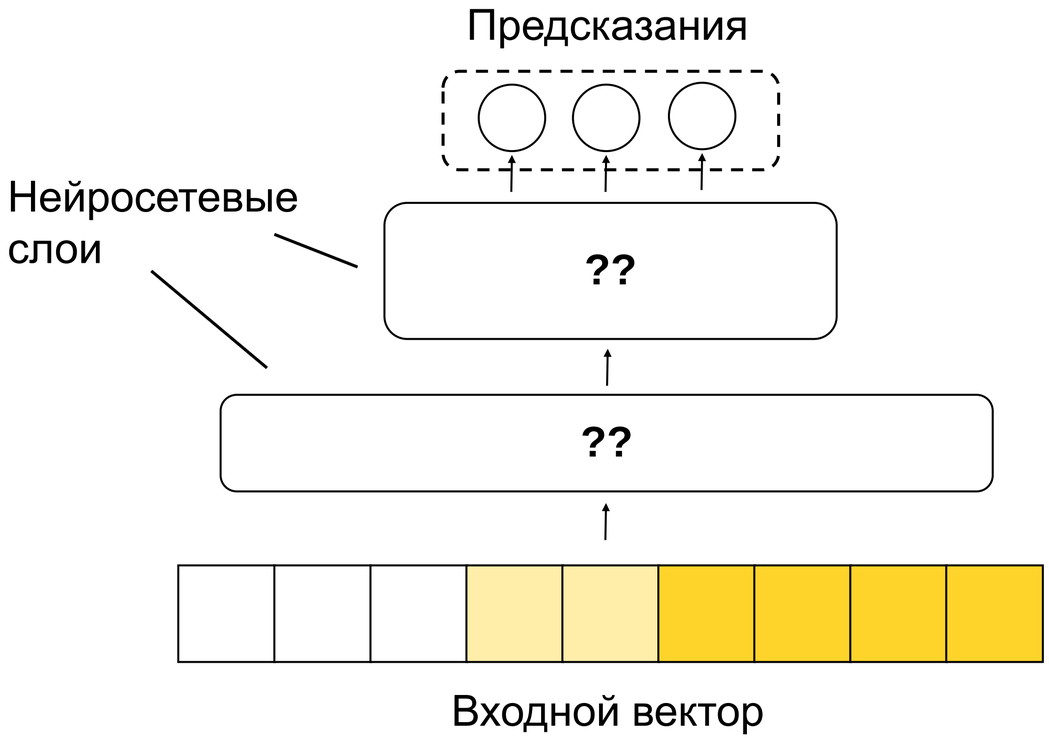
1 http://www.incompleteideas.net/IncIdeas/BitterLesson.html

Проблемы нейросетей

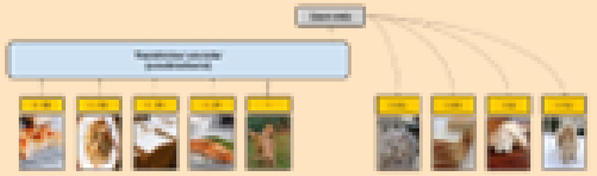
- Гораздо сложнее завести (чем бустинг)
- Модель может выглядеть рабочей, но не давать профита поверх бустинга
- Много подводных камней: как закодировать входы, какую архитектуру выбрать, лоссы, гиперпараметры, оптимизация
- Требуют умения работать с GPU, делать распределенное обучение; рантайм требует уметь внедрять нейросети

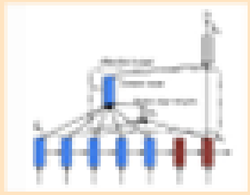
Ранжирующая нейросеть

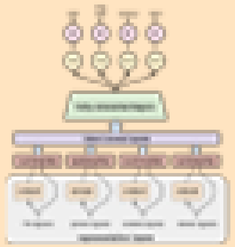
- Начинаем делать **ранжирующую нейросеть**:
  - Формируем вектор
  - Применяем нейросетевые слои
  - Получаем на выходе скаляр (или тоже вектор)



Эволюция Pinterest





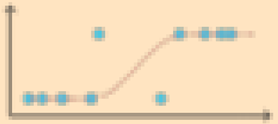


PinnerFormer

Attention Sequence

Deep Multitask NN model

GBDT Model + LR



Neural Network Model (Deep and Wide)*

Transformer Sequence

First ML model

Native Tensorflow Serving

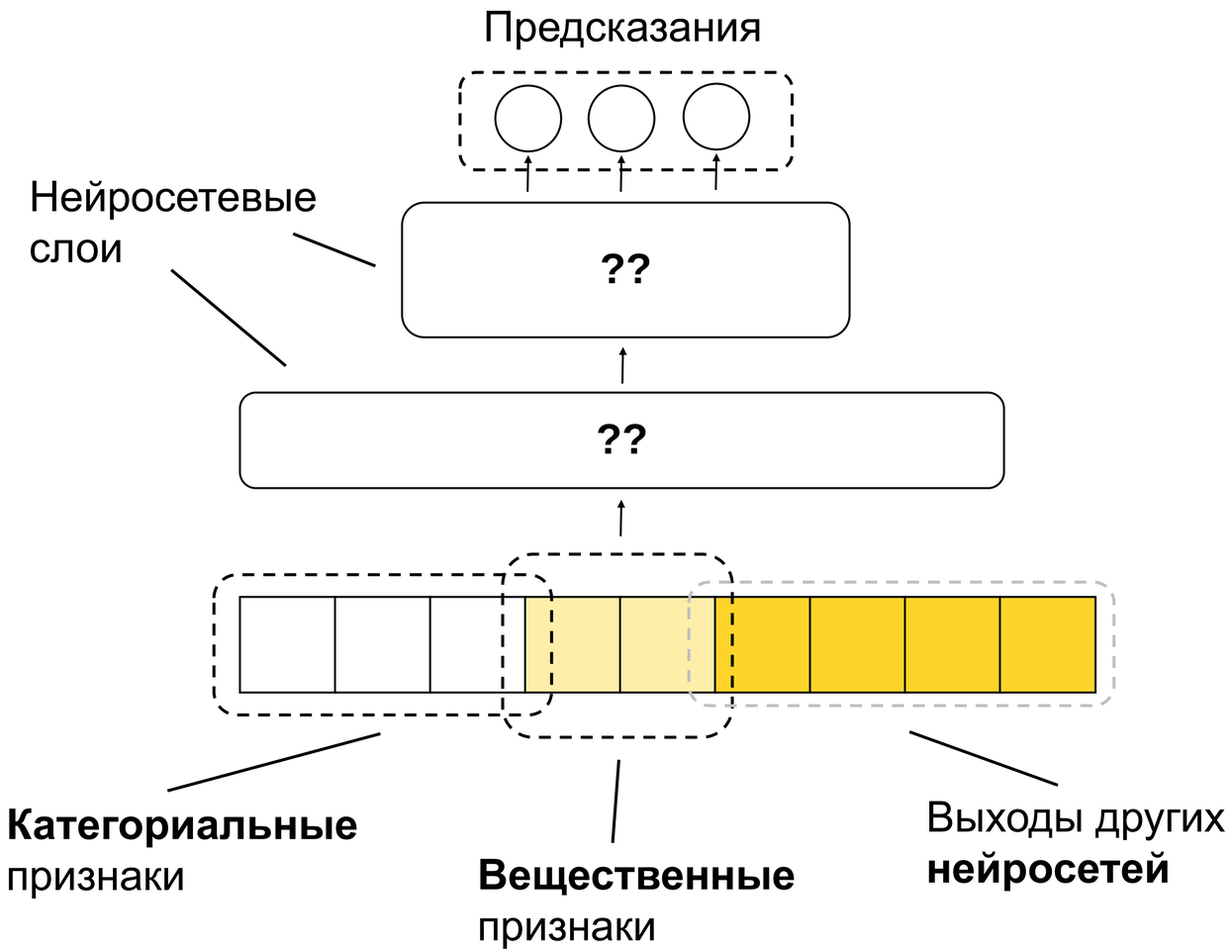
MLEnv & Pytorch

2014 2017 2018 2020 2021 2022 2022

12

Из выступления Evolution of Ads Ranking at Pinterest by Aayush Mudgal

Кодирование признаков



02

Категориальные признаки

Чуть-чуть терминологии

- **Айтем** – объект, который рекомендуем. Элемент (англ. item) каталога
- **Эмбеддинг** – почти синоним слова вектор. “Вкладываем” (англ. to embed) объекты в векторное пространство. Точки в этом пространстве – эмбеддинги
- **Импрешн** – объект, которые мы реально показали пользователю. Он оставил какой-то отпечаток, след, впечатление (англ. impression)
- **Обучаемый эмбеддинг** – когда в параметрах модели “вшиты” эмбеддинги для каких-то сущностей; альтернативный подход – какой-то отдельный энкодер
- **Айдишник** – почти синоним обучаемого эмбеддинга для объекта; ID-based представление; то есть используем для кодирования объекта только факт о том что это за объект

