Лекция 7
Введение в глубинное обучение

Е. А. Соколов
ФКН ВШЭ

1 декабря 2025 г.

1 Мотивация

Ранее мы подробно изучили линейные модели классификации и регрессии. Мы выяснили, что они обладают множеством преимуществ: скоростью обучения, интерпретируемостью и простотой реализации. Однако мы также столкнулись с их существенным недостатком - ограниченной выразительной способностью. Линейный классификатор может построить только гиперплоскость, что часто оказывается недостаточным для сложных данных.

В этой лекции мы перейдем к методам глубинного обучения (Deep Learning). Мы попытаемся понять, как можно преодолеть ограничения линейных моделей, выстраивая композиции из простых преобразований, и обсудим основные строительные блоки современных нейронных сетей.

Основная проблема, с которой мы сталкиваемся при использовании линейных моделей на сложных данных (например, изображениях или текстах), заключается в том, что зависимость между признаками и целевой переменной редко бывает линейной. До сих пор у нас было два пути решения этой проблемы:

1. Конструирование признаков (Feature Engineering). Мы могли вручную придумывать новые признаки (полиномиальные, комбинированные и т.п.), чтобы улучшить разделимость классов в пространстве признаков (вплоть до линейной разделимости).

2. Композиции моделей. Использование бустинга или бэггинга над простыми алгоритмами.

Глубинное обучение - это по сути систематизация второго пути: вместо ручной инженерии признаков мы строим модель как последовательность простых преобразований.

Определение глубинного обучения. Построение сложных моделей путем суперпозиции (композиции) простых базовых функций или слоев:

$$a(x) = b_1\big(b_2(\dots b_n(x)\dots)\big),$$

где x - входной объект, а b_i - функции слоя.

Такой подход имеет ряд преимуществ:

- Легкость дифференцирования: благодаря правилу дифференцирования сложной функции (chain rule), мы можем обучать такие конструкции методом градиентного спуска.
- Систематичность: это дает нам подход к построению моделей любой сложности, собирая их как конструктор из готовых блоков.

2 Полносвязные слои (Fully Connected)

2.1 Определение и математическая модель

Базовым строительным блоком нейронных сетей является полносвязный слой. По сути, это уже знакомое нам линейное преобразование, но с несколькими выходами.

Определение полносвязных сетей. Линейное преобразование, которое отображает входной вектор $x \in \mathbb{R}^n$ в вектор $z \in \mathbb{R}^m$. Каждый элемент выходного вектора z_j (где $j = 1, \ldots, m$) представляет собой взвешенную сумму всех входных признаков со сдвигом:

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j.$$

В матричном виде это записывается компактно:

$$z = Wx + b,$$

где $W \in \mathbb{R}^{m \times n}$ - матрица весов, а $b \in \mathbb{R}^m$ - вектор сдвигов. Заметим, что каждый выход является линейной комбинацией **всех** входов.

2.2 Аналогия из биологии

Часто структуру нейронных сетей объясняют через аналогию с биологическим мозгом. Рассмотрим упрощенную математическую модель нейрона:

- Дендриты соответствуют входным сигналам x_i.
- Синапсы моделируются весами w_i, которые могут усиливать или ослаблять сигнал.
- Тело клетки производит суммирование взвешенных сигналов.
- Аксон передает результат дальше следующему нейрону.

Эта модель в точности соответствует одному выходу полносвязного слоя. Однако стоит помнить, что это лишь *грубая математическая абстракция*. Современные представления о биологических нейронах намного сложнее: они описываются дифференциальными уравнениями, обладают временными задержками и сложной химической природой передачи сигнала.

2.3 Проблемы линейности

Если мы попробуем построить сеть, просто накладывая полносвязные слои друг на друга (например, $z = W_2(W_1x + b_1) + b_2$), мы столкнемся с определенной проблемой.

Из линейной алгебры известно, что композиция линейных преобразований сама является линейным преобразованием. Раскрыв скобки, мы получим, что такая *глубокая* сеть эквивалентна одному-единственному полносвязному слою с другими весами. Это возвращает нас к исходной проблеме ограниченной выразительности.

