Эволюция методов: от ARIMA и экспоненциального сглаживания до LSTM и трансформеров

Мурадян Денис Степанович

Санкт-Петербургский государственный университет
Математико-механический факультет
Искусственный интеллект и наука о данных

Бакалавриат, 3 курс

2025

Что такое временной ряд

- Последовательность наблюдений y_t, упорядоченных во времени
- Значение зависит от момента наблюдения t
- Примеры:
  - температура воздуха по дням
  - объём продаж по неделям
  - потребление электроэнергии по часам
  - колебания курсов валют

Добавим формализма: Постановка задачи прогнозирования

$$(y_t,\ t \in \mathbb{N}), \qquad \text{известно } y_1, y_2, \ldots, y_T$$

$$\hat{y}_{T+h} = f(y_1, y_2, \ldots, y_T, h), \quad h = 1, \ldots, H$$

- Требуется предсказать будущие значения ряда
- Горизонт прогноза H может быть различным
- Возможен прогноз с интервалом неопределённости

Неопределённость прогноза

$$(d_{T+h}, u_{T+h}), \qquad P(d_{T+h} \leq y_{T+h} \leq u_{T+h}) \geq \alpha$$

- Интервал содержит будущие значения с доверительной вероятностью α
- Учитывает влияние случайных факторов и ошибки модели

Пример и обозначение

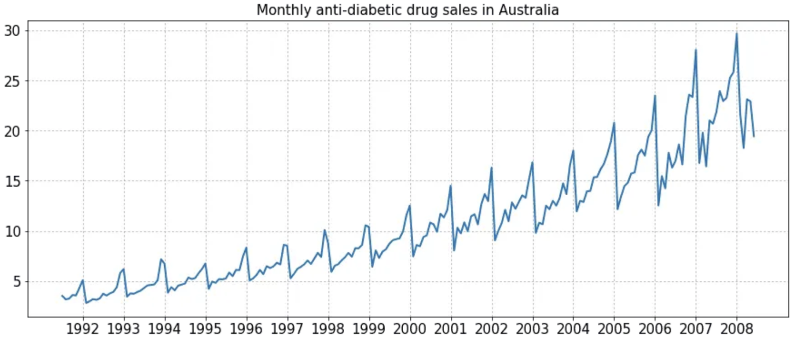
$$y_1, \ldots, y_{30} \Rightarrow \hat{y}_{31} = f(y_1, \ldots, y_{30}, 1), \quad \hat{y}_{35} = f(y_1, \ldots, y_{30}, 5)$$

После появления новых данных: $\hat{y}_{35} = f(y_1, \ldots, y_{33}, 2)$

$\hat{y}_{35|30}$ — прогноз на 35-й день, построенный на 30-й $\qquad$ $\hat{y}_{35|33}$ — уточнённый прогноз

Классические статистические подходы

- Идея: y_t зависит от прошлых значений $y_{t-1}, y_{t-2}, \ldots$
- Выделяем ключевые компоненты временного ряда:
  - уровень (долгосрочный средний)
  - тренд (направление изменения)
  - сезонность (повторяющиеся паттерны)
- Простые и интерпретируемые модели



Скользящее среднее

$$\tilde{y}_t = \frac{1}{k} \sum_{i=0}^{k-1} y_{t-i}$$

- Сглаживает шум и выделяет уровень ряда
- k — длина окна недавней истории
- Малое k: быстрая реакция, больше шум
- Большое k: плавно, но с запаздыванием

Простое экспоненциальное сглаживание (SES)

$$\hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\,\hat{y}_{t|t-1}, \qquad \alpha \in (0,1)$$

- Недавние значения имеют больший вес
- α — параметр памяти:
  - α ближе к 1 $\to$ быстрые изменения, меньше сглаживание
  - α ближе к 0 $\to$ сильное сглаживание, запаздывание
- Для рядов без выраженного тренда

Модель Холта: добавляем тренд

$$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$$
$$b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1}$$
$$\hat{y}_{t+h|t} = \ell_t + h\,b_t$$

- ℓ_t — оценка уровня ряда
- b_t — оценка тренда (скорости изменения)
- $\alpha, \beta \in (0, 1)$ — коэффициенты сглаживания
- Значения параметров подбирают по данным (минимизация ошибки прогноза)