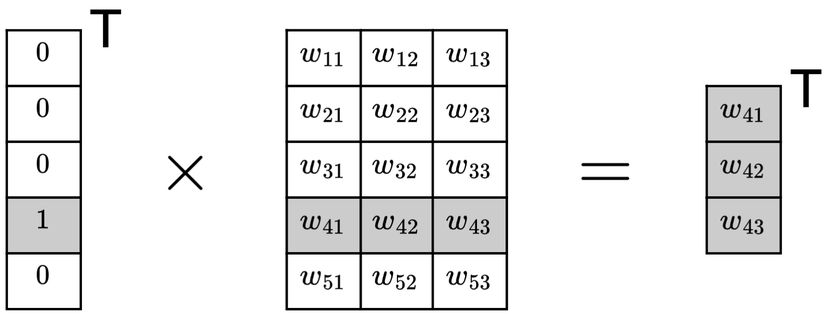
Категориальные признаки

- Самый простой способ закодировать категориальный признак – **one hot encoding** (один бит горит): $e = (0, \dots, 1, \dots, 0) \in \{0,1\}^n$

Что будет, если применить над таким вектором простейшую нейросеть – линейный слой? Пусть $W \in R^{\{n \times d\}}$. Тогда:

- $e_i W = \left(W_{ij}\right)_{j=1}^{d}$ – i-я строка матрицы W

Матрица эмбеддингов



То есть:

- W – **матрица эмбеддингов**, в которой каждому значению признака сопоставляется свой эмбеддинг
- $e_i W$ – операция **embedding lookup**, достающая нужный вектор из матрицы

Категориальные признаки

Пусть у нас есть несколько признаков, e.g. $e \in \{0,1\}^n, f \in \{0,1\}^m$.
Рассмотрим конкатенацию их **one hot** представлений $[e, f]$.

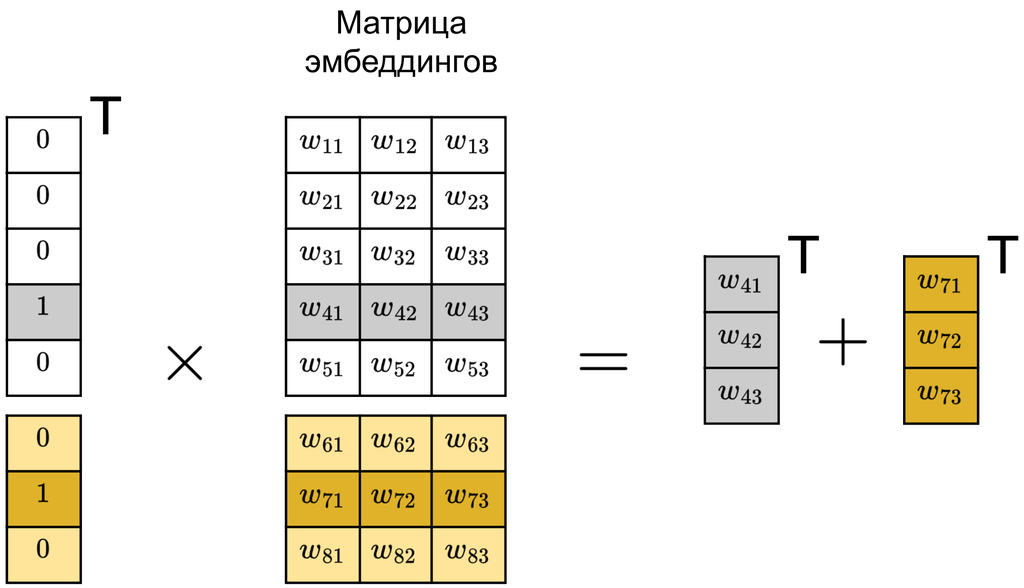
Пусть матрица $W \in R^{\{(n+m)\times d\}}$ – это конкатенация некоторых матриц $W_e \in R^{\{n\times d\}}$ и $W_f \in R^{\{m\times d\}}$. Тогда:

$$[e, f]W = eW_e + fW_f$$

То есть:

- W_e, W_f – **матрицы эмбеддингов** для признаков e и f
- Делается **embedding lookup** для каждого признака, затем эмбеддинги суммируются

Конкатенация эмбеддингов (вместо суммирования) позволяет использовать разные размерности эмбеддингов для разных признаков.

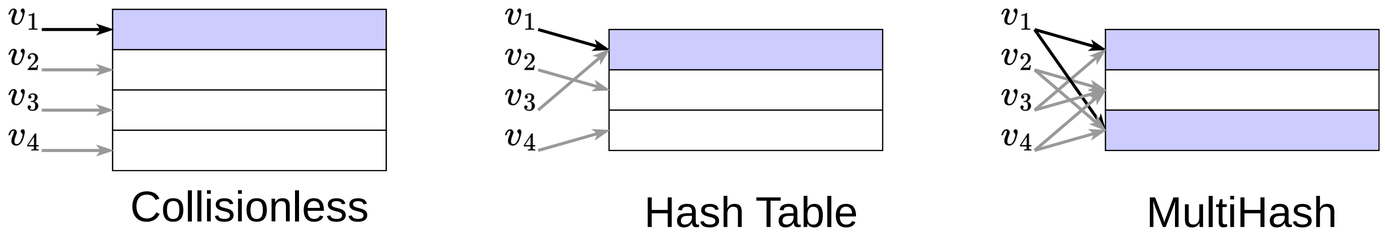


Гигантские матрицы эмбеддингов

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) – 9.5GB.

Потенциальные решения:

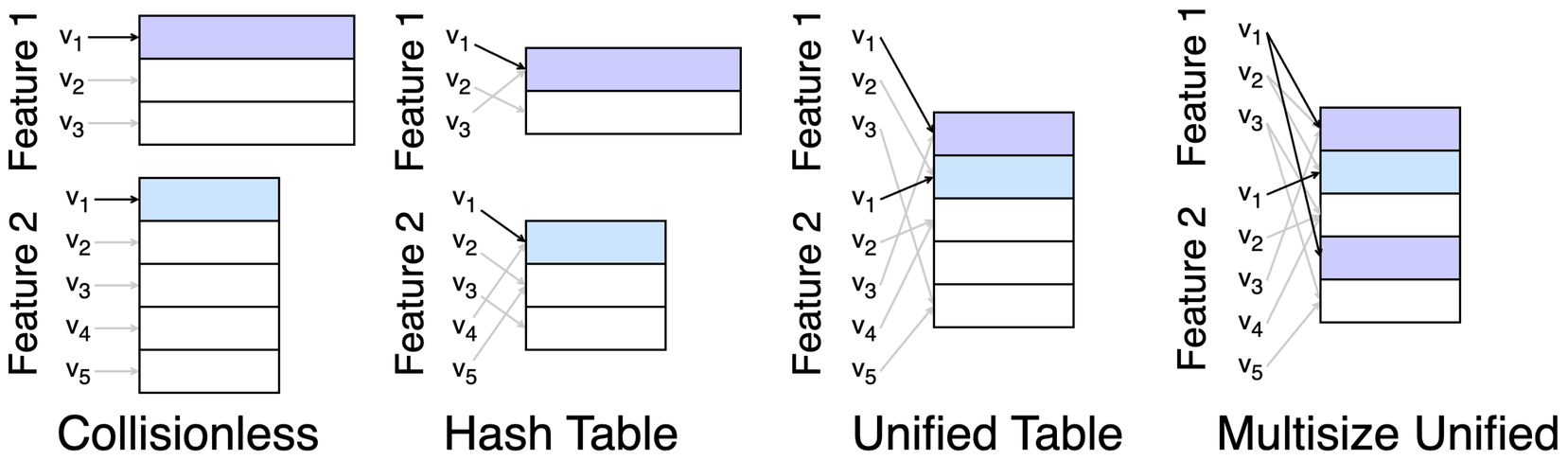
- продвинутая инфраструктура: CPU offloading, шардирование (**torchrec** [1])
- hashing trick



- vector quantization, semantic ids (обсудим в последней лекции курса)

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

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]

Размерности эмбеддингов

- Как подобрать размерность эмбеддингов для разных признаков?
- Использовать одинаковую
- Эвристики: $d = c\ log_2 n$; $\mathrm{d} = 6\sqrt[4]{n}$ (из DCN-v2)
- Регуляризация (разреживание)

- Про эмбеддинги можно думать в терминах передаваемой информации[1]:
- Эмбеддинг из **d** элементов, каждый из которых представлен **s** битами, может закодировать 2^{ds} значений. Пусть все значения равновероятны, тогда энтропия при передаче эмбеддинга равна $\mathrm{H}(v) = ds$.
- Для признака с n равновероятными значениями ($p_i = 1/n$) энтропия будет:

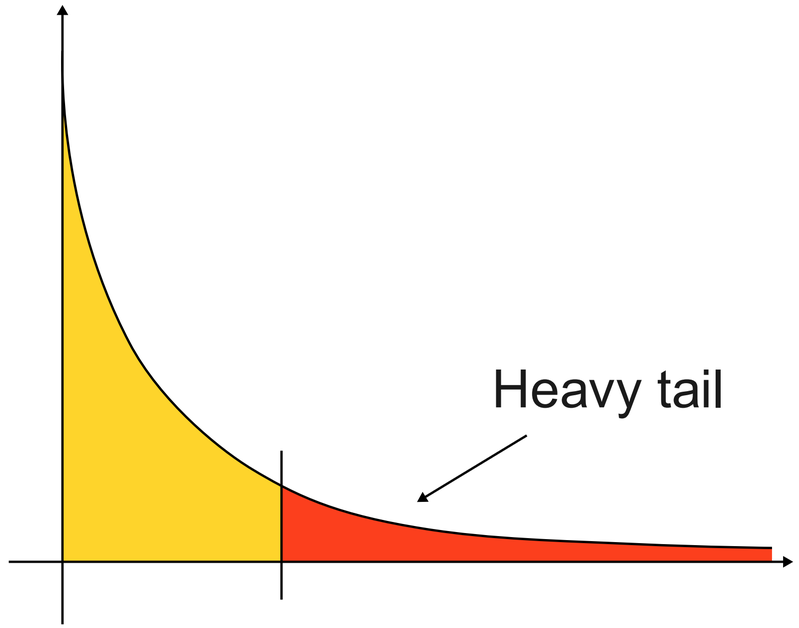
$$\mathrm{H}(e) = -\sum_{i=1}^{n} p_i\ log_2 p_i = log_2 n$$

1 On the Dimensionality of Embeddings for Sparse Features and Data

Переобучение

- Нейросети учатся как запоминать наблюдаемые данные (**меморизация**), так и находить обобщенные закономерности (**генерализация**)[1] .
- Распределения категориальных признаков имеют тяжелые хвосты (**long-tailed distributions**) – большая часть значений встречается очень редко.
- Для редких значений сложно вывести какие-то общие паттерны (генерализация), но их просто запомнить (меморизация).