Полносвязные слои также страдают от чрезмерного количества параметров. Для слоя с n входами и m выходами нам нужно хранить $n \times m$ весов, что при работе с высокоразмерными данными приводит к вычислительным сложностям.

3 Нелинейность и функции активации

Чтобы нейронная сеть могла восстанавливать сложные зависимости, необходимо внедрить нелинейность между линейными слоями.

Функция активации - это нелинейная функция $\sigma(z)$, которая применяется поэлементно к выходу линейного слоя:

$$y = \sigma(Wx + b)$$

Рассмотрим наиболее популярные функции активации.

3.1 ReLU (Rectified Linear Unit)

На данный момент это *стандарт по умолчанию* в глубинном обучении.

$$ReLU(x) = \max(0, x)$$

У этой функции активации есть как преимущества, так и недостатки. Сильные стороны:

- Вычислительная эффективность (простое сравнение с нулем).
- Простая производная (0 или 1), что облегчает обучение и спасает от проблемы затухания градиентов.

Слабые стороны:

- Проблема *умирающих нейронов* (Dying ReLU). Если вход нейрона уходит в отрицательную область, градиент становится равным нулю, и веса перестают обновляться.

3.2 Swish

Есть несколько вариаций, как можно улучшить свойства ReLU. Например, Swish.

$$Swish(x) = x \cdot \sigma(\beta x),$$

где σ - сигмоида, а β - обучаемый параметр.

В отличие от ReLU, Swish является гладкой функцией и не равна нулю при отрицательных значениях (при малых β), что позволяет градиентам протекать даже через отрицательные активации.

4 Теорема об универсальной аппроксимации

Возникает естественный вопрос: насколько мощным инструментом являются нейронные сети?

В этом вопросе мы обратимся к теореме Цыбенко (1989): для любой непрерывной функции $f : [0,1]^d \to \mathbb{R}$ и любого $\epsilon > 0$ существует двухслойная нейронная сеть (с одним скрытым слоем) и подходящей нелинейной функцией активации, которая приближает f с точностью ϵ равномерно на $[0,1]$.

Это фундаментальный результат, утверждающий, что нейронные сети являются *универсальными аппроксиматорами*. Теоретически, даже сеть с одним скрытым слоем может выучить любую зависимость.

Теорема гарантирует *существование* такой сети, но ничего не говорит о том, насколько широким должен быть скрытый слой. На практике ширина слоя может потребоваться экспоненциально большой. Именно поэтому мы используем *глубокие* (многослойные) сети: глубина позволяет строить сложные иерархические признаки более экономно с точки зрения количества параметров.

Итак, нам нужны архитектурные идеи, которые (а) уменьшают число параметров и (б) встраивают априорную структуру данных. Для изображений и спектрограмм таким следующим шагом становятся имеено свёрточные слои.

5 Сверточные нейронные сети

Несмотря на универсальность полносвязных слоев, при работе с изображениями они сталкиваются с проблемой потери пространственной структуры (вытягивание изображения в вектор) и колоссальным количеством параметров:

1. Если у слоя n входов и m выходов, то параметров $n \times m$, а при больших n, m вычисления становятся дорогими.

2. Полносвязный слой одинаково смешивает любые координаты входа, хотя в изображениях важна локальность, ведь соседние пиксели явно связаны сильнее, чем дальние.

Решением являются *сверточные слои* (Convolutional Layers), которые учитывают локальную топологию данных.

5.1 Операция свертки

Свертка - это операция, применяющая фильтр (ядро) к входному изображению для выделения определенных паттернов. Формально для двумерного случая:

$$Out(x,y) = \sum_{i=-d}^{d} \sum_{j=-d}^{d} In(x+i, y+j) \cdot K(i,j) + b,$$

где K - фильтр (матрица весов) размера $(2d+1) \times (2d+1)$, а b - смещение.