Хольта–Уинтерса: аддитивная сезонность

$$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$$

$$b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1}$$

$$s_t = \gamma(y_t - \ell_t) + (1-\gamma)s_{t-m}$$

$$\hat{y}_{t+h|t} = \ell_t + h\,b_t + s_{t+m_h}, \quad m_h = ((h-1) \bmod m) + 1$$

- ℓ_t — уровень, b_t — тренд, s_t — сезонность
- α, β, γ — коэффициенты сглаживания $(0, 1)$
- Сезонные колебания примерно одинаковой величины

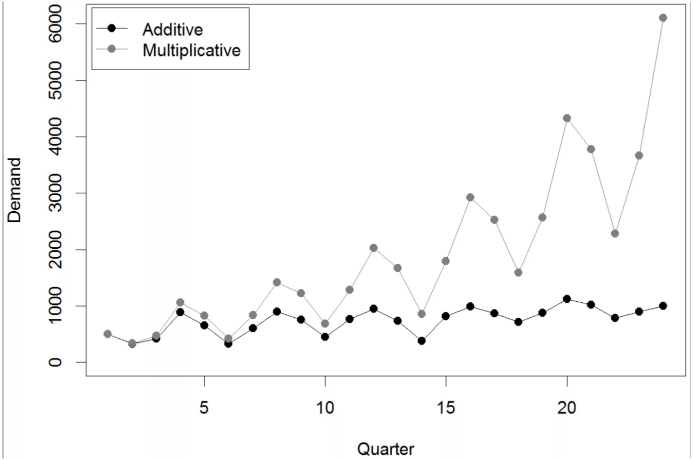
Хольта–Уинтерса: мультипликативная сезонность

$$
\begin{aligned}
\ell_t &= \alpha \frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
b_t &= \beta(\ell_t - \ell_{t-1}) + (1-\beta) b_{t-1} \\
s_t &= \gamma \frac{y_t}{\ell_t} + (1-\gamma) s_{t-m} \\
\hat{y}_{t+h|t} &= (\ell_t + h\, b_t)\, s_{t+m_h}
\end{aligned}
$$

- Амплитуда сезонности растёт вместе с уровнем
- Используется для пропорциональных колебаний

Как выбрать модель сезонности?

- **Аддитивная**:
  - сезонные колебания примерно постоянны по величине
- **Мультипликативная**:
  - амплитуда возрастает или уменьшается вместе с уровнем



Авторегрессионные модели: идея и маршрут

- Цель: формально описать зависимость y_t от прошлого и шумов
- База: лаги, ACF/PACF, стационарность и преобразования
- Семейства: AR, MA, ARMA, ARIMA; сезонные/расширенные: SARIMA, ARIMAX

Лаги и лаговый оператор

$$Ly_t = y_{t-1}, \qquad L^k y_t = y_{t-k}$$

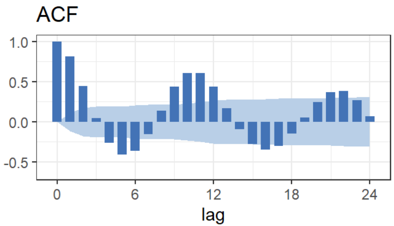
$$(1-L)y_t = y_t - y_{t-1}, \qquad (1-L^m)y_t = y_t - y_{t-m}$$

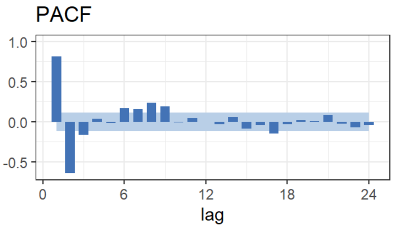
- Лаг τ — сдвиг на τ шагов назад
- $(1-L)$ — обычная разность (убирает тренд), $(1-L^m)$ — сезонная разность с периодом m

ACF и PACF: что смотреть перед моделированием

$$r_\tau = \frac{\sum_{t=1}^{T-\tau}(y_t - \overline{y})(y_{t+\tau} - \overline{y})}{\sum_{t=1}^{T}(y_t - \overline{y})^2}, \qquad \overline{y} = \frac{1}{T}\sum_{t=1}^{T} y_t$$