… модель начинает переобучаться.



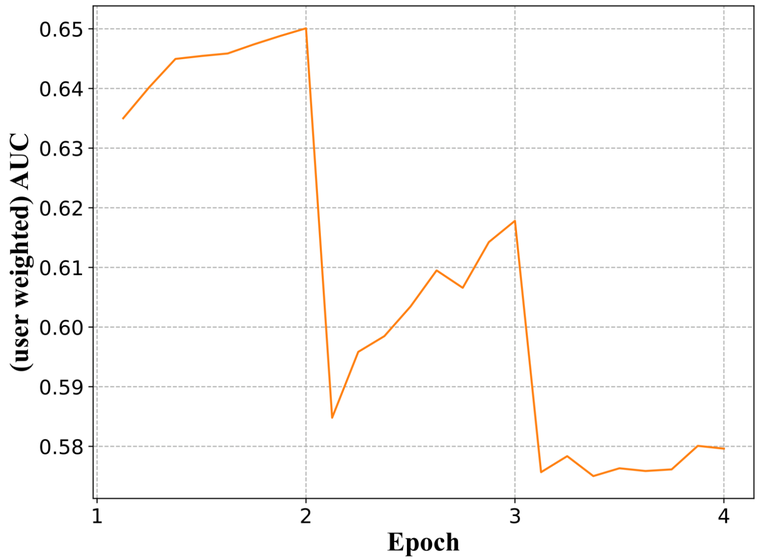
1 Wide & Deep Learning for Recommender Systems

Переобучение

Эмбеддинги, особенно для редких значений признаков, быстро “меморизируют” → переобучение за 1 эпоху.

Решения:

- Не использовать айдишники :)
- Регуляризация: L2, dropout, max-norm, freq-clipping
- Hashing trick (неявная регуляризация)
- Замораживать эмбеддинги после первой эпохи
- Сбрасывать матрицу эмбеддингов между эпохами



Из статьи Towards Understanding the Overfitting Phenomenon of Deep Click-Through Rate Prediction Models

Еще немного про эмбеддинги

Batch-lookup:

- Когда много категориальных признаков, приходится делать много мелких embedding lookup’ов → плохая утилизация GPU из-за kernel launch overhead
- Можно объединить все лукапы в один (пример: EmbeddingCollection в TorchRec)
- Хорошо комбинируется с unified embeddings подходом

Sparse gradients:

- У матрицы эмбеддингов на каждом батче очень разреженный градиент – не нужно делать полный allreduce при распределенном обучении

Rowwise adam[1]:

- Две статистики оптимизатора на весь эмбеддинг (вместо отдельных двух статистик на каждую координату эмбеддинга)

1 Training Highly Multiclass Classifiers

03

Вещественные признаки

Вещественные признаки

Вещественный признак as is использовать не получится, так как нейросети чувствительны к:

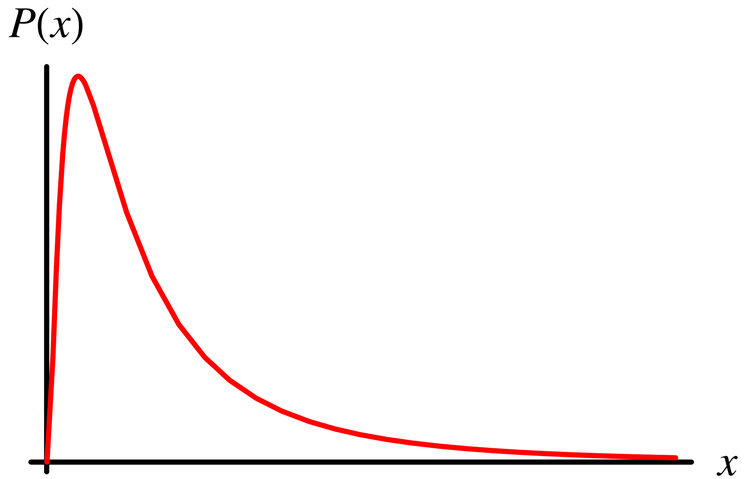
- **Выбросам**: лучше делать клиппинг
- **Масштабам**: нужно нормализовывать
- **NaN**: нужно обрабатывать пропуски (заполнение, отдельный эмбеддинг и т.д.)

… и этого может быть недостаточно.

Log1p

Простая и популярная трансформация:

- $x \to log(x + 1)$
- Уменьшает “скошенность” распределений (e.g., логнормальное распределение приводит к нормальному, а мы любим нормальные распределения в нейросетях!)
- “Сглаживает” разницу между значениями



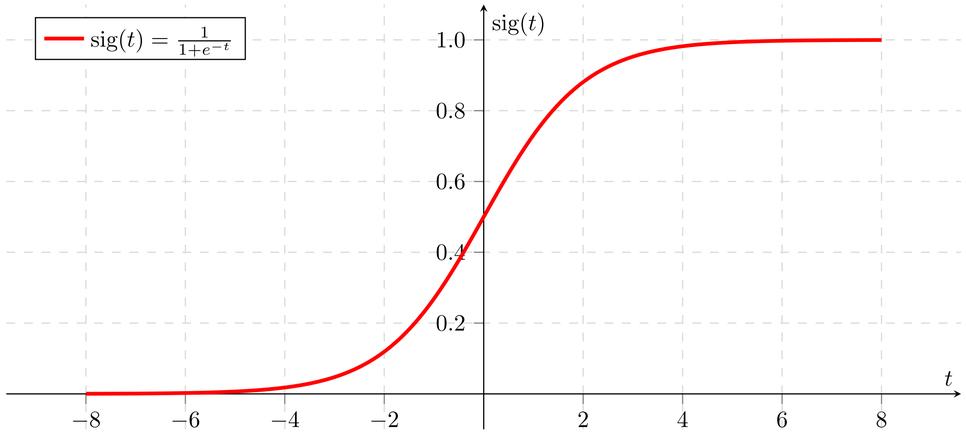
$$P(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left[-\frac{(\ln(x) - \mu)^2}{2\sigma^2}\right]$$

x	1	2	3	10	100	10000
log(x + 1)	0.69	1.1	1.39	2.39	4.61	9.21

Sigmoid Squashing

Альтернатива Log1p – прогон через сигмоиду:

- $x \to \sigma(ax + b)$
- Сильно сглаживает выбросы, не требует клиппинга
- Изначально сигмоиды часто использовали в качестве функций активаций
- Важно нормализовывать входы перед сигмоидами, чтобы не было затухающих градиентов (из-за попадающих на “края” сигмоиды значений)
- Можно использовать **несколько сигмоид** с разными параметрами (a, b)



Функция распределения

Заменим значение признака на оценку распределения:

- $x \to P(X \leq x)$
- Приводит признак в **равномерное** распределение на [0, 1]
- Устраняет скошенность, выбросы
- Реализуется через квантильную аппроксимацию эмпирической функции распределения

Периодические функции

Для выражения **периодических зависимостей**:

- $x \to [sin(2\pi c_1 x), cos(2\pi c_1 x), \dots, sin(2\pi c_n x), cos(2\pi c_n x)]$
- Коэффициенты c_i могут быть фиксированы или **обучаемы**
- Используют для обработки признаков, связанных с временем
- Что-то похожее делали для позиционных эмбеддингов в оригинальном трансформере

Квантование (бинаризация)