Интуитивно это означает сканирование изображения *окном фильтра*. Один и тот же фильтр скользит по входу и ищет один конкретный паттерн в разных местах: где паттерн совпал - там выход большой.

В классическом компьютерном зрении фильтры (например, Собеля) задавались вручную. В глубоком обучении значения $K(i, j)$ являются обучаемыми параметрами: сеть сама придумывает, какие признаки ей нужно искать.

Техническое замечание. Во многих библиотеках под "convolution"на практике реализуется *корреляция* (без переворота ядра). Для обучения это почти не важно: параметры ядра всё равно подстраиваются.

5.2 Обобщение на многоканальные изображения

Реальные изображения (RGB) и скрытые слои имеют множество каналов. Введем определение многоканальной свертки.

Пусть вход имеет C каналов. Тогда каждый фильтр является трехмерным тензором глубины C, а результат вычисляется как сумма сверток по всем каналам:

$$Out(x, y, t) = \sum_{c=1}^{C} \sum_{i=-d}^{d} \sum_{j=-d}^{d} In(x + i, y + j, c) \cdot K(i, j, c, t) + b_t,$$

где $t = 1, \ldots, T$ — номер выходного фильтра (канала).

Важно: фильтр всегда имеет ту же глубину C, что и вход. Количество выходных каналов T определяется количеством фильтров, которое мы задаем при проектировании.

Количество параметров. Для сверточного слоя с фильтрами размера $(2d+1) \times (2d+1)$, C входными каналами и T выходными фильтрами число параметров равно:

$$N_{params} = T \times (C \times (2d + 1)^2 + 1)$$

Пример: для $C = 3$, фильтров 3×3 и $T = 1000$ получаем всего $28,000$ параметров, что на порядки меньше, чем в полносвязных слоях. Ключевое: число параметров зависит от размера ядра и числа каналов, а не от размера изображения целиком.

5.3 Свойства сверточных сетей

Эффективность CNN обусловлена следующими моментами:

1. Локальная связность. Каждый выходной нейрон смотрит только на небольшую область входа. Это соответствует природе изображений, где зависимости локальны.

2. Локальная связность и разделение весов представляют собой форму индуцированного смещения (inductive bias) - встраивания априорных знаний о структуре данных в архитектуру модели

3. Разделение весов (Shared Weights). Один и тот же фильтр применяется ко всему изображению. Это обеспечивает инвариантность к сдвигу (объект будет найден, где бы он ни находился).

5.4 Гиперпараметры свёрточных слоёв

Основные параметры, управляющие геометрией свертки:

- Stride - шаг перемещения фильтра. Фактически, как часто мы сдвигаем ядро. Увеличение stride уменьшает размер выхода
- Padding (дополнение). Добавление нулей по краям входа (Zero-padding) позволяет сохранить пространственный размер изображения после свертки.
- Dilation (прореживание ядра). Пропуск пикселей при применении фильтра для увеличения области обзора без роста числа параметров.

Глянем на формулу (размер выхода, 1D/2D по компонентам) для одной пространственной оси:

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} + 2P - \text{dil} \cdot (K-1) - 1}{S} \right\rfloor + 1,$$

где K - размер ядра, S - stride, P - padding, dil - dilation.

5.5 Пулинг (Pooling) и сжатие карт признаков

Пулинг - это детерминированная операция агрегации по локальному окну, которая уменьшает пространственное разрешение карт признаков. Выделяем два типа пуллинга:

- MaxPool: берём максимум в каждом окне $k \times k$ (часто 2×2 со stride 2).
- AvgPool: берём среднее.

Зачем нам это в целом? Мы уменьшаем размерность, а значит, вычислений дальше становится меньше. Также делаем представление более устойчивым к малым сдвигам/деформациям. Нам не так важна точная позиция, важен сам факт наличия некого паттерна.

Это один из способов расширять область восприятия следующих слоёв: после нескольких шагов свёртка начинает видеть всё большие фрагменты исходного сигнала.