- r_τ — автокорреляция на лаге τ; $\overline{y}$ — среднее по выборке
- Пики на лагах $m, 2m, \dots \Rightarrow$ сезонность с периодом m
- Шаблоны: PACF «обрывается» для AR(p); ACF «обрывается» для MA(q)





Стационарность: интуиция и два уровня

- Интуитивно: статистические свойства не меняются при сдвиге по времени
- **Узкий (строгий) смысл:** совместные распределения инвариантны к сдвигу
- **Широкий смысл:** $\mathbb{E}[y_t] = \mu$, $\mathrm{Var}(y_t) < \infty$,
  $\mathrm{cov}(y_{t+\tau}, y_{s+\tau}) = \mathrm{cov}(y_t, y_s)$ (зависит только от лага)

Стационарность: пример и типичные нарушения

Пример (широкая, но не узкая): $y_t = \xi_1 \cos t + \xi_2 \sin t$, где $\xi_1, \xi_2 \in \{\pm 1\}$ независимы.
$\mathbb{E}[y_t] = 0$, $\operatorname{cov}(y_t, y_s) = \cos(t-s)$, но распределения y_0 и $y_{\pi/4}$ различны.

Типовые нарушения:

- *Случайное блуждание:* $y_t = y_{t-1} + \varepsilon_t$ (дисперсия растёт)
- *Линейный тренд:* $y_t = \alpha + \beta t + \varepsilon_t$ (среднее меняется)
- *Чистая сезонность:* $y_t = \sin t + \varepsilon_t$ (среднее периодично)

Приведение к стационарности: масштаб и разности

$$z_t = \begin{cases} \dfrac{y_t^{\lambda} - 1}{\lambda}, & \lambda \neq 0, \\ \ln y_t, & \lambda = 0 \end{cases} \quad \text{— преобразование Бокса–Кокса}$$

$$(1 - L)y_t = y_t - y_{t-1}, \qquad (1 - L^m)y_t = y_t - y_{t-m}$$

- λ — параметр Box–Cox; при $\lambda = 0$ используется логарифм
- Практика: сначала $(1 - L^m)$, затем при необходимости $(1 - L)$ до приемлемой стационарности

AR(p): авторегрессия p-го порядка

$$y_t = \alpha + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

- ε_t — белый шум (нуль-среднее, постоянная дисперсия, некоррелированность)
- AR(1): $y_t = \alpha + \phi y_{t-1} + \varepsilon_t$; если $|\phi| < 1$, то стационарно; $r_h = \phi^h$

MA(q): скользящее среднее q-го порядка

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

- Память по прошлым возмущениям ε_{t-i} — влияние постепенно затухает
- Параметры θ_i отражают силу и длительность «эхо» прошлых случайных колебаний

ARMA(p, q): объединяем AR и MA

$$y_t = \alpha + \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q}$$

$$a(L) y_t = \alpha + b(L)\varepsilon_t, \quad a(L) = 1 - \phi_1 L - \dots - \phi_p L^p, \;\; b(L) = 1 + \theta_1 L + \dots + \theta_q L^q$$

ARIMA(p, d, q): работаем с нестационарностью

$$(1 - L)^d y_t = y_t - y_{t-1}$$

$$a(L)(1 - L)^d y_t = \alpha + b(L)\varepsilon_t$$

- d — порядок дифференцирования (сколько раз берём разность) перед применением ARMA

SARIMA: сезонная ARIMA

$$\Phi(L^m)\,\phi(L)\,(1-L)^d(1-L^m)^D y_t = \Theta(L^m)\,\theta(L)\,\varepsilon_t + c$$

- (p, d, q) — несезонные порядки; $(P, D, Q)_m$ — сезонные; m — длина периода
- Учитываются краткосрочные лаги и сезонные лаги $m, 2m, \ldots$

ARIMAX / SARIMAX: учитываем внешние факторы

$$(1 - L)^d y_t = \mu + \sum_{i=1}^{n} \beta_i x_{t,i} + \frac{b(L)}{a(L)} \varepsilon_t$$

- $x_{t,i}$ — известные на момент прогноза факторы (например, календарь, температура)
- Внешние драйверы помогают снизить остаточную ошибку прогноза