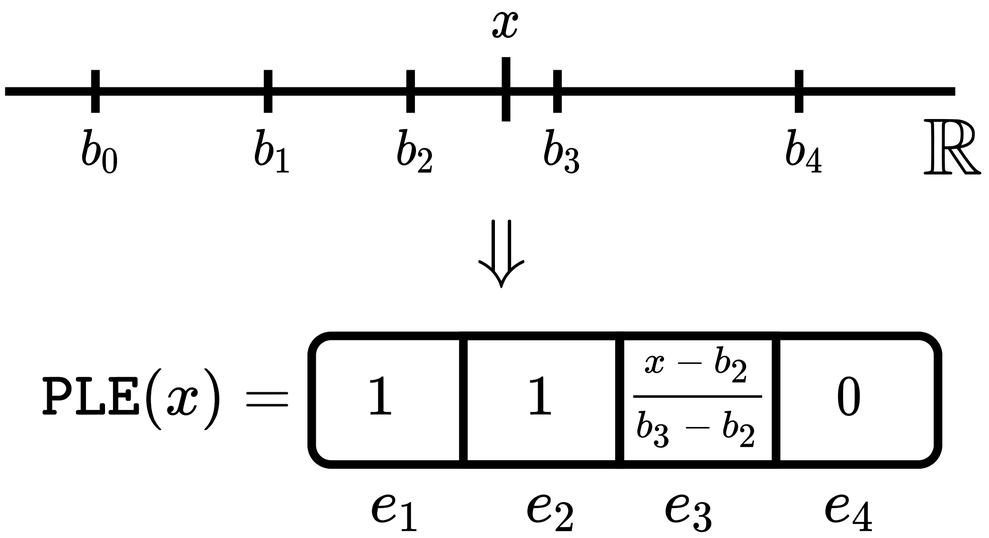
Можно превратить число в категорию:

- Квантование: разбиваем значения на **бины** по квантилям
- Каждому бину соответствует категория (номер бина)
- Это эквивалентно **кусочно-постоянному кодированию**

Кусочно-линейное кодирование (PLE)

$$x \to PLE(x)$$

- Кодируем не только номер бина, но и где внутри него находимся
- Сохраняем отношение порядка между бинами
- SOTA-подход



Из статьи On Embeddings for Numerical Features in Tabular Deep Learning

04

Feature interaction layers

Кросс-признаки

Пусть у нас есть линейная модель над конкатенацией one-hot представлений признаков:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i \; x_i$$

Её можно усложнить, добавив всевозможные одночлены второй степени:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i \; x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} \; x_i x_j$$

Добавили всевозможные кросс-признаки – комбинации исходных признаков (их декартовы произведения)

Проблемы:

- Квадратичный рост количества признаков $W \in R^{n \times n}$
- Разреженность: редкие комбинации, не пересекающиеся на train и test (e.g., user id × item id)

Факторизационные машины[1]

Введём матрицу $V \in R^{n \times d}$, и будем формировать W как $W = Upper(VV^T)$, то есть $w_{ij} = \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$. Тогда наша новая модель:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i \, x_i + \underbrace{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle \, x_i x_j}_{\text{Попарные взаимодействия признаков}}$$

- $\boldsymbol{v}_i \in R^d$ – эмбеддинг i-го признака
- $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \sum_{f=1}^{d} v_{i,f} v_{j,f}$ – скалярное произведение эмбеддингов

Решили обе проблемы:

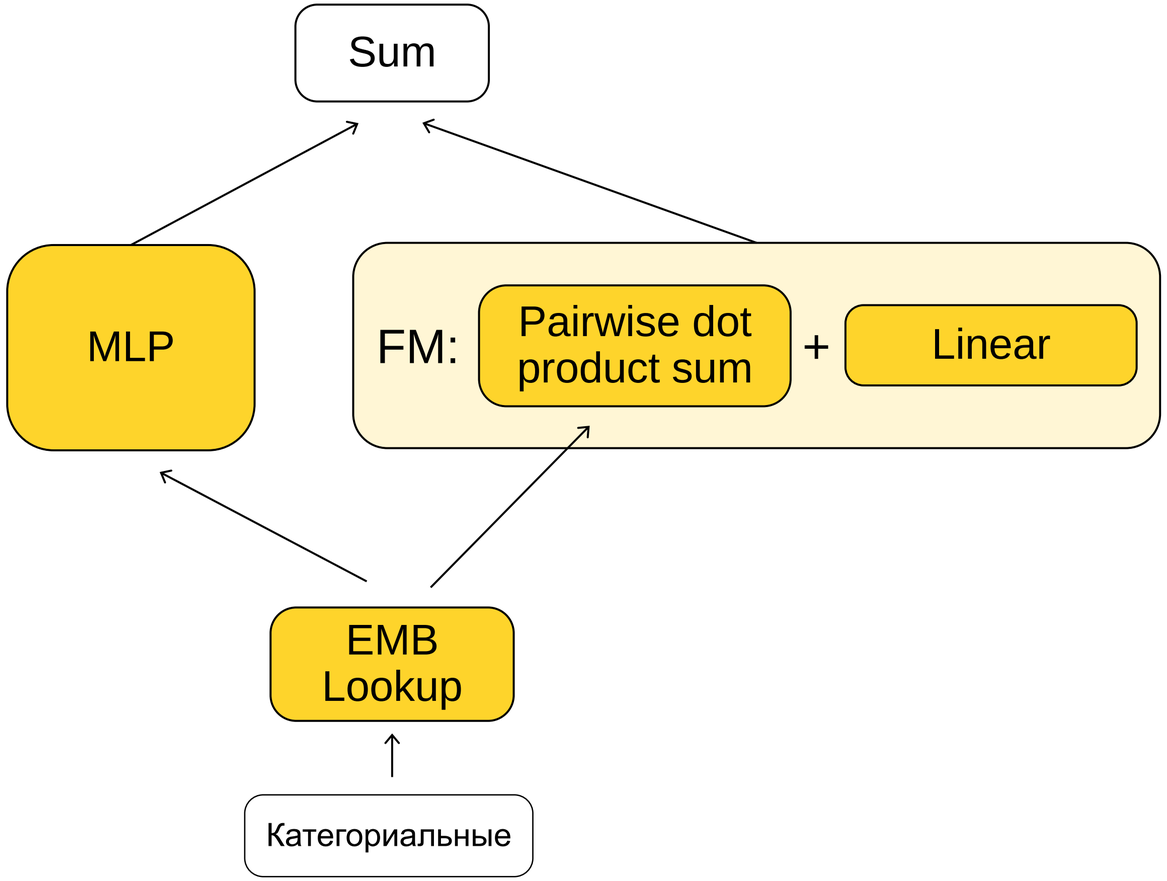
- Параметров $O(nd)$ вместо $O(n^2)$
- Обобщается на новые комбинации признаков

Улучшение: сделать отдельные эмбеддинги признаков для каждой пары[2].

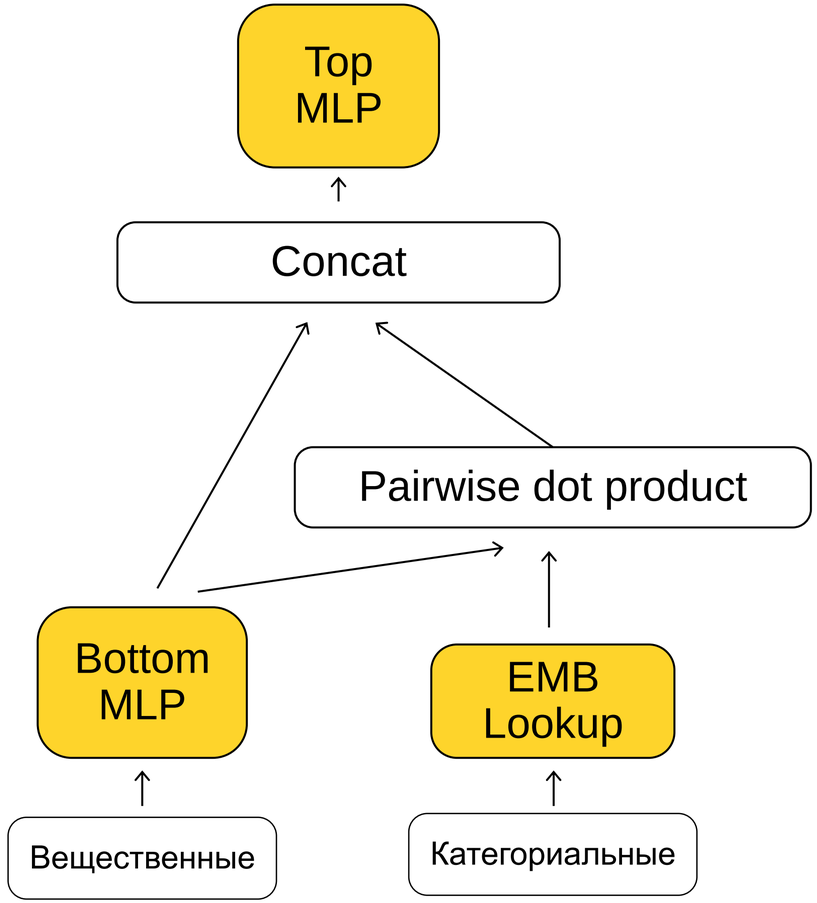
1 Factorization Machines
2 Field-aware Factorization Machines for CTR Prediction

FM × MLP

DeepFM[1]



DLRM[2]



1 DeepFM: A Factorization-Machine based Neural Network for CTR Prediction
2 Deep Learning Recommendation Model for Personalization and Recommendation Systems

MLP vs. Dot Product

> MLP над конкатенацией векторов не может выучить скалярное произведение![1]

Что нужно сделать: **поэлементно умножить** векторы, и потом уже подать в MLP. Тогда скалярное произведение воспроизводится тривиальным линейным слоем с единичками.

… как обобщить это на большое количество векторов?