На практике во многих архитектурах пулинг частично заменяют свёртками с stride > 1: идея та же - сжать разрешение, - но параметры обучаемые.

5.6 Проблемы обучения глубоких сетей

При обучении глубоких сетей с многими слоями часто возникают следующие проблемы: медленная сходимость или отсутствие сходимости и нестабильность градиентов.

Здесь незаменимой становится гипотеза внутреннего ковариационного сдвига (Internal covariate shift) - это предположение, что изменение распределения активаций во внутренних слоях сети в процессе обучения затрудняет обучение последующих слоев, которые оказываются неадаптированными к новым распределениям.

6 Batch Normalization

Batch Normalization - это метод нормализации активаций внутренних слоев, вычисляемый по мини-батчам данных во время. Изначально батч-нормализацию мотивировали именно гипотезой внутреннего ковариационного сдвига: если распределения активаций во внутренних слоях постоянно плывут, то последующим слоям тяжелее адаптироваться. Однако важно понимать, что современное (более аккуратное) объяснение эффективности BN не сводится только к этой гипотезе. Нормализация также стабилизирует и сглаживает процесс оптимизации - это мы обсудим детальнее в следующем разделе.

Батч-нормализация масштабирует каждый признак на выходе слоя, вычитая среднее и деля на стандартное отклонение с последующей настройкой параметров γ и β.

Пусть на некотором слое мы получаем значения активаций x_{ij}, где $i = 1, \ldots, N$ - индекс объекта в мини-батче, а $j = 1, \ldots, d$ - индекс признака (координаты). Batch normalization нормализует каждый признак по батчу:

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}, \qquad \sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \mu_j)^2,$$

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}, \qquad y_{ij} = \gamma_j \hat{x}_{ij} + \beta_j,$$

где $\varepsilon > 0$ - малая константа для численной устойчивости, а γ_j, β_j - обучаемые параметры масштаба и сдвига.

В сверточных сетях batch normalization обычно применяют по каналам: среднее и дисперсия считаются по оси батча и по всем пространственным координатам внутри канала. Это сохраняет согласованность нормализации с топологией данных (внутри канала признаки однотипны).

6.1 Переосмысление Batch Normalization

Мы уже ввели понятие Batch Normalization (BN) как метода нормализации активаций, который позволяет обучать глубокие сети. Однако понимание причин его эффективности со временем изменилось.

Первоначальная статья, предложившая BN, утверждала, что метод решает проблему Internal Covariate Shift (ICS). Суть гипотезы была в том, что в процессе обучения параметры предыдущих слоев меняются, из-за чего меняется распределение входов для последующих слоев. Глубоким слоям приходится постоянно подстраиваться под новые статистики входов, что замедляет обучение.

Спустя несколько лет исследователи провели серию экспериментов, которые опровергли эту гипотезу. В одном из экспериментов после каждого слоя BN специально добавляли шум с ненулевым средним и меняющейся дисперсией, искусственно создавая сильный ковариационный сдвиг. Оказалось, что такая зашумленная сеть с BN все равно обучается лучше и быстрее, чем обычная сеть без BN.

Более того, замеры показали, что BN не стабилизирует, а иногда даже увеличивает изменение градиентов в глубоких слоях. Это означает, что борьба со сдвигом распределений не является главной причиной успеха метода.

Настоящий механизм работы batch normalization заключается в упрощении функционала ошибки - он сглаживает ландшафт функции потерь (Loss Landscape Smoothing).

Эксперименты по анализу поведения функции потерь вдоль направления градиента показали:

- Без batch normalization наблюдаются значительные колебания значения функции. потерь
- С batch normalization функционал становится более гладким и предсказуемым.
- Углы между градиентами в соседних точках становятся более стабильными.

Это упрощение ландшафта функции ошибки облегчает процесс градиентного спуска и ускоряет обучение.

6.2 Другие нормализации

BatchNorm завязан на статистики батча, поэтому в некоторых режимах (малые батчи, переменная длина последовательностей и т.п.) удобнее использовать альтернативы.