Итоги блока: ARIMA-семейство

- **AR/MA/ARMA**: стационарные зависимости по лагах и шумам
- **ARIMA**: разности + ARMA для нестационарных рядов
- **SARIMA**: явная сезонность; **ARIMAX**: учёт известных факторов

От классики к RNN: идея рекуррентности

- В отличие от обычной (прямой) сети, рекуррентная сеть хранит **скрытое состояние** h_t
- На каждом шаге учитываются и **текущий вход** x_t, и **память** h_{t-1}
- Типичная запись: $h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \quad \hat{y}_t = g(W_{hy}h_t + b_y)$
- **Проблема:** при длинных последовательностях градиент *затухает/взрывается* $\Rightarrow$ новая информация слабо влияет

LSTM: как стабилизировать память на длинных шагах

- Идея: **вентили** управляют потоком информации
- Компоненты: забывание f_t, вход i_t, выход o_t, состояние ячейки C_t
- Ключевая «магистраль» памяти:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

- Эффект: градиент течёт по почти линейному пути через $C_t \Rightarrow$ меньше затухания/взрыва

Детали пропускаем: коллега уже разбирал математику LSTM на прошлом докладе.

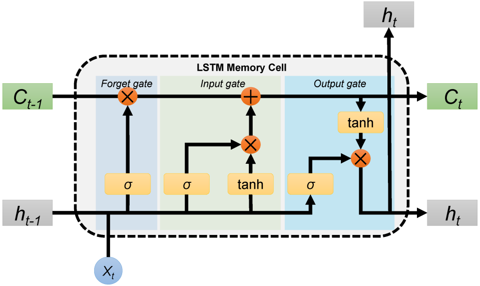


Схема LSTM-ячейки

GRU: упрощённая альтернатива LSTM (порядок вычислений)

$$
\begin{aligned}
\textbf{(1) Вентиль обновления:} \quad & z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
\textbf{(2) Вентиль сброса:} \quad & r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\textbf{(3) Кандидат состояния:} \quad & \tilde{h}_t = \tanh\big(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h\big) \\
\textbf{(4) Обновление памяти:} \quad & h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

- Нет отдельного C_t: память и выход объединены в h_t
- **Хронология шага t:** сначала $z_t, r_t \rightarrow$ затем $\tilde{h}_t \rightarrow$ затем h_t
- Интуиция: r_t дозирует влияние прошлого в кандидате, z_t решает, *насколько* обновлять состояние

От RNN к Attention: обработка и генерация последовательностей

- Рекуррентные сети хорошо моделируют временные зависимости, но выдают выход фиксированной длины.
- Во многих задачах нужно предсказывать **последовательности произвольной длины** — например, в машинном переводе, где длина перевода не совпадает с длиной исходной фразы.
- Возникает архитектура **Sequence-to-Sequence (Seq2Seq)**:
  - **Энкодер** считывает входную последовательность и преобразует её в векторное представление — «контекст».
  - **Декодер** шаг за шагом генерирует выходную последовательность, опираясь на этот контекст.
- Проблема: один фиксированный вектор не способен вместить всю информацию о длинной последовательности.
- Решение — **механизм внимания (Attention)**, который позволяет декодеру выбирать, на какие части входа смотреть в каждый момент.

Механизм внимания (Attention)

Идея: на каждом шаге декодер сам решает, *на какие части входа смотреть*, формируя индивидуальный контекст.

- Пусть энкодер выдал скрытые состояния: $H = (h_1, \ldots, h_T)$
- На i-м шаге декодер имеет своё состояние s_i
- Сравниваем s_i с каждым h_j и считаем **оценки внимания**:

$$e_{i,j} = \text{score}(s_i, h_j)$$

- Считаем веса через softmax:

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_k \exp(e_{i,k})}$$

Механизм внимания (Attention)

- Формируем **контекстный вектор** — взвешенную сумму по входным состояниям:

$$a_i = \sum_{j=1}^{T} \alpha_{i,j} h_j$$

- Контекст a_i используется декодером для генерации очередного токена y_i

Интерпретация: $\alpha_{i,j}$ показывает, насколько важна позиция j входа при генерации i-го выхода. Механизм внимания снимает ограничение на один фиксированный контекст и даёт модели возможность динамически фокусироваться на релевантных фрагментах входной последовательности.