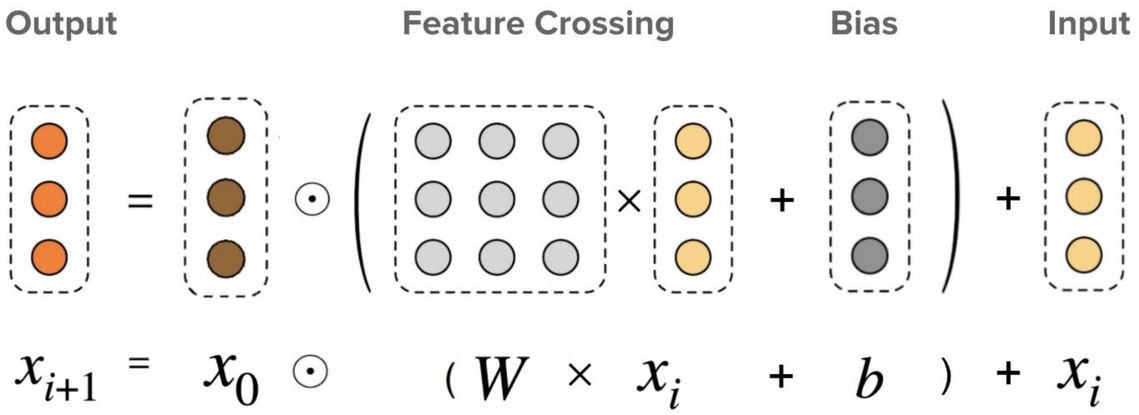
1 Neural Collaborative Filtering vs. Matrix Factorization Revisited

DCN-v2

Кросс-слой:

- $x_{l+1} = x_0 \odot (W_l x_l + b_l) + x_l$
- L слоев моделируют все возможные комбинации признаков степени L + 1
- Diminishing returns: максимум профита при 2-3 слоях

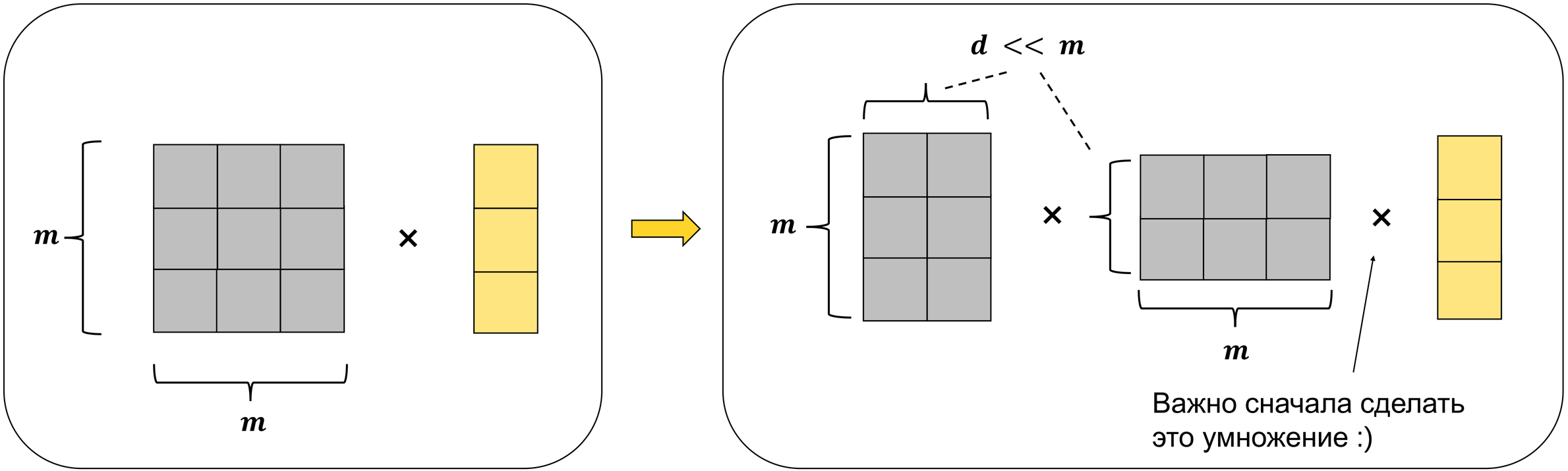
Проблема: он очень дорогой, так как работает над очень широкой конкатенацией векторов всех признаков



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

Низкоранговый DCN

- Убираем боттлнек в умножении на большую матрицу с помощью низкорангового разложения



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:

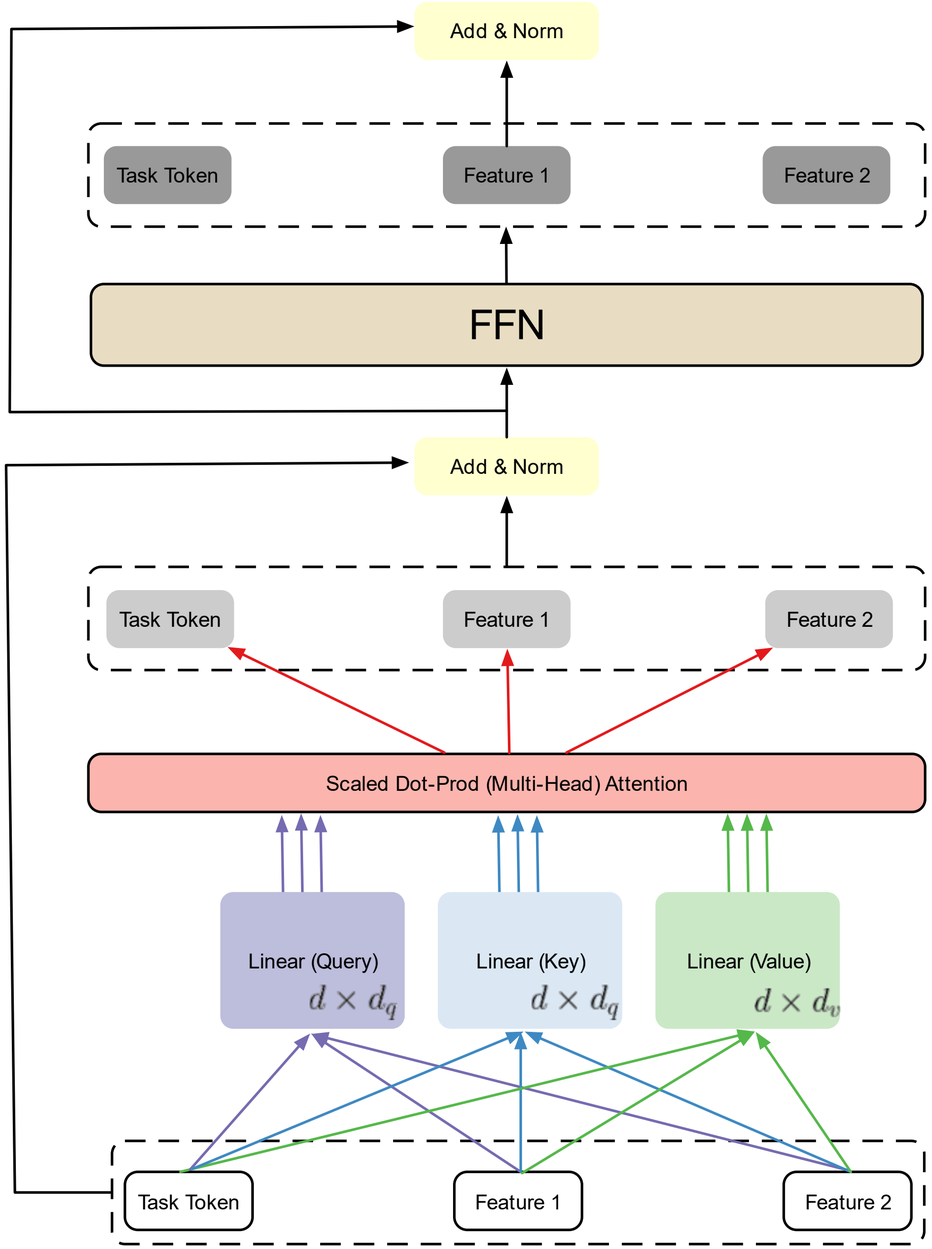
- MLP **неявно моделирует взаимодействия высокого порядка** между признаками
- На практике лучше сначала сделать DCN, потом MLP (параллельное применение работает хуже); перед MLP обычно сужаем вектор из DCN

Attention

Интуитивно кажется, что механизм внимания должен хорошо моделировать взаимодействие признаков.