Например, Layer Normalization. LayerNorm нормализует внутри одного объекта по координатам признакового вектора:

$$\mu = \frac{1}{d}\sum_{j=1}^{d} x_j, \qquad \sigma^2 = \frac{1}{d}\sum_{j=1}^{d}(x_j - \mu)^2, \qquad y_j = \gamma_j \frac{x_j - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta_j.$$

Статистики не зависят от других объектов, поэтому метод стабильно работает при любом размере батча.

Еще есть вариация с Group Normalization. Это такой промежуточный вариант: каналы делятся на группы, и нормализация выполняется внутри каждой группы каналов. Это полезно в сверточных сетях при малых батчах, когда BatchNorm становится шумным.

Отдельно стоит рассмотреть RMSNorm. Эта форма нормализует по корню из среднего квадрата (без вычитания среднего):

$$\mathrm{RMS}(x) = \sqrt{\frac{1}{d}\sum_{j=1}^{d} x_j^2 + \varepsilon}, \qquad y_j = \gamma_j \frac{x_j}{\mathrm{RMS}(x)}.$$

На практике это часто используют в архитектурах трансформеров: метод дешевле по вычислениям и достаточно стабилен.

7 Аугментация данных

Для задач компьютерного зрения, особенно при работе с ограниченными наборами данных, важную роль играет *аугментация* - техника искусственного расширения выборки за счет преобразований исходных изображений. Типичные аугментации для изображений:

- случайные повороты, сдвиги, масштабирование, отражения;
- изменения яркости/контраста/насыщенности;

- добавление шума, небольшие размытия.

Ключевое требование здесь следующее: преобразование не должно менять *семантический класс* объекта. В таком случае аугментация действует как регуляризация: модель вынуждена быть инвариантной к допустимым изменениям входа.

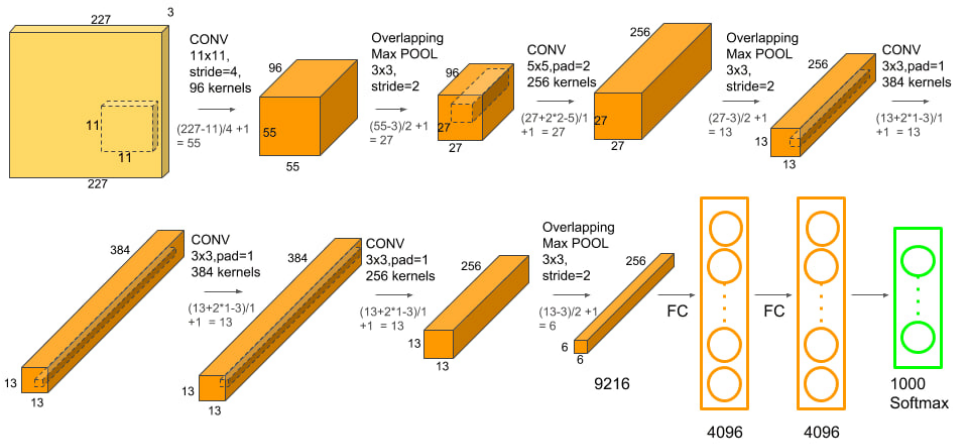
8 Архитектуры сверточных сетей

Прорыв в глубоком обучении для компьютерного зрения начался с создания набора данных ImageNet (2011-2012 гг.), содержащего около миллиона изображений, размеченных по 1000 классам. Это позволило проводить масштабные сравнительные исследования.

8.1 AlexNet

Первой успешной архитектурой стала **AlexNet** (названа в честь Алекса Ижевского). Основные характеристики:

- Вход: цветные изображения $224 \times 224 \times 3$ (RGB).
- Последовательность сверточных слоев с фильтрами 11×11 и 5×5.
- Максимальное пулинговые слои (max pooling).
- В конце - вытягивание в вектор и несколько полносвязных слоев.
- Активация ReLU, оптимизация с моментумом.
- 60 миллионов параметров.
- Ошибка top-5: около 17% (против 25% у классических методов).