Self-Attention: идея и формирование Q, K, V

Мотивация. Классический механизм внимания связывает энкодер и декодер, но внутри одной последовательности токены тоже зависят друг от друга.
Self-Attention позволяет каждому токену смотреть на другие токены в том же предложении и формировать контекст, учитывающий всю последовательность.

1. Вход. Пусть есть предложение: «кошка сидит на столе». Каждый токен преобразуется в embedding $x_i \in \mathbb{R}^d$. Все векторы объединяются в матрицу:

$$X = [x_0, x_1, \dots, x_n] \in \mathbb{R}^{(n+1)\times d}.$$

2. Формируем три набора векторов: Queries, Keys, Values.

$$Q_i = x_i W_Q, \quad K_i = x_i W_K, \quad V_i = x_i W_V,$$

где $W_Q, W_K, W_V \in \mathbb{R}^{d\times d}$ — обучаемые матрицы.

Self-Attention: вычисление внимания и контекста

Интерпретация:

- Q — «от кого идёт внимание» (что ищет токен),
- K — «на кого направлено внимание»,
- V — «какая информация передаётся».

На этом этапе каждый токен получает три разные роли, хотя исходный embedding был один.

3. Вычисляем оценки внимания (scores). Для токена i сравниваем его Q_i со всеми K_j:

$$\text{score}_{ij} = Q_i \cdot K_j.$$

Так получаем вектор оценок — насколько каждый токен j важен для токена i.

4. Преобразуем оценки в веса.

$$\alpha_{ij} = \text{softmax}(\text{score}_{ij})$$

Теперь для токена i мы знаем, с какой важностью учитывать каждый другой токен.

Self-Attention: вычисление внимания и контекста

4. Преобразуем оценки в веса.

$$\alpha_{ij} = \mathrm{softmax}(\mathrm{score}_{ij})$$

Теперь для токена i мы знаем, с какой важностью учитывать каждый другой токен.
5. Формируем контекстный вектор.

$$\mathrm{output}_i = \sum_j \alpha_{ij} V_j$$

Каждый выход $output_i$ — это новый embedding токена i, который уже учитывает весь контекст предложения.

Маскирование (Masking)

Зачем: в авторегрессионных задачах модель не должна смотреть в будущее.

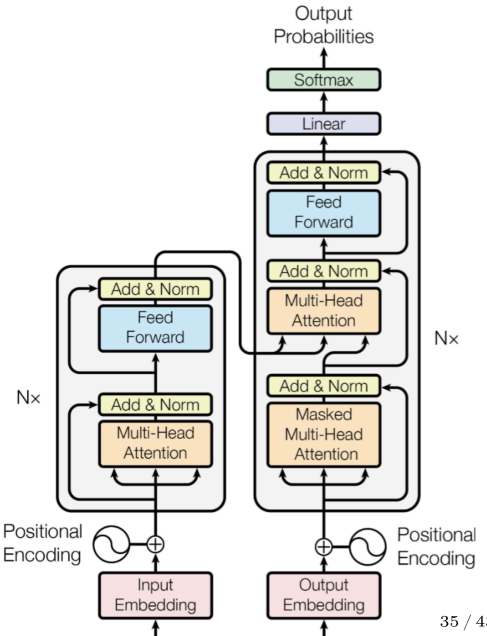
Причинная маска:

$$M_{ij} = \begin{cases} 0, & j \leq i, \\ -\infty, & j > i, \end{cases} \qquad A = \operatorname{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)$$

Элементы $-\infty$ зануляются после softmax — модель видит только прошлые токены.

Transformer: обзор архитектуры

- Без RNN/конв: полностью на механизме внимания — позволяет параллельно обрабатывать всю последовательность.
- Базовые блоки: Multi-Head Self-Attention, Add&Norm, позиционные кодировки, Position-wise FFN.
- Encoder / Decoder — оба состоят из повторяющихся слоёв этих блоков.



Scaled dot-product attention (формула и смысл)

Пусть для набора позиций заданы матрицы запросов, ключей и значений Q, K, V ($Q, K \in \mathbb{R}^{n\times d_k}$, $V \in \mathbb{R}^{n\times d_v}$). Тогда:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V.$$

- $QK^\top$ — попарные скоры между запросами и ключами.
- Деление на $\sqrt{d_k}$ стабилизирует распределение скоров при больших размерностях.
- softmax даёт веса внимания по каждой позиции; итог — взвешенная сумма по значениям V.