AutoInt[1]:

- Получаем эмбеддинги признаков одинаковой размерности (для вещественных умножаем скаляр на обучаемый вектор)
- Применяем multi-head attention
- Конкатим все получившиеся векторы и делаем MLP



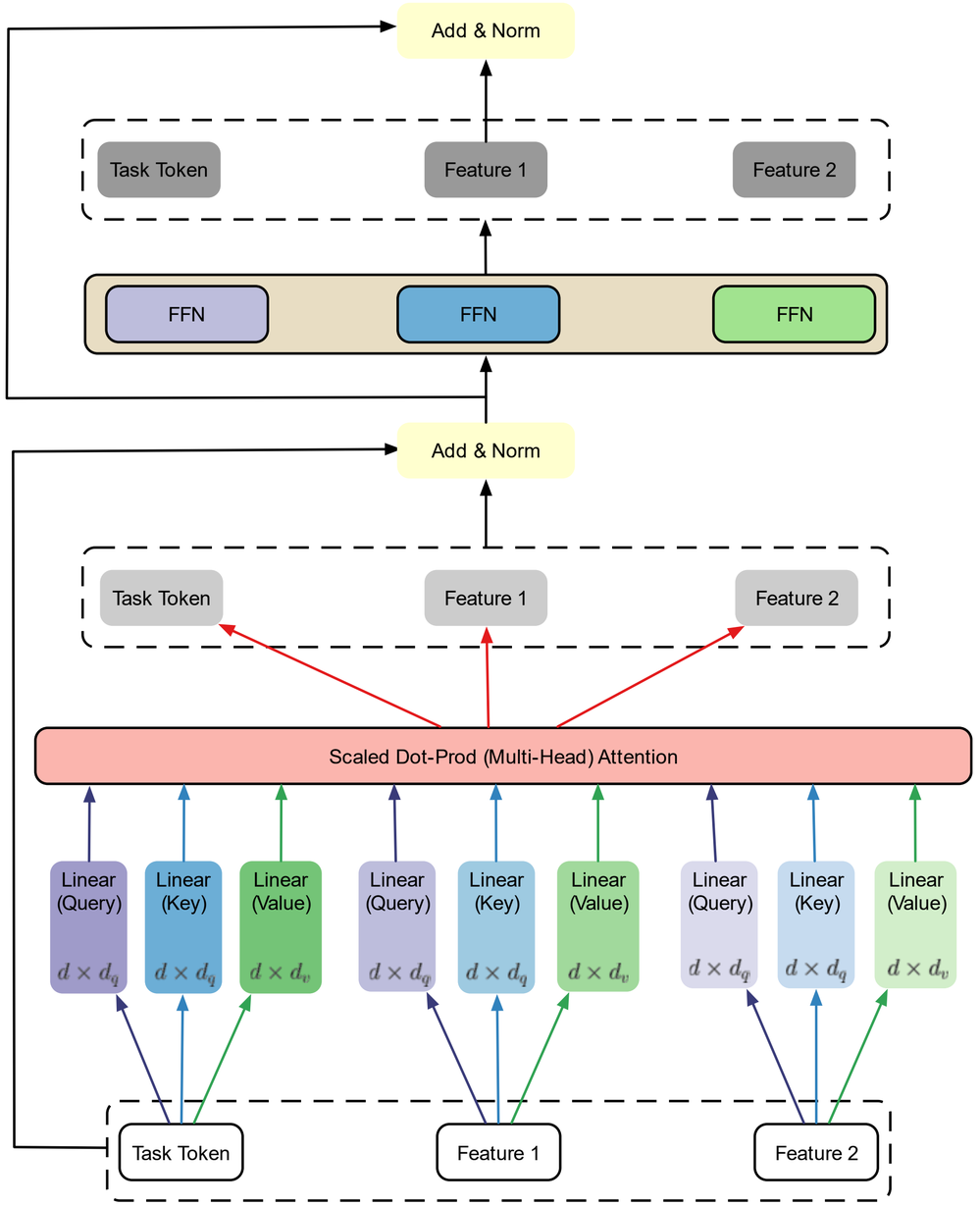
1 AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks

Attention

Hiformer[1]:

- Фичи гетерогенные, имеют разную структуру
- Обычный механизм внимания применяет одни и те же Q, K, V матрицы ко всем признакам
- Разные пары признаков нужно по-разному обрабатывать (field-awareness!)

… сделаем **гетерогенный аттеншн**: отдельные Q, K, V для каждого признака.



1 Hiformer: Heterogeneous Feature Interactions Learning with Transformers for Recommender Systems

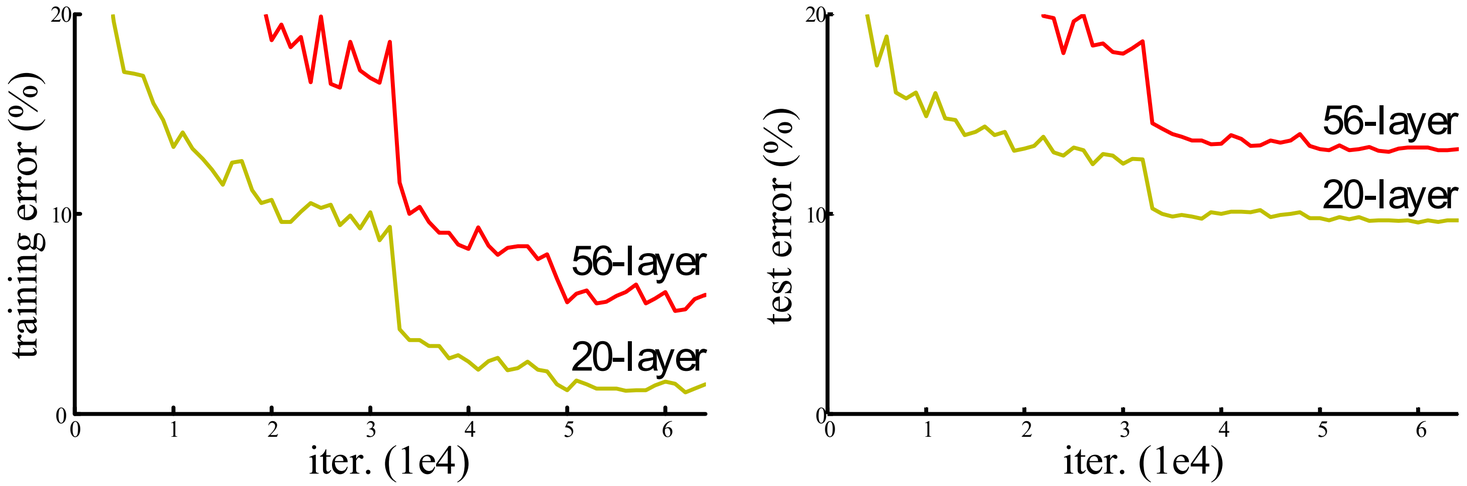
05

Is MLP enough?

Чем плох MLP?

MLP – полносвязная нейросеть из линейных слоев с нелинейностями in between (чаще всего ReLU):

- с увеличением количества слоев начинает хуже оптимизироваться
- Затухающие градиенты (помогает нормализация)



Из статьи Deep Residual Learning for Image Recognition

Чем плох MLP?

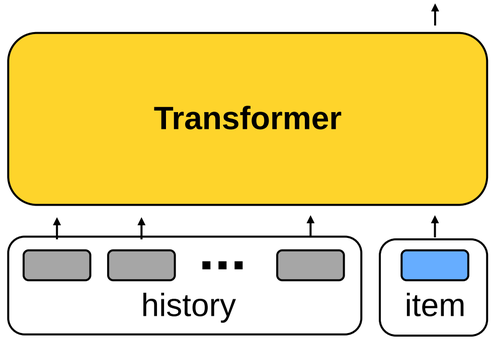
- История из практики:

- Нужно было переделать модель с ранним связыванием в двухбашенную

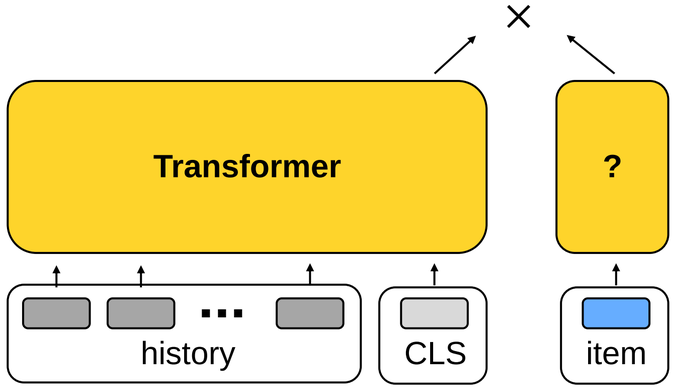
- Если использовать MLP над вектором item’а: сохраняем 50% профита

- Если применить трансформер над вектором item’а как над последовательностью из одного события: 80% профита

- Упрощенный трансформер над одним токеном вырождается в ResNet



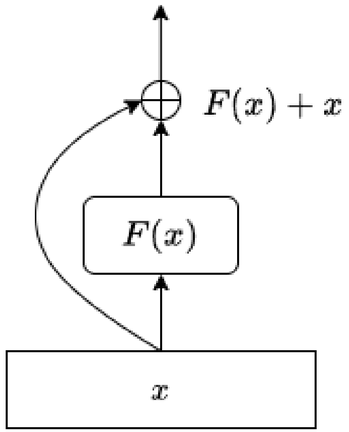
Early fusion



Two-tower model

Альтернативы MLP

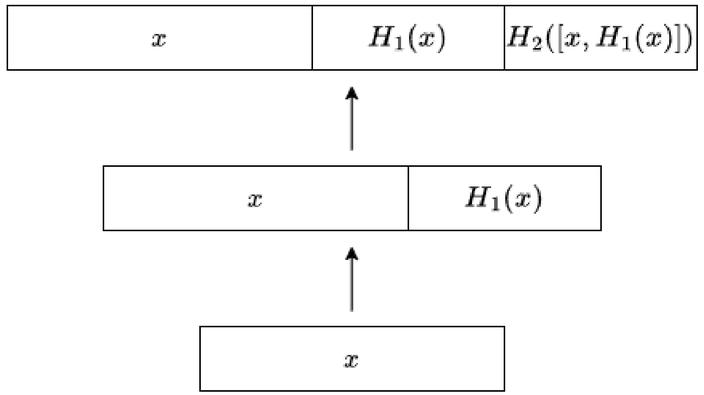
ResNet[1]



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

- 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

06

Многозадачность

Многозадачность

В рекомендациях почти всегда **много различных сигналов**, каждый из которых важно учитывать: клики, добавления в корзину, в избранное, заказы; прослушивания, скипы, лайки, дизлайки, добавления в плейлисты.