Исторический смысл AlexNet - показать, что глубокая CNN при корректной оптимизации и данных большого масштаба радикально превосходит классические подходы того времени.

8.2 VGG Network

Архитектура VGG от Visual Geometry Group (Оксфорд) предложила ключевые улучшения - глубину через маленькие фильтры. Это простой, но мощный дизайн:

- использовать почти исключительно свертки 3×3;
- наращивать глубину сети (в известных конфигурациях - до 16-19 слоев).

Малые фильтры позволяют строить много нелинейных преобразований подряд и постепенно увеличивать область восприятия (receptive field), не раздувая параметры слишком быстро. Это повышает выразительную способность модели. Однако обучение таких глубоких архитектур потребовало решения проблем сходимости.

Глубокая архитектура VGG с 19 слоями столкнулась с проблемами обучения: сеть не сходилась или демонстрировала плохое качество. В качестве временного решения использовался костыльный подход. Сначала обучали урезанную версию модели с меньшим числом слоев, затем постепенно добавляли слои, инициализируя их случайно и дообучая сеть. С появлением batch normalization эта проблема была решена, глубокие сети стали обучаться с нуля без дополнительных ухищрений.

Глубокая архитектура VGG достигла ошибки top-5 около 8% с использованием композиции моделей, что значительно превосходит результаты AlexNet. Ошибка top-5 означает, что в 92% случаев правильный класс находится среди пяти наиболее вероятных классов по мнению модели.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

8.3 ResNet: революция остаточных связей

При обучении очень глубоких сетей (50+ слоев) возникла проблема деградации. Качество падало даже на обучающей выборке из-за затухания градиентов.

При увеличении глубины сети иногда наблюдается парадокс: качество ухудшается даже на обучающей выборке. Это не похоже на переобучение - это похоже на то, что оптимизация перестала справляться.

Окей, а что если добавть остаточные (residual) связи, которые бы пропускали сигнал в обход нескольких слоев? Это и стало ключевой идеей ResNet. Давайте введем Skip Connection (ту самую остаточную связь):

$$H(x) = F(x) + x,$$

где x - вход блока, $F(x)$ - преобразование (обычно несколько сверток + нелинейности), а $H(x)$ - выход блока.

Вместо того чтобы учить всю функцию целиком, блок учит *поправку* к тождественному отображению. Это облегчает распространение информации и градиента по сети и позволяет эффективно обучать очень глубокие архитектуры.

ResNet позволил обучать сети глубиной в 152 и даже 1000 слоев, снизив ошибку на ImageNet до 4.5%. Также в ResNet отказались от полносвязных слоев в конце, заменив их на Global Average Pooling, или свертки со страйдом, что резко сократило число параметров.

Список литературы

[1] Goodfellow, I., Bengio, Y., Courville, A. *Deep Learning.* // MIT Press, 2016.

[2] Zhang, A., Lipton, Z. C., Li, M., Smola, A. J. *Dive into Deep Learning.* // Online book.

[3] Stanford CS231n: *Convolutional Neural Networks for Visual Recognition.* // Course notes.

[4] Krizhevsky, A., Sutskever, I., Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks.* // NeurIPS, 2012.

[5] He, K., Zhang, X., Ren, S., Sun, J. *Deep Residual Learning for Image Recognition.* // CVPR, 2016.

[6] Ioffe, S., Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* // ICML, 2015.

[7] Santurkar, S., Tsipras, D., Ilyas, A., Madry, A. *How Does Batch Normalization Help Optimization?* // NeurIPS, 2018.

[8] Ba, J. L., Kiros, J. R., Hinton, G. E. *Layer Normalization.* // 2016.

[9] Wu, Y., He, K. *Group Normalization.* // ECCV, 2018.

[10] Zhang, B., Sennrich, R. *Root Mean Square Layer Normalization.* // NeurIPS, 2019.