Multi-Head self-attention (формулы и мотивация)

$$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q,\ KW_i^K,\ VW_i^V), \qquad \mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}$$

- Каждая голова — собственное линейное проецирование Q/K/V → разные «взгляды» на зависимости.
- Конкатенация + выходная проекция объединяют информацию от всех голов.
- Self-attention = Q,K,V берутся из одного и того же слоя (токен обращается к другим токенам в той же последовательности).

Позиционные кодировки (почему и как)

- Attention сам по себе не учитывает порядок — нужна информация о позиции токена.
- Оригинальный вариант (синусно-косинусные):

$$\mathrm{PE}_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right),$$
$$\mathrm{PE}_{pos,2i+1} = \cos\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right).$$

- Такие кодировки дают постоянные периодические шаблоны, позволяющие моделям аппроксимировать относительный порядок и сдвиги.
- Альтернативы: обучаемые позиционные эмбеддинги, относительные позиционные кодировки.

Position-wise Feed-Forward Network (FFN)

После слоя внимания каждый токен проходит независимую небольшую MLP:

$$\mathrm{FFN}(x) = \mathrm{Activation}(xW_1 + b_1)W_2 + b_2,$$

где обычно Activation = ReLU или GELU.

- FFN применяется отдельно к каждому положению (position-wise), не смешивая информацию между токенами — это даёт нелинейную проекцию признаков.
- Типичный выбор: увеличение размерности в скрытом слое (например, $d_{\mathrm{model}} \to 4d_{\mathrm{model}} \to d_{\mathrm{model}}$).

Add & Norm (Residual + LayerNorm)

В каждом субблоке используется остаточное соединение и нормализация:

$$\text{SublayerOutput} = \text{LayerNorm}(x + \text{Sublayer}(x)).$$

- Residual (skip) соединение ускоряет оптимизацию и помогает градиенту проходить через глубокие слои.
- LayerNorm нормализует по признакам каждого токена независимо (по feature-dimension).
- Нормализация делается по токену, а не по батчу — это критично для NLP (см. следующую страницу).

Почему LayerNorm, а не BatchNorm? (важно)

- В трансформерах нормализуем **каждый токен отдельно по признакам**, а не по всем токенам или батчу.
- Причины:
  - Разная длина последовательностей: нормализация по всем токенам или батчу потребовала бы учитывать паддинги, искажая статистику.
  - Семантика токенов различна: токены — слова/части слов, их значения несопоставимы, поэтому «сквозная» нормализация между токенами бессмысленна.
  - Стабильность при inference: генерация авторегрессивно идёт по одному токену; LayerNorm не зависит от других токенов $\rightarrow$ одинаковое поведение при обучении и при генерации.
  - Независимость от батча: BatchNorm требует статистику по батчу; в NLP батчи часто малы и разноширoки $\rightarrow$ статистика шумная. LayerNorm работает корректно даже при batch$_s ize = 1$.

Короткое резюме: роль компонентов Transformer

- Self-attention — гибкий механизм для установления длинно- и коротко-диапазонных зависимостей, полностью параллельный.
- Multi-head — разные представления/внимания в параллели.
- Позиционные кодировки — вводят порядок в безпорядочную матрицу внимания.
- FFN — проекция на позицию для внесения нелинейности и увеличения выразительности.
- Add&Norm (Residual + LayerNorm) — стабилизация обучения и согласованное поведение при inference.

Заключение

- Прогнозирование временных рядов прошло длинный путь: от интерпретируемых статистических моделей (ARIMA, ETS) к мощным нейросетевым архитектурам (LSTM, Transformer).
- Классические подходы хорошо работают при ограниченных данных, чёткой сезонности и трендах.
- Нейросетевые методы раскрывают потенциал при больших объёмах данных и сложных нелинейных зависимостях.
- Современные трансформеры объединяют гибкость self-attention и параллельность обучения, становясь универсальным инструментом для анализа последовательностей — от текста до временных рядов.
- Направления развития: гибридные модели (ARIMA+NN), энергоэффективные архитектуры и использование контекстных признаков.

Главная идея: модели усложняются, но цель остаётся прежней — *научиться понимать и предсказывать динамику процессов во времени.*