- Можно обучить отдельные модели под каждый сигнал
- Сделать синтетический таргет (лайк = 2, прослушивание = 1, скип = 0, дизлайк = -1)
- Взвешивание сэмплов по важности сигнала

Проблемы:

- Отдельные модели – дорого; некоторые сигналы невозможно отдельно выучить
- Парето-фронт качества предсказания сигналов сдвигается при каждом значительном изменении ранжирования, нужно переподбирать таргет / веса

Multi-task learning

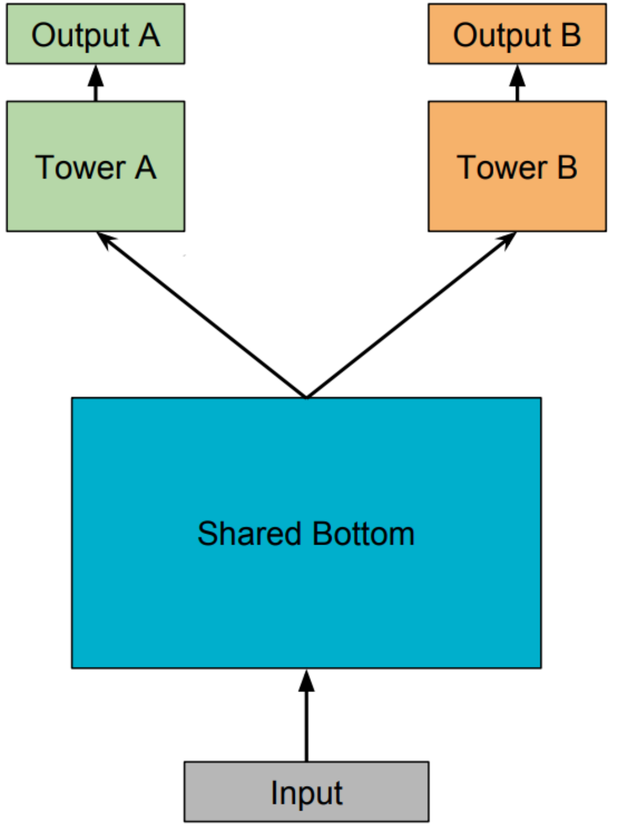
Разные задачи (сигналы) часто связаны друг с другом (e.g., покупка и клик – $P(\text{заказ} \mid \text{показ}) = P(\text{заказ} \mid \text{клик}) \cdot P(\text{клик} \mid \text{показ})$). Если модель хорошо предсказываем клики, это поможет предсказывать заказы.

Нейросети позволяют выучить **многоголовую** модель, предсказывающую сразу несколько сигналов:

- Есть общий backbone (shared слои)
- Есть отдельные головы (task-specific слои)
- Одна крайность – отдельные модели (все слои task-specific)
- Другая – вся сеть общая (backbone), а task-specific слои максимально простые, то есть линейные слои

Плюсы:

- Одна общая модель на все сигналы экономит ресурсы
- **Knowledge transfer**: улучшаем общее качество решения задач



Из статьи Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts

Negative Transfer

Negative transfer – начиная с какого-то момента обучения задачи тянут модель в разные стороны (градиенты по разным задачам не сонаправлены)

- **Task conflict** – ухудшаемся на одной задаче в пользу другой

Решения:

- Увеличить модель – чем больше емкость модели, тем проще избегать конфликтов
- Больше task-specific слоев
- **Mixture of experts**

Mixture of Experts

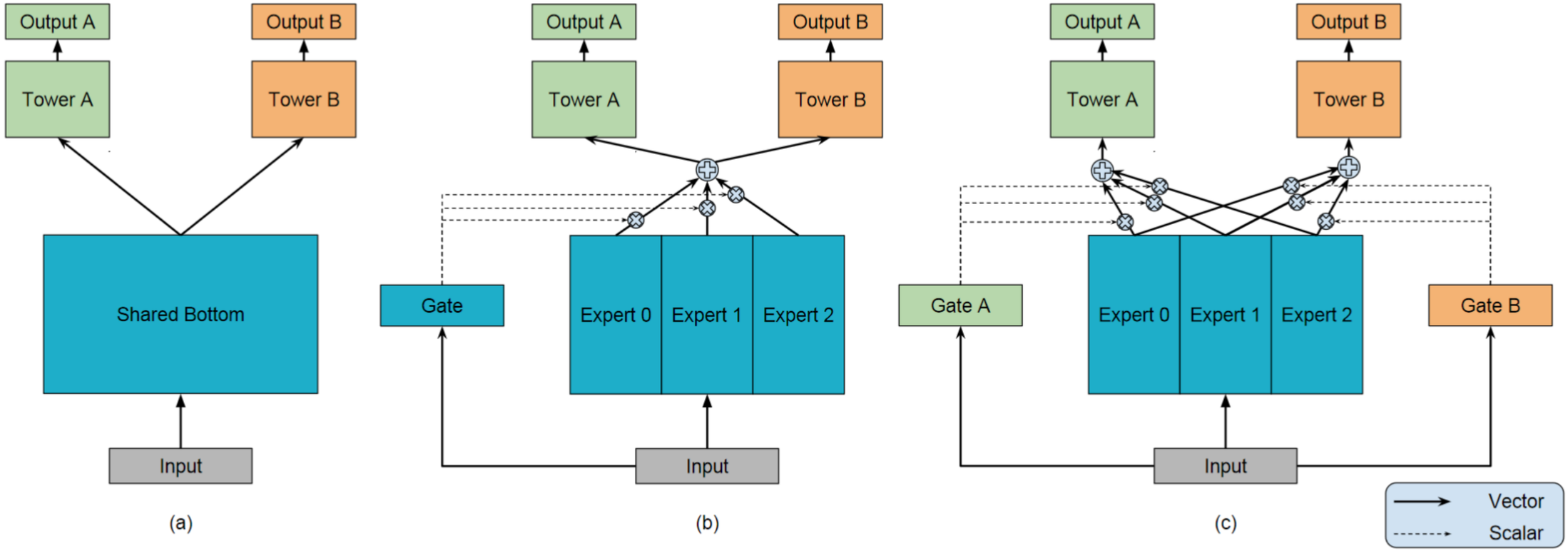
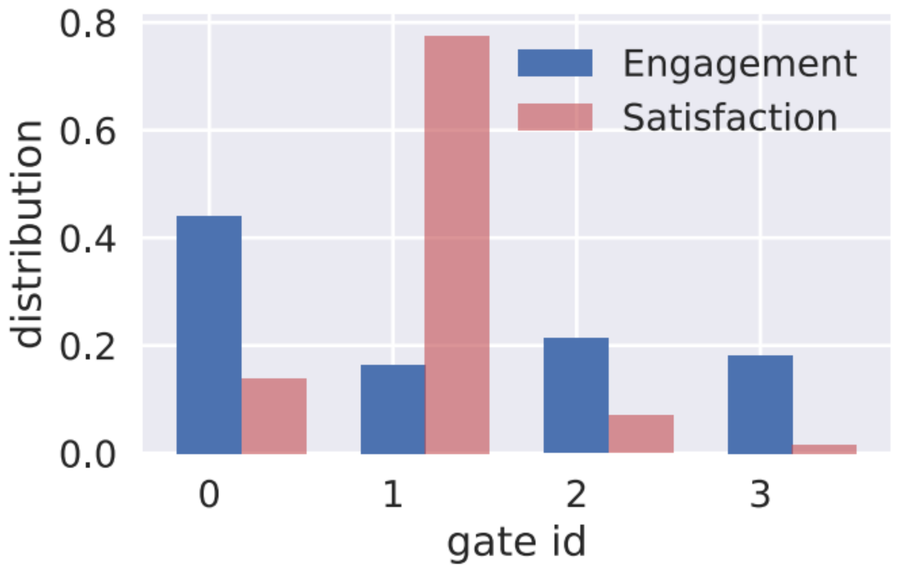


Figure 1: (a) Shared-Bottom model. (b) One-gate MoE model. (c) Multi-gate MoE model.

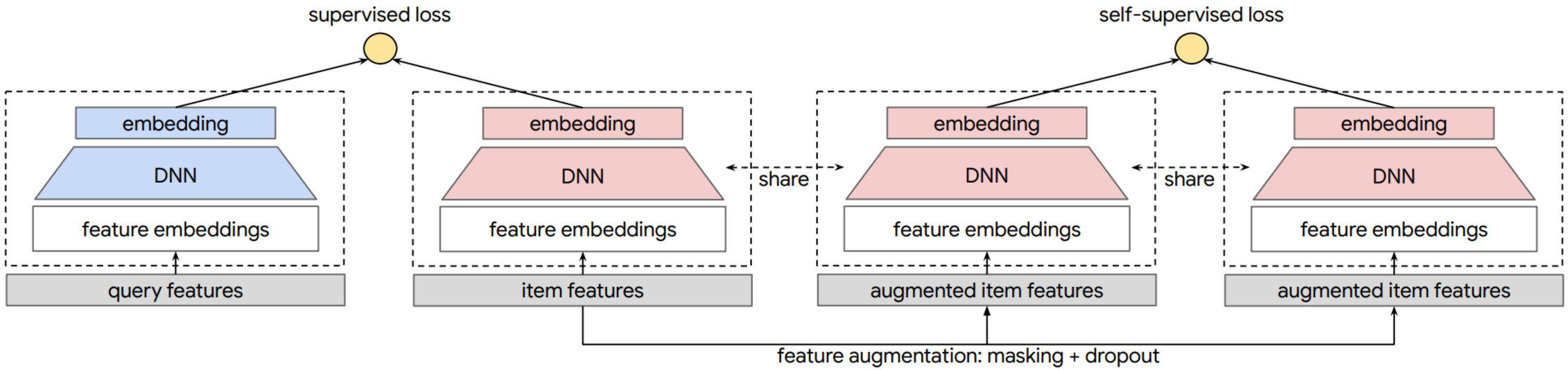
Из статьи Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts

Mixture of Experts



Из статьи Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts

Self-supervised Learning



Из статьи Self-supervised Learning for Large-scale Item Recommendations

Можно “портить” признаки на входе в модель в качестве аугментации и делать contrastive learning:

- Улучшаем качество на тяжелом хвосте
- Улучшаем обобщающую способность за счет регуляризации

Contrastive Pre-training

У Пинтереста добавление эмбеддингов по item ID приводило к переобучению ранжирующей модели после первой эпохи:

- У Пинтереста добавление эмбеддингов по item ID приводило к переобучению после первой эпохи
- Сделали отдельную стадию для предобучения ID-based эмбеддингов

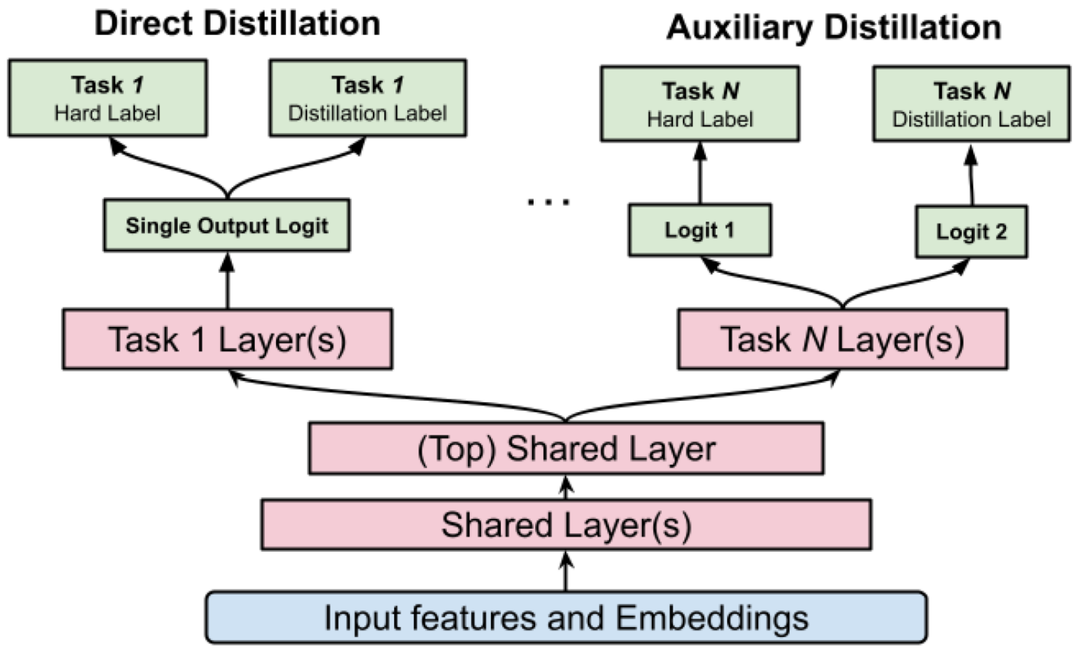
Table 1: Hit@3 performance across various training schema. Single-stage training is worse than the baseline due to overfitting, and two-stage strategies are always better. Additionally, fine-tuning performs better than freezing the embedding.

Pre-training Methods	Downstream Lift	
	Homefeed	Related Pins
Baseline	+0.000% ± 0.032%	+0.000% ± 0.066%
Single Stage	−3.347% ± 0.160%	−1.907% ± 0.116%
Two-stage Frozen	+1.157% ± 0.025%	+1.929% ± 0.070%
Two-stage Fine-tuned	**+1.323%** ± 0.048%	**+2.187%** ± 0.041%

Из статьи Taming the One-Epoch Phenomenon in Online Recommendation System by Two-stage Contrastive ID Pre-training

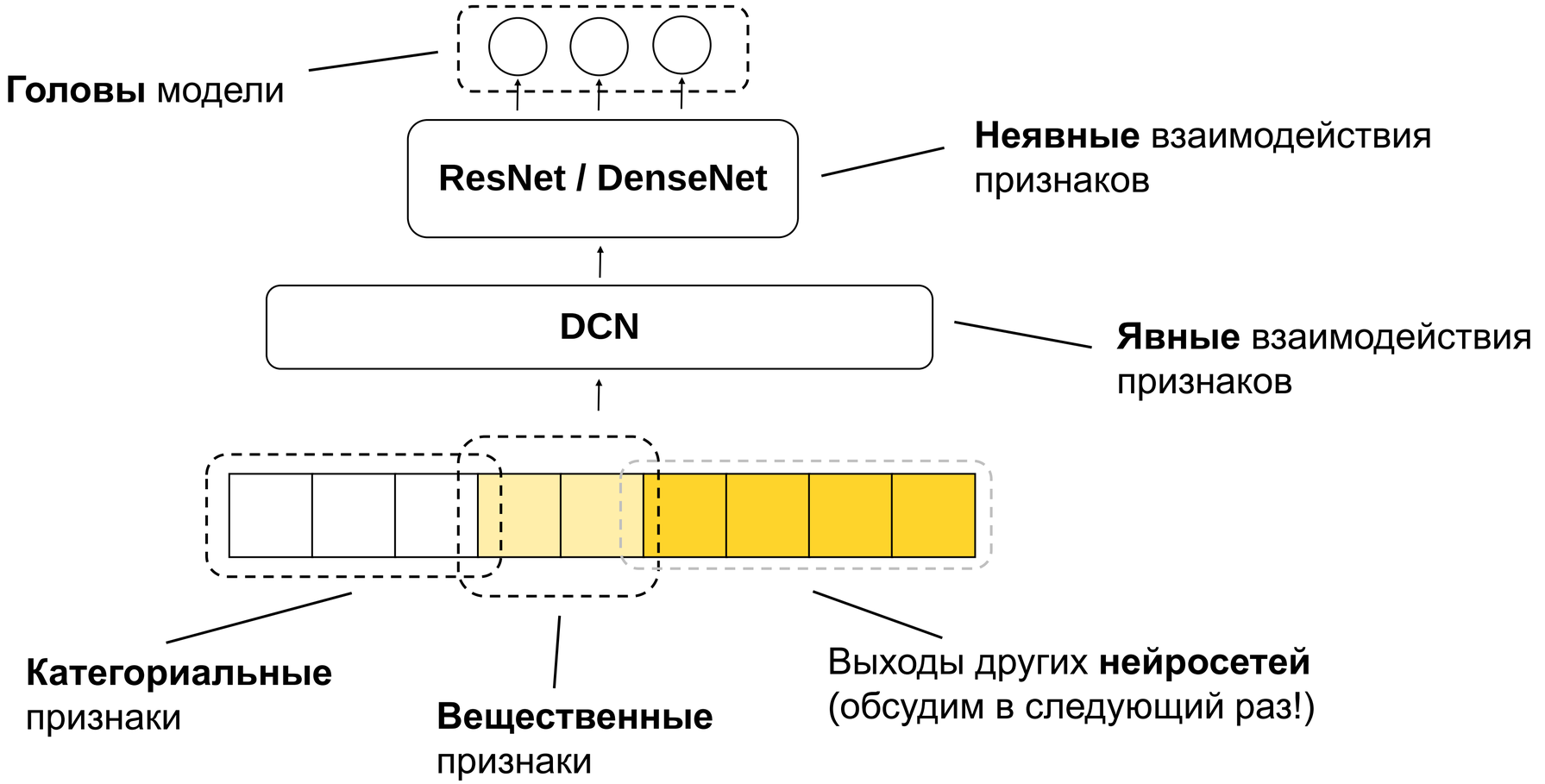
Knowledge Distillation

- Обучаем учителя (большую конфигурацию модели)
- Учим ученика на комбинацию истинных “меток” и предсказаний учителя
- **Auxilliary distillation:** Можно учить отдельную голову ученика на предсказания учителя



Из статьи Bridging the Gap: Unpacking the Hidden Challenges in Knowledge Distillation for Online Ranking Systems

Recap


Головы модели
ResNet / DenseNet
Неявные взаимодействия признаков
DCN
Явные взаимодействия признаков
Категориальные признаки
Вещественные признаки
Выходы других **нейросетей** (обсудим в следующий раз!)

В следующий раз

- Двухбашенные модели и как их обучать
- Всё про кодирование айтемов и пользователей (включая трансформеры)